Aus Fehlern in der Softwareentwicklung lernen

Wie durch Fehleranalysen die Prozesse der Anforderungs-analyse und der Qualitätssicherung verbessert werden können

Inauguraldissertation zur Erlangung des Doktorgrades
der Wirtschafts- und Sozialwissenschaftlichen Fakultät
der Universität zu Köln

2007

vorgelegt von

Diplom-Wirtschaftsinformatiker Oral Avcı

aus Köln

Referent: Prof. Dr. W. Mellis

Korreferent: Prof. Dr. D. Seibt

Promotionsdatum: 12.02.2008

Kapitelübersicht

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis:

AK+SNT	Anwendungskern und Schnittstellen
AS	Ausgangssuche
ASM	International Conference on Applications of Software Measurement
CACM	Communications of the ACM
CAD	computer-aided design
CAE	computer-aided engineering
CAIS	Communications of the AIS
CAM	computer-aided manufacturing
CASE	Computer-Aided Software Engineering
CMM	Capability Maturity Model
CMMi	Capability Maturity Model Integrated
DCCA	IFIP Working Conference on Dependable Computing for Critical Applications
DSN	International Conference on Dependable Systems and Networks
EJIS	European Journal of Information Systems
ERP	Enterprise-Resource-Planning
ESE	Empirical Software Engineering
FTCS	International Symposium on Fault Tolerant Computing
GQM	Goal Question Metric
HP	Hewlett Packard
i. e. S.	im engeren Sinn
i. w. S.	im weiteren Sinn
IBMSJ	IBM Systems Journal
ICIS	International Conference on Information Systems
ICRE	IEEE International Conference on Requirements Engineering
ICSE	International Conference on Software Engineering
ICSM	IEEE International Conference on Software Maintenance
IEEESw	IEEE Software
IEEETSE	IEEE Transactions on Software Engineering
IFIP	International Federation for Information Processing
ISESE	International Symposium on Empirical Software Engineering
ISJ	Information Systems Journal

ISO	International Organization for Standardization
ISR	Information Systems Research
ISRE	IEEE International Symposium on Requirements Engineering
ISS	Interational Space Station
ISSN	International Standard Serial Number
ISSRE	International Symposium on Software Reliability Engineering
JAIS	Journal of the Association for Information Systems
JMIS	Journal of Management Information Systems
METRICS	International Symposium on Software Metrics
MISQ	MIS Quarterly
MS	Management Science
NASA	National Aeronautics and Space Administration
NBA	National Basketball Association
NIST	National Institute of Standards and Technology
o. S.	ohne Seitenangaben
ODC	Orthogonal Defect Classification
PDM	product data management
PDP	Produktdaten und Plausibilitätsbedingungen
PSQM	prozessorientiertes Softwarequalitätsmanagement
QS	Qualitätssicherung
RE	Requirements Engineering
RS	Rückwärtssuche
SEL	Software Engineering Laboratory
SQJ	Software Quality Journal
UML	Unified Modeling Language
VS	Vorwärtssuche
WIRT	Wirtschaftsinformatik

1 Gegenstand und methodisches Konzept der Arbeit

1.1 Problemstellung

1.1.1 Aus Fehlern lernen

„When I was growing up, my father was fond of using sayings to encourage me to remember important lessons. One of his favorites was ‚Do you know the difference between a wise man and a fool?' He would then go on to say that a wise man makes a mistake only once. A fool makes the same mistake over and over again."[1]
Robert B. Grady

Softwarefehler[2] gibt es, seit Menschen Software entwickeln. Sie können mitunter zu erheblichen wirtschaftlichen Verlusten und im schlimmsten Fall zum Verlust von Leben führen. So waren es Softwarefehler, die dazu geführt haben, dass das Strahlentherapiegerät Therac-25 Mitte der achtziger Jahre Patienten verstrahlte und infolge dessen einige von ihnen verstarben.[3] Es waren Softwarefehler, die dazu geführt haben, dass Ariane 5 1996 wenige Sekunden nach dem Start explodierte.[4] Es waren Softwarefehler, die 2003 wesentlich zum Stromausfall in weiten Teilen der USA und Kanada beitrugen.[5]

Die vorliegende Arbeit beschreibt, wie aus Fehlern in der Softwareentwicklung gelernt werden kann, um gemäß Gradys Anekdote zu vermeiden, dass sich die gleichen, aber auch ähnliche Fehler wiederholen. Aus Fehlern zu lernen, darunter wird in dieser Arbeit insbesondere verstanden, dass auf Basis einer Fehleranalyse die Art und Weise verbessert wird, wie Softwareanforderungen erhoben und aufbereitet werden, also kurz der Prozess der Anforderungsanalyse[6]. Anforderungsfehler und mögliche Folgefehler können damit vermieden werden. Zusätzlich besteht die Möglichkeit aus Fehlern zu lernen, indem der Prozess der Quali-

---

1 Grady /Software failure analysis/ 155.

2 Vgl. zum Begriff eines Fehlers Kapitel 2.2.5.

3 Vgl. Levenson, Turner /Therac-25/ 18-41.

4 Vgl. Ben-Ari /Bug/ 58 f.

5 Vgl. U.S.-Canada Power System Outage Task Force /Causes/ 93-99.

6 In der angloamerikanischen Literatur wird häufig die Bezeichnung „requirements engineering" für die Anforderungsanalyse verwendet.

tätssicherung verbessert wird. Entstandene Fehler können so früher gefunden und weitere Folgefehler vermieden werden.

Eine wissenschaftliche Arbeit muss präzise die ihr zu Grunde liegende Problemstellung darlegen, da wissenschaftliches Handeln auf das Lösen von Problemen ab zielt.[7] Die Abbildung 1-1 stellt den Aufbau der Problemstellung dieser Arbeit zusammenfassend dar.

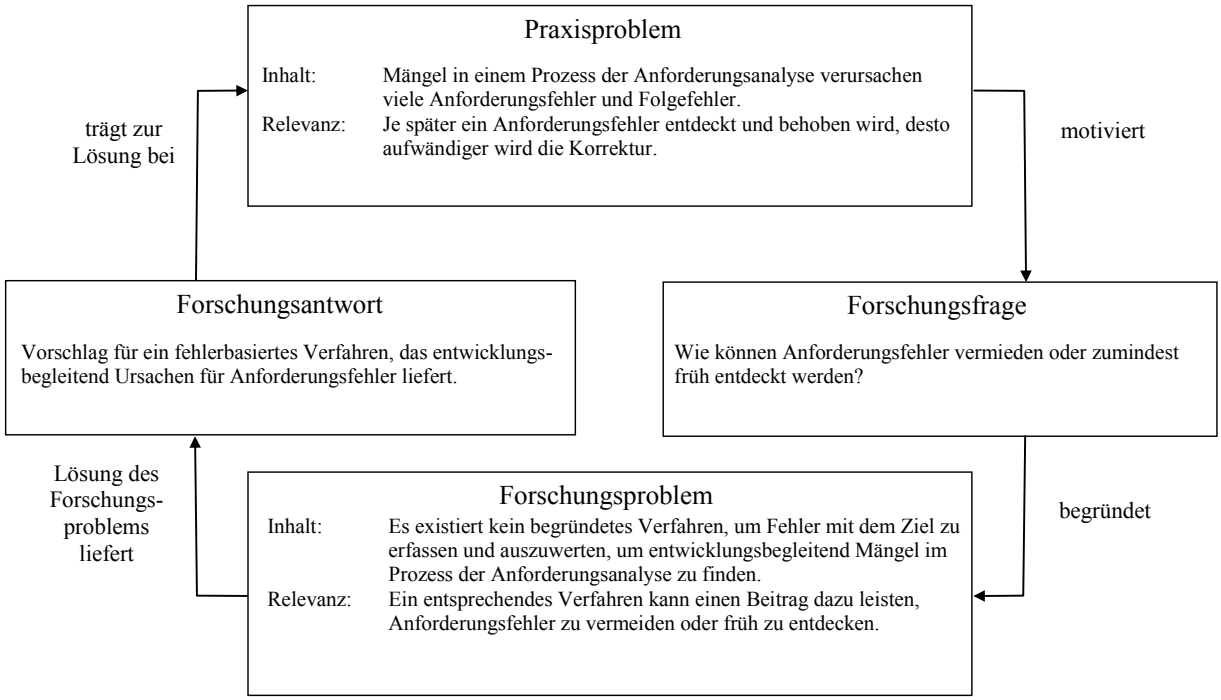


Abbildung 1-1: Zusammenfassung der Problemstellung[8]

Viele Fehler sind zurückzuführen auf Mängel im Prozess der Anforderungsanalyse (*Inhalt des Praxisproblems*).[9] Werden beispielsweise bei der Entwicklung einer Anwendungssoftware die zu unterstützenden Arbeitsabläufe nicht ausreichend untersucht bevor die Softwareanforderungen erhoben werden, kann dies ein Mangel im Prozess der Anforderungsanalyse sein; ein Prozessmangel, infolge dessen Benutzer und Entwickler Softwareanforderungen übersehen oder unterschiedlich verstehen. Er kann also zu Anforderungsfehlern führen. Dass Prozessmängel viele Anforderungs- und Folgefehler verursachen, ist ein unbefriedigender Zustand

---

[7] Vgl. Booth, Colomb, Williams /Research/ 57-60.

[8] Der Aufbau der Problemstellung ist angelehnt an Booth, Colomb, Williams /Research/ 57-64. Die drei Autoren empfehlen eine Struktur für wissenschaftliche Problemstellungen, die explizit zwischen einem Praxis- und Forschungsproblem unterscheidet. Diese Struktur wurde in dieser Arbeit übernommen. Sie trägt nach Überzeugung des Verfassers dazu bei, dass die praktische Relevanz wissenschaftlicher Arbeiten nicht vernachlässigt wird; eine praktische Relevanz wie sie von Benbasat und Zmud für die Wirtschaftsinformatik gefordert und vermisst wird. Vgl. Benbasat, Zmud /Practice of relevance/ 3-14.

[9] Vgl. hierzu die empirischen Befunde in Kapitel 3.2.

und bedeutsam zugleich, da die Korrektur eines Anforderungsfehlers aufwändiger wird, je später er entdeckt und behoben wird (*Relevanz des Praxisproblems*).[10]

Das Praxisproblem motiviert die Frage, wie Anforderungsfehler vermieden oder zumindest früh gefunden werden können (*Forschungsfrage*). Mit dieser Forschungsfrage wird ein Forschungsproblem begründet. Es fehlt bislang ein Verfahren, mit dem begleitend zu einem Softwareentwicklungsvorhaben[11] Fehler erfasst und ausgewertet werden können, um die Ursachen für die Anforderungsfehler und damit auch für ihre Folgefehler im Prozess der Anforderungsanalyse zu ermitteln (*Forschungsproblem*). Wird dieses Forschungsproblem gelöst, erhält man zugleich eine Antwort auf die Forschungsfrage (*Forschungsantwort*). Sobald aus Fehlern auf die Ursachen im Prozess geschlossen wird, beginnt man aus Fehlern zu lernen. Sind nämlich einmal die Ursachen im Prozess bekannt, können sie behoben werden, sofern möglich, um damit vergleichbare Fehler zukünftig zu vermeiden.[12] Ebenso ermöglicht die Kenntnis der Ursachen gezielt nach entstandenen Fehlern zu suchen, um sie früher zu finden. Wird die Forschungsantwort, also das hergeleitete Verfahren, in der Praxis angewandt, kann sie dazu beitragen, das Praxisproblem zu lösen.

In Anlehnung an Booth, Colomb, Williams[13] wird in Kapitel 1.1.2 das Praxisproblem dieser Arbeit begründet. Forschungsfrage und Forschungsproblem werden in Kapitel 1.1.3 behandelt.

1.1.2 Praxisproblem

Softwareanforderungen beschreiben funktionale und qualitätsbezogene Merkmale einer Software unabhängig von ihrer technischen Umsetzung.[14] Im Mittelpunkt der Anforderungsanalyse steht also die Frage, *was* in einem Softwareentwicklungsprojekt erstellt werden soll.[15] Die Beantwortung dieser Frage gehört zu den wichtigsten Aufgaben in der Softwareentwicklung. Keine andere Aufgabe hat einen größeren Einfluss auf den Erfolg oder Misserfolg eines Softwareentwicklungsvorhabens.[16]

---

[10] Vgl. hierzu die empirischen Befunde in Kapitel 3.3.

[11] Die Begriffe Softwareentwicklungsvorhaben, Softwareentwicklungsprojekt, Softwareprojekt und Projekt werden in dieser Arbeit synonym verwendet, sofern nicht explizit hiervon abgewichen wird.

[12] Vgl. hierzu und zum nächsten Satz Jacobs /Defect causes/ 400 f.

[13] Vgl. Booth, Colomb, Williams /Research/ 57-64.

[14] Der Begriff der Softwareanforderung sowie weitere zentrale Begriffe werden in Kapitel 2.2.1 erörtert.

[15] Die Teilaufgabe der Anforderungsanalyse wird in Kapitel 2.3 definiert und abgegrenzt.

[16] Diese These wird prägnant beschrieben in Brooks /Man-month/ 199. Empirisch wird diese These gestützt durch z. B. Krishnan u. a. /Productivity/ 753, Blackburn, Scudder, Van Wassenhove /Speed/ 883 f. und Bowen, Heales, Vongphakdi /Reliability factors/ 201-203.

Seit Mitte der siebziger Jahre wird die Anforderungsanalyse als eine eigenständige Teilaufgabe der Softwareentwicklung wahrgenommen.[17] Seitdem wurden zahlreiche Vorschläge zum Prozess der Anforderungsanalyse veröffentlicht, also zu der Frage, wie entweder bestimmte Aspekte der Anforderungsanalyse oder die Anforderungsanalyse insgesamt durchgeführt werden können.[18] Trotz zahlreicher Fortschritte weist der Prozess der Anforderungsanalyse in der Praxis weiterhin häufig erhebliche Mängel auf.[19] Diese Prozessmängel können zu Anforderungsfehlern führen, wie zum Beispiel zu übersehenen oder falsch verstandenen Anforderungen.[20] Mehrere empirische Untersuchungen zeigen, dass die Praxis der Anforderungsanalyse durch ein zentrales Problem gekennzeichnet wird:

> *Mängel in einem Prozess der Anforderungsanalyse verursachen viele Anforderungsfehler und Folgefehler*.[21]

In vielen dieser Untersuchungen bilden Anforderungsfehler einschließlich ihrer Folgefehler die größte Teilmenge und können bis zu 50% aller Fehler ausmachen.[22] Dieses Problem ist für die Praxis von großer Bedeutung, da Anforderungen die wesentliche Grundlage für weitere Entwicklungsentscheidungen bilden. Deshalb kann ein Anforderungsfehler zu zahlreichen Folgefehlern im Entwurf und der Implementierung führen, sofern er nicht rechtzeitig entdeckt wird. Je später ein Anforderungsfehler entdeckt wird, desto mehr Folgefehler können tendenziell entstehen. Der Aufwand für die Korrektur dieser Folgefehler muss dem Korrekturaufwand des Anforderungsfehlers zugerechnet werden, da sie durch ihn verursacht wurden. Folglich steigt mit der Zahl der Folgefehler auch der erforderliche Überarbeitungsaufwand für einen Anforderungsfehler.[23]

Werden fehlerbehaftete Arbeitsergebnisse einer Softwareentwicklung nicht überarbeitet, erhalten Kunden eine Software, welche ihren Wünschen nicht entspricht. In der Regel werden dann aufwändige Nachbesserungen notwendig, nachdem die Software eingeführt wurde. Werden die Arbeitsergebnisse hingegen während einer Softwareentwicklung überarbeitet, erfor-

---

[17] Vgl. Boehm /Requirements Engineering/ 255.

[18] Siehe hierzu z. B. die Beiträge in den Konferenzbänden der IEEE International Requirements Engineering Conference und der im Springer Verlag erscheinenden Zeitschrift Requirements Engineering.

[19] Vgl. z. B. Hall, Beecham, Rainer /Requirements/ 153-160 und Kamsties, Hörmann, Schlich /Requirements Engineering/ 84-90.

[20] Vgl. hierzu Kapitel 4.3.

[21] Dieses Praxisproblems wird in Kapitel 3.2 auf der Grundlage empirischer Quellen detailliert beschrieben.

[22] Vgl. z. B. Basili, Perricone /Software errors/ 42-52 und Lauesen, Vinter /requirement defects/ 37-50.

[23] Die empirischen Erkenntnisse über den Aufwand zur Korrektur von Anforderungsfehlern werden in Kapitel 3.3 beschrieben.

dert die zusätzlichen Aufwand und Zeit. In der Praxis verursachen vermeidbare Überarbeitungen (rework) einen großen Anteil des Projektaufwands.[24]

Die Relevanz des Praxisproblems lässt sich zusammenfassend mit der These formulieren:

> *Je später ein Anforderungsfehler entdeckt und korrigiert wird, desto aufwändiger wird die Korrektur.*

Die Folgen sind bekannt: Software wird zu spät ausgeliefert, der Aufwand und damit die Kosten für die Entwicklung übersteigen häufig das ursprünglich Geplante um ein Vielfaches und die ausgelieferte Software erfüllt die Kundenbedürfnisse nicht.[25]

1.1.3 Forschungsfrage und Forschungsproblem

Das Praxisproblem zeigt auf, dass zum einen die Zahl der Anforderungs- und Folgefehler in Softwareentwicklungsvorhaben hoch ist und dass zum anderen der Aufwand für die Korrektur eines Anforderungsfehlers mit Fortschreiten des Projekts steigt. Damit wirft das Praxisproblem folgende grundlegende Forschungsfrage auf:

> *Wie können Anforderungsfehler und Folgefehler vermieden oder zumindest früh entdeckt werden?*

Anforderungsfehler können vermieden werden, wenn ihre Ursachen beseitigt werden.[26] Hierzu ist zu klären, welcher Aspekt der Durchführung der Anforderungsanalyse begünstigt hat, dass Anforderungsfehler entstehen. D. h., man muss erst die Mängel im Prozess der Anforderungsanalyse kennen, um sie beheben zu können. Anforderungsfehler vermeiden heißt folglich, den Prozess der Anforderungsanalyse zu verbessern. Diese Vorgehensweise geht von der Annahme aus, dass Investitionen in die Vermeidung von Fehlern, insbesondere von Anforderungsfehlern, geringer sind als Investitionen in Maßnahmen der Qualitätssicherung, um diese Fehler zu entdecken. Dieser Zusammenhang kann insbesondere dadurch begründet werden, dass mit einem besseren Verständnis der Anforderungen weniger Anforderungsfehler und damit auch weniger Folgefehler entstehen, so dass auch weniger fehleranfällige und aufwändige Überarbeitungen erforderlich werden.[27] Befunde mehrerer empirischer Studien zeigen, dass

---

[24] Vgl. Shull u. a. /defects/ 252 und Dion /Process Improvement/ 32

[25] Vgl. The Standish Group /CHAOS/ 1-9.

[26] Vgl. Card /Mistakes/ 58 f.

[27] Vgl. hierzu z. B. Krishnan u. a. /Productivity/ 747, Blackburn, Scudder, Van Wassenhove /Speed/ 883 f. und Kamsties, Hörmann, Schlich /Requirements Engineering/ 89.

Investitionen in die Anforderungsanalyse der Softwareentwicklung die Produktivität erhöhen und die Qualität der Software verbessern.[28]

In bestimmten Fällen ist die Behebung der Fehlerursache jedoch nicht möglich oder mit erheblichen Nachteilen verbunden, so dass sie ausbleibt. So werden in der Praxis zum Beispiel Anforderungen häufig natürlich-sprachlich beschrieben.[29] Aufgrund der Einschränkungen der natürlichen Sprache nimmt man damit in Kauf, dass Anforderungen mehrdeutig oder missverständlich dokumentiert werden können. Dies geschieht, obwohl ausgereifte formalere Beschreibungssprachen vorliegen, durch die dieses Risiko reduziert werden könnte. Dieser offensichtliche Mangel im Prozess der Anforderungsanalyse wird nicht behoben, weil Benutzer mit den Ergebnissen solcher formaleren Anforderungsbeschreibungen in der Regel überfordert sind.[30]
Sofern Anforderungsfehler bereits entstanden sind oder, wie oben beschrieben, nicht vermieden werden können, ist es hilfreich, sie so früh wie möglich zu entdecken. „Früh“ impliziert hierbei nicht eine bestimmte Zeitspanne, sondern bedeutet vielmehr, dass die Zahl der Folgefehler möglichst klein bleibt. Je früher ein Anforderungsfehler entdeckt wird, desto weniger Folgefehler können entstehen. Somit kann der Korrekturaufwand für einen Anforderungsfehler in Grenzen gehalten werden. Die Qualitätssicherung verfolgt das Ziel, Fehler zu entdecken. Anforderungsfehler früh zu finden bedeutet folglich, den Prozess der Qualitätssicherung zu verbessern. Auch hierzu ist die Kenntnis der Ursache von Anforderungsfehlern hilfreich, um mit gezielten Prüf- und Testkriterien nach entstandenen Anforderungs- und deren Folgefehlern zu suchen.

Ausgehend von der Forschungsfrage könnten nun weitere Vorschläge für die Verbesserung des Prozesses der Anforderungsanalyse oder der Qualitätssicherung erarbeitet werden. Jedoch mangelt es nicht an guten Vorschlägen in diesen Bereichen. Vielmehr bereitet die Übertragung dieser Vorschläge auf die Praxis Schwierigkeiten, was sich als ein Problem des Technologietransfers beschreiben lässt.[31] Ein Praktiker steht vor der Herausforderung, aus der Vielzahl von existierenden Vorschlägen jene auszuwählen, die zweckmäßig sind, um die Ursachen für Anforderungsfehler in seinem Softwareentwicklungsvorhaben mit diesen spezi-

---

[28] Vgl. Krishnan u. a. /Productivity/ 753 Blackburn, Scudder, Van Wassenhove /Speed/ 883 f. und Levendel /Reliability Analysis/ 151.

[29] Vgl. Nikula, Sajaniemi, Kälviäinen /Requirements Engineering/ 7 f. und Neill, Laplante /Requirements Engineering/ 42-44.

[30] Vgl. Al-Rawas, Easterbrook /Communication problems/ 47-60.

[31] Vgl. zum Problem des Technologietransfers im Fall der Anforderungsanalyse Nikula, Sajaniemi, Kälviäinen /Management View/ 81-83 und Nikula, Sajaniemi, Kälviäinen /Requirements Engineering/ 16 f.

fischen Rahmenbedingungen zu beseitigen. Hierzu muss er erst einmal die Ursachen bestimmen können.

Da die Ursachen von Anforderungsfehlern bestimmt werden müssen, um nachhaltig Anforderungsfehler zu vermeiden oder früh zu finden, sollte dies bereits *während* des Softwareentwicklungsvorhabens möglich sein. So können selbst für das laufende Entwicklungsvorhaben potenzielle Anforderungsfehler vermieden oder bereits entstandene früher entdeckt werden. Werden die Ursachen bestimmt, nachdem ein Entwicklungsvorhaben abgeschlossen wurde, kann man ausschließlich in zukünftigen Entwicklungsvorhaben Anforderungsfehler vermeiden und früh finden. Dies jedoch auch nur dann, wenn die Rahmenbedingungen gleich bleiben, da Ursachen häufig situativ sind.[32]

Werden Fehler erfasst und ausgewertet, können sie ein Ausgangspunkt für die entwicklungsbegleitende Bestimmung der Ursachen für Anforderungsfehler sein. Zum einen liegen Fehler bereits während der Entwicklung vor. Zum anderen stellen Anforderungs- und Folgefehler Symptome von Mängeln im Prozess der Anforderungsanalyse dar. Es gibt in anderen Anwendungsgebieten, wie dem Betrieb von Anlagen wie Atomkraftwerken, eine lange Tradition ausgehend von Wirkungen Ursachen zu analysieren.[33] Ebenso erlauben Art und Häufigkeit von Fehlern in der Softwareentwicklung Rückschlüsse auf den Prozess.[34] In vielen Softwareentwicklungsvorhaben werden jedoch Fehler korrigiert ohne deren Informationsgehalt zu nutzen. So werden gemäß einer 1997 durchgeführten empirischen Untersuchung in 80% der Unternehmen zwar Fehler dokumentiert, jedoch nur bei 42% in einer Fehlerdatenbank und nur bei 47% anhand eines standardisierten Formulars.[35] Zudem sind verbreitete, aktuell verfügbare kommerzielle Software und Open-Source-Software zur Fehlerverfolgung nicht darauf ausgerichtet, über Fehleranalysen auf den Prozess zu schließen.[36] Aus Fehlern systematisch

---

[32] Die Eignung eines Prozesses der Anforderungsanalyse ist in hohem Maße abhängig von den Rahmenbedingungen des Projektes. Insofern ist ein Streben nach allgemein gültigen Gestaltungsempfehlungen für die Anforderungsanalyse nicht zweckmäßig. Vgl. hierzu Chatzoglou /Factors/ 637, Morris, Masera, Wilikens /Requirements Engineering/ 137, Hickey, Davis /Elicitation technique/ 169 f. und Damian u. a. /Requirements Engineering/ 256. Eine weitere Schlussfolgerung daraus ist, dass auch Ursachen für Fehler abhängig von den Rahmenbedingungen sind. Vgl. hierzu Grady /Software metrics/ 139.

[33] Vgl. zur Verbreitung der Ursacheanalyse bzw. „root cause analysis“ in verschiedenen Branchen die Fallstudien in Latino, Latino /Root cause analysis/ 179-236.

[34] Vgl. hierzu zum Beispiel Bhandari u. a. /Case study/, Bhandari u. a. /Improvement/, Leszak, Perry, Stoll /Case study/ und Mays u. a. /Defect prevention/.

[35] Vgl. hierzu Müller, Wiegmann, Avci /Prüf- und Testprozesse/ 38. Die Untersuchung berücksichtigt 78 deutsche Softwareunternehmen. Sie wird in Müller /Prüf- und Testprozesse/ 26-135 im Detail dargestellt.

[36] Diese These stützt sich auf Erfahrungen des Autors mit entsprechenden Softwarewerkzeugen. Empirische Erkenntnisse liegen diesbezüglich nicht vor.

lernen zu können, diese Möglichkeit wird so versäumt.[37] Dies geschieht, obwohl Fehlerdaten zu der wichtigsten Informationsquelle zählen, wenn Entscheidungen zur Verbesserung von Softwareentwicklungsprozessen getroffen werden sollen.[38] Dies geschieht, obwohl in mehreren Fallstudien fehlerbasierte Verfahren zur Prozessverbesserung erfolgreich für die Entwicklung und Wartung von Software durchgeführt worden sind.[39]

Vorhandene fehlerbasierte Verfahren für die Softwareentwicklung sind häufig nicht ausreichend theoretisch begründet und bieten nur rudimentäre Unterstützung für die Identifikation und Analyse von Mängeln im Prozess der Anforderungsanalyse als Ursache für Anforderungsfehler.[40] Der Inhalt des Forschungsproblems der vorliegenden Arbeit spiegelt diesen unbefriedigenden Zustand wider und kann mit der folgenden These formuliert werden:

> *Es existiert kein begründetes Verfahren, um Fehler mit dem Ziel zu erfassen und auszuwerten, um entwicklungsbegleitend Mängel im Prozess der Anforderungsanalyse zu finden.*

Dieses Forschungsproblem ist relevant, weil seine Lösung wiederum einen Beitrag zur Lösung des Praxisproblems eröffnet. Werden mit einem entsprechenden Verfahren Ursachen für Anforderungsfehler bestimmt, kann sowohl der Prozess der Anforderungsanalyse als auch der der Qualitätssicherung gezielt verbessert werden.[41] Werden infolge dessen Anforderungsfehler früher entdeckt, sinkt auch die Anzahl der Folgefehler im Entwurf und in der Implementierung. Werden Anforderungsfehler sogar vermieden, können Folgefehler erst gar nicht entstehen. Insgesamt würde so die Anzahl der Fehler sinken, die durch Mängel im Prozess der Anforderungsanalyse verursacht werden.

1.2 Ziele der Arbeit

Viele Fehler werden durch Mängel im Prozess der Anforderungsanalyse verursacht; dieses Praxisproblem gilt sowohl für Anwendungssoftware als auch Systemsoftware oder systemnahe Software.[42] Die Art der entwickelten Software hat jedoch Einfluss auf den Prozess der Anforderungsanalyse und damit auch auf ein fehlerbasiertes Verfahren zum Auffinden von

---

[37] Vgl. Grady /Software Metrics/ 124, Koru, Tian /Defect handling/ 58 f. und Fredericks, Basili /Defect tracking/ 18 f.

[38] Vgl. Grady /Software failure analysis/ 155 f. und Müller /Prüf- und Testprozesse/ 57 f.

[39] Vgl. zum Beispiel Briand u.a. /Change analysis process/, Bridge, Miller /Orthogonal Defect Classification/, Bhandari u. a. /Improvement/ und Mays u. a. /Defect prevention/.

[40] Vgl. hierzu das Kapitel 3.4, insbesondere das Unterkapitel 3.4.5.

[41] Vgl. Grady /Software metrics/ 138.

[42] Vgl. die empirischen Studien in Kapitel 3.2. Die unterschiedlichen Arten von Software werden in Kapitel 2.2 definiert und abgegrenzt.

Mängeln in diesem Prozess. Um einen klaren Fokus zu erhalten, ist es sinnvoll, sich bei der Herleitung eines Verfahrens auf eine Softwareart zu beschränken. Im Rahmen dieser Arbeit erfolgt diese Einschränkung auf die Anwendungssoftware.

Auf der Grundlage des Forschungsproblems werden die Ziele dieser Arbeit abgeleitet. Wie bereits im letzten Kapitel beschrieben lautet das Forschungsproblem:

> *Es existiert kein begründetes Verfahren, um Fehler mit dem Ziel zu erfassen und auszuwerten, um entwicklungsbegleitend Mängel im Prozess der Anforderungsanalyse zu finden.*

Wird das Forschungsproblem als Frage umformuliert und auf die Entwicklung von Anwendungssoftware beschränkt, liefert diese das **Hauptziel** der vorliegenden Arbeit:

> *Wie können Mängel im Prozess der Anforderungsanalyse bereits während der Entwicklung einer Anwendungssoftware gefunden werden, indem Fehler erfasst und ausgewertet werden?*

Um die Untersuchung des Hauptziels zugänglicher zu machen, wird es in Teilziele zerlegt. Hierzu wird die Frage zum Hauptziel in mehrere Unterfragen aufgeteilt. Diese Unterfragen beziehen sich auf die Grundlagen der Arbeit, die Konzeption des Verfahrens und die Anwendung des Verfahrens. Ihre Beantwortung führt zugleich zu einer Antwort auf die Hauptfrage.

Die ersten beiden Unterfragen schaffen die Voraussetzung für einen konzeptionellen Bezugsrahmen:

1. *Was sind die konstituierenden Merkmale der Anforderungsanalyse und der Qualitätssicherung für die Entwicklung von Anwendungssoftware? (Grundlagen)*
2. *Was ist ein Prozess und was ein Fehler? (Grundlagen)*

Die nächsten drei Unterfragen geben einen Überblick über den Stand der Forschung und untermauern damit sowohl das Praxis- als auch das Forschungsproblem der vorliegenden Arbeit:

3. *Welche empirischen Erkenntnisse liegen über die Häufigkeit von Fehlern vor, die auf Mängel im Prozess der Anforderungsanalyse zurückzuführen sind? (Grundlagen)*
4. *Welcher Aufwand entsteht gemäß empirischen Untersuchungen, um Anforderungsfehler zu korrigieren? (Grundlagen)*
5. *Welche fehlerbasierten Verfahren existieren, um Mängel in Prozessen der Softwareentwicklung zu finden? (Grundlagen)*

Zwei Unterfragen richten sich auf die Entwicklung eines konzeptionellen Bezugsrahmens zur Erklärung relevanter Zusammenhänge zwischen Prozessen und Fehlern. Sie bereiten die Konzeption eines fehlerbasierten Verfahrens vor und lauten:

6. *Wie können Mängel im Prozess der Anforderungsanalyse beschrieben und erklärt werden? (Konzeption)*
7. *Was sind gemäß empirischen Untersuchungen bekannte Mängel im Prozess der Anforderungsanalyse? Welche Merkmale weisen Fehler auf, die durch diese Prozessmängel verursacht wurden? (Konzeption)*

Mit der Konzeption des fehlerbasierten Verfahrens im engeren Sinn beschäftigen sich folgende Unterfragen:

8. *Wie sollte ein praktisch einsetzbares Verfahren aussehen, mit dem Fehler erfasst und ausgewertet werden, um bereits während der Entwicklung einer Anwendungssoftware Mängel im Prozess der Anforderungsanalyse zu finden? (Konzeption)*
9. *Welche Nutzenpotenziale bietet die Anwendung eines solchen Verfahrens? (Konzeption)*

Eine abschließende Teilfrage bezieht sich auf die Anwendung des Verfahrens:

10. *Welche Erfahrungen werden gemacht, wenn das entwickelte Verfahren praktisch eingesetzt wird? (Anwendung)*

1.3 Einordnung der Arbeit und Forschungsmethodik

1.3.1 Vorüberlegungen

Das Hauptziel dieser Arbeit ist es, ein Verfahren systematisch herzuleiten, das Mängel im Prozess der Anforderungsanalyse bereits während der Entwicklung einer Anwendungssoftware findet, indem Fehler erfasst und ausgewertet werden. Dieser Anspruch wirft die Frage auf, wie ein entsprechendes Verfahren *begründet* werden kann. Das Problem der Begründung ist für jede Wissenschaft und damit auch für jede wissenschaftliche Arbeit von zentraler Bedeutung.[43] Sie beinhaltet das Streben zum einen nach Wahrheit über die Beschaffenheit bestimmter Ausschnitte der Wirklichkeit, zum anderen nach Sicherheit darüber, ob das, was herausgefunden wurde, auch wirklich wahr ist.

[43] Vgl. hierzu und zum folgenden Satz Albert /Traktat/ 9 f.

Insbesondere der wissenschaftstheoretische Standpunkt prägt die Herangehensweise an das Begründungsproblem.[44] Die wissenschaftstheoretische Positionierung ist für eine wissenschaftliche Arbeit unverzichtbar und sollte vor allem aus zwei Gründen offen gelegt werden. Erstens hat sie maßgeblichen Einfluss auf die Auswahl der Methoden zur Gewinnung und Überprüfung von wissenschaftlicher Erkenntnis (kurz Forschungsmethoden[45]). Zweitens kann der Grad der Zielerreichung einer wissenschaftlichen Arbeit nur dann vollständig beurteilt werden, wenn der wissenschaftstheoretische Standpunkt explizit dargelegt wird.[46]

Da Forschungsmethoden auch durch den Gegenstandsbereich einer Wissenschaft bestimmt werden,[47] wird in Kapitel 1.3.2 zunächst erörtert, wie der Beitrag dieser Arbeit in die Wirtschaftsinformatik eingeordnet werden kann. Daran schließt in Kapitel 1.3.3 die wissenschaftstheoretische Positionierung der Arbeit an. Darauf aufbauend wird in Kapitel 1.3.4 erläutert, welche Forschungsmethoden in dieser Arbeit angewandt werden.

1.3.2 Einordnung in die Wirtschaftsinformatik

Die vorliegende Arbeit versteht sich als ein Beitrag zur Wirtschaftsinformatik. Die Wirtschaftsinformatik ist eine Wissenschaft und untersucht als solche Informations- und Kommunikationssysteme in Wirtschaft und Verwaltung, kurz *Informationssysteme*.[48] Zum einen versucht sie Struktur und Verhalten von Informationssystemen zu *erklären*. Zum anderen entwickelt sie Verfahren, Methoden und Werkzeuge, um Informationssysteme zu *gestalten*. Informationssysteme umfassen sowohl menschliche als auch maschinelle Komponenten, weshalb sie auch als soziotechnische Systeme bezeichnet werden. Sie sollen nach wirtschaftlichen Kriterien für betriebliche Aufgaben relevante Informationen bereitstellen und die Kommunikation unterstützen.

Diese Arbeit ist im Kontext der Gestaltungsaufgabe der Wirtschaftsinformatik einzuordnen. Gegenstand der Arbeit ist die Verbesserung der Entwicklung einer Anwendungssoftware als Bestandteil eines Informationssystems. Hierzu wird ein Verfahren begründet hergeleitet. Die-

---

[44] Vgl. zu diesem Absatz Albert /Wissenschaftstheorie/ 4675-4677.

[45] Vgl. Heinrich /Wirtschaftsinformatik/ 97.

[46] Vgl. Albert /Wissenschaftstheorie/ 4675-4677.

[47] Vgl. Heinrich /Wirtschaftsinformatik/ 97.

[48] Vgl. zu diesem Absatz Heinrich /Wirtschaftsinformatik/ 3 f., 14-20 und WKWI /Wirtschaftsinformatik/ 80. Neben der Erklärungs- und Gestaltungsaufgabe hat die Wirtschaftsinformatik eine Beschreibungs- sowie Prognoseaufgabe. Ziel der Beschreibungsaufgabe ist die eindeutige Beschreibung des Untersuchungsgegenstands. Sie liefert erst die Voraussetzungen dafür, dass Aussagen zur Erklärung, Prognose und Gestaltung gemacht werden können. Im Rahmen der Prognoseaufgabe wird angestrebt, durch Modelle Vorhersagen über das Verhalten von Informationssystemen zu machen. Siehe zur Beschreibungs- und Prognoseaufgabe der Wirtschaftsinformatik die eingangs aufgeführten Quellenverweise.

se Arbeit hat einen methodischen Auftrag in dem Sinne, dass sie aufgrund ihrer zugrunde liegenden Ziele sowohl zur *Entwicklung* als auch zu dem *Verständnis* von Methoden und Techniken zur Beschreibung, Entwicklung, Einführung und Nutzung von Informationssystemen beiträgt.[49] Zugleich impliziert dies, dass sie neben einem Gestaltungsziel auch ein Erkenntnisziel verfolgt. Die Zielfragen zur Konzeption des Verfahrens entsprechen Gestaltungszielen (Unterfragen 8 und 9 aus Kapitel 1.2). In Abgrenzung hierzu bereiten bestimmte Zielfragen die Ergebnisse erkenntniszielgeleiteter Forschung zur Begründung des Verfahrens vor. Sie entsprechen folglich Erkenntniszielen (Unterfragen 1 bis 7 und 10 aus Kapitel 1.2).
Die Wirtschaftsinformatik versteht sich aufgrund ihres Untersuchungsgegenstandes als ein interdisziplinäres Fach.[50] Sie greift insbesondere auf Erkenntnisse der Betriebswirtschaftslehre, der Informatik und der Organisationstheorie zurück und ergänzt diese Erkenntnisse um weitere für die Wirtschaftsinformatik spezifische Forschungsergebnisse.
In dieser Arbeit wird verstärkt auf Erkenntnisse der Organisationstheorie sowohl im Hinblick auf ihre Forschungsmethodik als auch auf ihren Untersuchungsgegenstand Bezug genommen. Für die Wirtschaftsinformatik lassen sich theoretische Konzepte vor allem gewinnen, indem organisationstheoretische Erkenntnisse stärker eingebunden werden.[51]

1.3.3 Wissenschaftstheoretische Einordnung: Neo-Positivismus

Häufig fehlt in wissenschaftlichen Beiträgen zur Wirtschaftsinformatik eine explizite Diskussion der zu Grunde liegenden wissenschaftstheoretischen Annahmen und damit der Forschungsmethodik.[52] Forschung sollte jedoch unter weit reichender Offenlegung der wissenschaftlichen Grundannahmen und Vorgehensweisen erfolgen.[53] Erst diese Transparenz ermöglicht es den Lesern, die Forschungsergebnisse angemessen zu verstehen und insbesondere einer Bewertung zu unterziehen. Des Weiteren ist die Wirtschaftsinformatik mit unterschiedlichen wissenschaftstheoretischen Paradigmen konfrontiert, bei denen unklar ist, ob eine gemeinsame Nutzung sinnvoll, geschweige denn möglich ist.[54]

---

[49] Vgl. zum Begriff des methodischen Auftrags sowie der Unterscheidung von Erkenntnis- und Gestaltungszielen in der Wirtschaftsinformatik Becker u. a. /Epistemologische Positionierung/ 346 f.

[50] Vgl. zu diesem Absatz Heinrich /Wirtschaftsinformatik/ 109-120.

[51] Vgl. Rolf /Wirtschaftsinformatik/ 261.

[52] Heinrich untersucht 538 Beiträge aus der Zeitschrift Wirtschaftsinformatik und kommt u. a. zum Ergebnis, dass nur in 11% der Fälle die Autoren ihre Forschungsmethodik transparent machen. Vgl. Heinrich /Forschungsmethodik/ 112.

[53] Vgl. hierzu und zum nachfolgenden Satz Becker u. a. /Epistemologische Positionierung/ 335-337, 349-351.

[54] Vgl. Mingers /Pluralist Methodology/ 247-257.

Für die wissenschaftstheoretische Positionierung einer Arbeit sind insbesondere zwei Fragen zu beantworten, die das Realitätsverständnis charakterisieren:[55]

- Ontologische Frage: Wie ist die Realität beschaffen?
- Erkenntnistheoretische (epistemologische) Frage: Was kann ein Subjekt von dem, was in der Realität existiert, so erkennen, wie es existiert?

Die festgelegten ontologischen und erkenntnistheoretischen Annahmen haben Einfluss auf die Einschätzung, welche Erkenntnismethoden eingesetzt werden sollen, und damit auf die methodologische Frage, wie man zu neuer Erkenntnis gelangt.[56]

Der Positivismus ist das dominierende wissenschaftstheoretische Paradigma[57] in der Wirtschaftsinformatik,[58] was nicht bedeutet, dass er frei von Kritik ist.[59] Gerade die Kritik am Positivismus hat sowohl zu alternativen wissenschaftstheoretischen Paradigmen als auch zu Weiterentwicklungen geführt.[60] Es bleibt jedoch festzuhalten, dass ein wissenschaftstheoretisches Paradigma eine Menge von grundlegenden, philosophischen Annahmen darstellt, die sich einem Beweis entziehen.[61] Folglich ist ein Paradigma weder richtig noch falsch.[62] Vielmehr muss begründet werden, inwiefern ein gewähltes Paradigma für eine Forschungsarbeit zweckmäßig ist.

Die vorliegende Arbeit berücksichtigt die Kritik am Realitätsverständnis des klassischen Positivismus. Sie geht deshalb von *neopositivistischen Annahmen* zum Realitätsverständnis aus.[63] Hierbei wird die ontologische Frage durch die Annahme eines ontologischen Realismus wie

---

55 Vgl. Guba, Lincoln /Paradigms/ 107 f. und Schütte /Basispositionen/ 214 f.

56 Vgl. Schütte /Basispositionen/ 215 f.

57 Vgl. zum Begriff eines Paradigmas Guba, Lincoln /Paradigms/ 107 f.

58 Vgl. Orlikowski, Baroudi /Research approaches/ 6 und Dubé, Paré /Case Research/ 601-604. Streng genommen gibt es nicht *den* Positivismus, sondern verschiedene Formen des Positivismus. Juhos /Positivismus/ 27-62 gibt einen Überblick über Formen des Positivismus. Lincoln, Guba /Inquiry/ 28 nennen grundlegende Annahmen, die einen gemeinsamen Nenner verschiedener Formen darstellen.

59 Vgl. hierzu z. B. die Kritik in Lincoln, Guba /Inquiry/ 24-33.

60 Vgl. Guba, Lincoln /Paradigms/ 106-116.

61 Lincoln und Guba sprechen deshalb in diesem Zusammenhang nicht von Annahmen, sondern von Axiomen, die sie als nicht beweisbare Grundüberzeugungen definieren. Vgl. Lincoln, Guba /Inquiry/ 33. Schütte stellt hierzu in Schütte /Basispositionen/ 224 fest: „Die Frage, ob eine Realität objektiv existiert und diese objektiv erkannt werden kann, lässt sich ‚objektiv' nicht beantworten. Jede Entscheidung, wie wir erkennen, ist zwangsläufig selbstreflexiv. Es handelt sich um eine metaphysische Annahme, die ein Forscher seiner Arbeit zugrunde legt, ohne dass er diese Annahme beweisen könnte".

62 Vgl. hierzu und zum nachfolgenden Satz Guba, Lincoln /Paradigms/ 108.

63 Diese neopositivistischen Annahmen werden in der Literatur mitunter unter den Bezeichnungen Neopositivismus, kritischer Realismus, Postpositivismus oder einfach nur Positivismus aufgeführt. Vgl. Krauss /Research paradigms/ 761. Es sei darauf hingewiesen, dass diese Bezeichnungen in der Literatur z. T. mit sehr unterschiedlichen Inhalten verknüpft werden.

im klassischen Positivismus beantwortet. Der erkenntnistheoretischen Frage wird abweichend vom klassischen Positivismus durch einen kritischen Realismus begegnet. Die Annahmen lauten im Einzelnen:

> *Annahme 1 (ontologischer Realismus): Es gibt eine einzige Realität, die unabhängig vom menschlichen Denken und Sprechen ist.*[64]
>
> *Annahme 2 (kritischer Realismus): Die Wahrnehmung der Realität wird durch subjektabhängige Verzerrungen beeinträchtigt. Aussagen über die Realität haben deshalb hypothetischen Charakter und müssen stets kritisch hinterfragt werden.*[65]

Die Annahme des ontologischen Realismus wird mehrheitlich vertreten.[66] Sie ist auch für diese Arbeit zweckmäßig, da Fehler und ihre Prozessursachen real gegeben sein müssen; d. h. insbesondere, dass sie unabhängig von einem Subjekt vorliegen müssen. Dies ist für ein Verfahren zwingend erforderlich, anhand dessen Fehler quantitativ erfasst und ausgewertet werden sollen. Eine objektive Realität ist zudem auch als Regulativ erforderlich,[67] um zu überprüfen, ob die Änderung eines Prozesses die gewünschte Wirkung aufzeigt.

Aus erkenntnistheoretischer Sicht liegt dem klassischen Positivismus ein naiver Realismus zugrunde.[68] Der Ansatz des naiven Realismus geht davon aus, dass die Realität so ist, wie sie wahrgenommen wird. Gegen den naiven Realismus gibt es starke Vorbehalte.[69] In dieser Arbeit wird deshalb abweichend vom klassischen Positivismus statt eines naiven Realismus ein kritischer Realismus angenommen. Letzterer geht von der Subjektivität der Sinneswahrnehmung aus und legt eine Grundhaltung nahe, Aussagen bezüglich der Realität kritisch zu beleuchten und von ihrer subjektiv bedingten Beschränkung nach Möglichkeit zu befreien, um so einer objektiven Wahrnehmung näher zu kommen.[70] Deshalb beschreibt Albert den kri-

---

[64] Vgl. Lincoln, Guba /Inquiry/ 37 f., Becker u. a. /Epistemologische Positionierung/ 338 und Schütte /Basispositionen/ 220.

[65] Vgl. Krauss /Research paradigms/ 761 f. und Guba, Lincoln /Paradigms/ 110.

[66] Vgl. Schütte /Basispositionen/ 220 und die dort aufgeführten Literaturquellen.

[67] Vgl. Schütte /Basispositionen/ 224.

[68] Vgl. Lincoln, Guba /Inquiry/ 28.

[69] Vgl. Haug /Wissenschaftstheoretische Problembereiche/ 95 und Schütte /Basispositionen/ 220.

[70] Vgl. Schütte /Basispositionen/ 221 f.
Zahlreiche Arbeiten, die sich selbst dem positivistischen Paradigma zuordnen oder gemeinhin dem positivistischen Paradigma zugeordnet werden, gehen implizit von den hier explizit formulierten Annahmen zum Realitätsverständnis aus. Dazu zählen insbesondere auch Arbeiten, die neben einer quantitativen auch eine qualitative Datenerhebung und –auswertung für zweckmäßig erachten (wie z. B. Yin /Case study research/ 14 f.). Der naive Realismus des klassischen Positvismus ist mit einer qualitativen Datenerhebung und –auswertung nicht vereinbar.

tischen Realismus als „die These der Existenz einer objektiven Realität, die approximativ erkennbar ist“[71].

Auf dem ontologischen und erkenntnistheoretischen Standpunkt aufbauend sind zwei weitere Annahmen des klassischen Positivismus für die vorliegende Arbeit bedeutsam:[72]

> *Annahme 3 (Existenz nomologischer Aussagen): Es gibt generelle Aussagen, die für eine größere Zahl von Fällen in der Realität Gültigkeit besitzen. Solche Gesetzmäßigkeiten[73] (synonym: nomologische Aussagen) haben oder können in eine Wenn-Dann-Form überführt werden, die besagt unter welchen Bedingungen (Wenn-Teil) regelmäßig bestimmte Ereignisse (Dann-Teil) eintreten.*
>
> *Annahme 4 (lineare Kausalität): Jeder Zustand oder jedes Ereignis kann erklärt werden als das Ergebnis (Wirkung) eines oder mehrerer vorhergehender Ereignisse (Ursachen).*

Nomologische Aussagen erklären zum einen, *warum* bestimmte Ereignisse oder Zustände auftreten.[74] Genauso geben sie Hinweise, *wie* durch eine zielgerichtete Gestaltung angestrebte Zustände erreicht werden können. In dieser Arbeit wird ein Verfahren hergeleitet, das darauf abzielt, Gestaltungsempfehlungen für eine bestimmte Klasse von praktischen Problemen zu geben. Diese Gestaltungsempfehlungen abstrahieren von einem einzelnen, individuellen Problemfall. Folglich müssen sie sich auf nomologische Aussagen stützen. Wird die Existenz nomologischer Aussagen verneint, können demnach keine praktischen Gestaltungsempfehlungen formuliert werden.

Eine nomologische Aussage impliziert bereits eine lineare (einseitige) Kausalität, da sie eine gerichtete Beziehung zwischen einer Maßnahme unter bestimmten Bedingungen und ihrer Wirkung beschreibt. Für das hier hergeleitete Verfahren ist die Annahme bezüglich der linearen Kausalität unabdingbar. Es baut gerade auf einem Ursache-Wirkungs-Zusammenhang auf, indem es über die Analyse von Fehlern Rückschlüsse über die Ursachen im Prozess schließen möchte.

---

[71] Vgl. Albert /Wissenschaftstheorie/ 4676.

[72] Vgl. zum Folgenden Lincoln, Guba /Inquiry/ 37 f.

[73] Vgl. Kubicek /Organisationsforschung/ 24 f.

[74] Vgl. hierzu und zum nächsten Satz Kubicek /Organisationsforschung/ 24-27, 40 und Gadenne /Rationalismus/ 13.

1.3.4 Forschungsmethodik der Arbeit

Mit Forschungsmethodik bezeichnet Heinrich die wissenschaftlich begründete, systematische Verwendung von Forschungsmethoden.[75] Letztere sind Methoden zur Gewinnung und Überprüfung wissenschaftlicher Erkenntnis. Gegenstand dieses Kapitels ist die Forschungsmethodik der vorliegenden Arbeit. Zunächst wird in Kapitel 1.3.4.1 der Theoriebegriff sowie das Konzept gedanklicher Bezugsrahmen erörtert, um die Begründung zugänglicher zu machen, warum welche Forschungsmethoden ausgewählt werden. Anschließend wird in Kapitel 1.3.4.2 die Auswahl der Forschungsmethoden beschrieben.

1.3.4.1 Vorstufen einer Theorie: gedankliche Bezugsrahmen

Diese Arbeit ist im Kontext der Gestaltungsaufgabe der Wirtschaftsinformatik einzuordnen.[76] Sie erhebt den Anspruch, ein begründetes Verfahren systematisch herzuleiten, das über die Analyse von Fehlern in der Softwareentwicklung Mängel im Prozess aufdeckt. Diese praktische Orientierung impliziert jedoch in keiner Weise einen Verzicht auf eine theoretische Fundierung.[77] Vielmehr ist eine Theorie zwingend erforderlich, um zweckmäßige, praxisorientierte Aussagen aufzustellen.[78]

Was aber ist eine Theorie?[79] Eine *Theorie im engeren Sinn (i. e. S.)* oder synonym *erklärende Theorie* ist ein System von Gesetzeshypothesen, also nomologischen Aussagen, die Ausschnitte der Realität erklären.[80] Diesen engen Theoriebegriff erweitert Grochla erstens um solche Aussagen, die zum einen die Vorstufe von erklärenden, nomologischen Aussagen bilden, und zweitens um praxeologische Aussagen, aus denen praktische Handlungsanweisungen zur Lösung realer Probleme abgeleitet werden können.[81] In diesem Sinne besteht eine *Theorie im weiteren Sinn (i. w. S.)* aus begrifflichen, beschreibenden als auch erklärenden Aussagen über reale Tatbestände sowie schließlich praxeologische Aussagen. Sofern im Folgenden die Begriffe „Theorie“ oder „theoretisch“ verwendet werden, beziehen sie sich auf den Theoriebegriff im weiteren Sinn, sofern nicht anderes angegeben ist.

---

[75] Vgl. hierzu und zum nächsten Satz Heinrich /Wirtschaftsinformatik/ 95-97.

[76] Vgl. hierzu die Ausführungen in Kapitel 1.3.2 .

[77] Vgl. Grochla /Organisationstheorie/ 54.

[78] Dies äußert sich in dieser Arbeit dadurch, dass neben dem Gestaltungsziel ein Erkenntnisziel verfolgt wird. Vgl. hierzu die Ausführungen in Kapitel 1.3.2.

[79] Dass diese Frage nicht trivial ist, zeigen die Ausführungen in Sutton, Staw /Theory/ 371-377.

[80] Vgl. Sutton, Staw /Theory/ 378 und Gadenne /Rationalismus/ 5.

[81] Vgl. zu diesem und dem nachfolgenden Satz Grochla /Organisationstheorie/ 54 f. Es sei daraufhin gewiesen, dass Grochla die Begriff „praxeologisch“ und „instrumental“ synonym verwendet.

Ausführungen zur Forschungsmethodik im positivistischen Kontext legen im Allgemeinen einen Schwerpunkt auf die Begründung von erklärenden Theorien.[82] So wurden im Rahmen der Diskussion um Methoden zur Prüfung von aus erklärenden Theorien abgeleiteter Hypothesen eine Vielzahl von Qualitätskriterien für empirische, quantitative Untersuchungen erarbeitet.[83] In der Wirtschaftsinformatik liegen jedoch noch kaum erklärende Theorien vor; dies gilt im Besonderen für den in dieser Arbeit relevanten Zusammenhang zwischen Fehlern in der Softwareentwicklung und Mängel im Softwareentwicklungsprozess.[84] Ein zu starker Fokus auf den Begründungszusammenhang vernachlässigt zudem den Entdeckungszusammenhang, also die Theoriebildung. Ein vergleichbarer Umstand hat in der Organisationstheorie als auch in den Sozialwissenschaften zu einer Fülle von empirischen Einzelergebnissen geführt, die erstens nicht vergleichbar und zweitens nicht zu Theorien integrierbar sind.[85]

Als ein Ausweg aus diesem Problem werden in der Organisationstheorie gedankliche Bezugsrahmen vorgeschlagen. Nach Grochla handelt es sich hierbei um „Ordnungsschemata für erkenntnisbezogene und handlungsbezogene Vorstellungen über die Realität“[86]. Gedankliche Bezugsrahmen können als die Vorstufe von erklärenden Theorien angesehen werden. Sie führen nämlich theoretische Begriffe ein und beinhalten bereits allgemeine Gesetzeshypothesen, die meist jedoch nur Zusammenhänge andeuten.[87] Sowohl die theoretischen Begriffe als auch die Gesetzeshypothesen können zukünftig Bestandteil einer erklärenden Theorie werden.

Das Konzept der gedanklichen Bezugsrahmen wird aufgrund des Stands der Forschung des untersuchten Themas auch für diese Arbeit übernommen. Zwei weitere Vorteile sprechen für ihre Anwendung. Obwohl das Stadium von erklärenden Theorien noch nicht erreicht ist, können gedankliche Bezugsrahmen *bereits praktische Gestaltungshilfe leisten*, indem sie „Interpretationsmuster“ zur Verfügung stellen, die ein besseres Verständnis relevanter Ausschnitte der Wirklichkeit ermöglichen.[88] Gedankliche Bezugsrahmen unterstützen zudem die *Vorbereitung explorativer Studien*, indem sie z. B. aufzeigen, welche Daten überhaupt empirisch erhoben werden sollen.[89]

---

[82] Vgl. Orlikowski, Baroudi /Research approaches/ 5

[83] Für einen umfassenden Überblick über entsprechende Kriterien siehe Straub, Boudreau, Gefen /Validation guidelines/ 380-418.

[84] Vgl. zu den Theoriedefiziten in der Wirtschaftsinformatik Lehner /Theoriebildung/ 16 f. Insbesondere fehlen Theorien, die erklären wie Fehler in Arbeitsergebnisse der Softwareentwicklung eingeführt werden. Vgl. hierzu Kitchenham u. a. /Guidelines/ 724.

[85] Vgl. Kubicek /Organisationsforschung/ 38 und Gadenne /Rationalismus/ 6.

[86] Vgl. Grochla /Organisationstheorie/ 65.

[87] Vgl. Kubicek /Organisationsforschung/ 37 f.

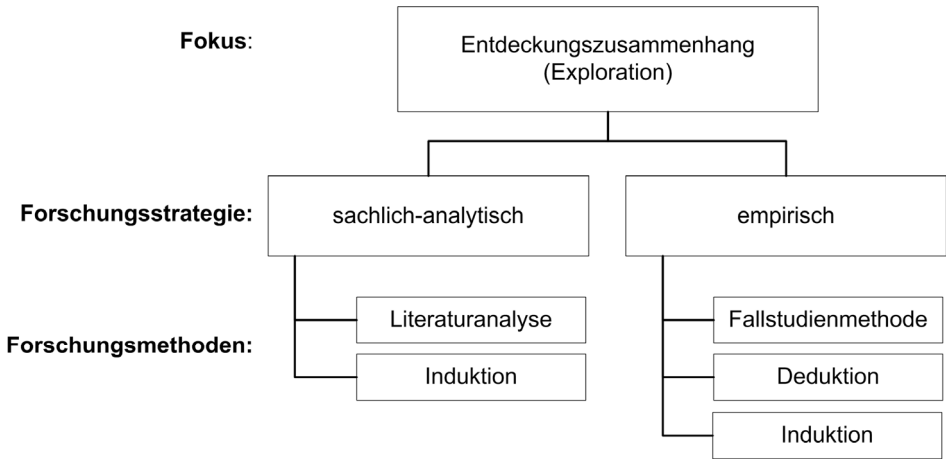
[88] Vgl. Kubicek /Organisationsforschung/ 39, 45.

[89] Vgl. Staehle /Empirische Analyse/ 108-112 und Kubicek /Organisationsforschung/ 39.

Nach Kubicek kann der Forschungsprozess so umschrieben werden, dass fortlaufend gedankliche Bezugsrahmen entwickelt und aufgrund explorativer Studien ständig präzisiert und angepasst werden.[90] Diese Vorstellung vom Forschungsprozess wird auch in dieser Arbeit geteilt. Hierdurch wird zudem die Trennung zwischen Entdeckungs- und Begründungszusammenhang zugunsten eines „Wechselspiel[s] zwischen der Entwicklung von Konzepten und ihrer empirischen Überprüfung als dynamischer Lernprozess“[91] aufgehoben.

1.3.4.2 Forschungsmethoden der Arbeit

In dieser Arbeit wird ein begründetes Verfahren systematisch hergeleitet, das Mängel im Prozess der Softwareentwicklung aufdeckt, indem es Fehler analysiert. Der Stand der Forschung zu diesem Thema erfordert, dass der Entdeckungszusammenhang im Vordergrund steht. Deshalb wird in dieser Arbeit das erstmalige Erkennen von Zusammenhängen (*Exploration*[92]) betont.[93] Hierzu werden sowohl die sachlich-analytische als auch die empirische Forschungsstrategie verfolgt und bestimmte Forschungsmethoden eingesetzt. Die Auswahl der Forschungsstrategien und Forschungsmethoden werden nachfolgend begründet. Im Anschluss daran wird auf den Entdeckungs- und Verwertungszusammenhang wissenschaftlicher Aussagen in dieser Arbeit eingegangen.



Die Abbildung 1-2 gibt einen Überblick über die Forschungsmethodik dieser Arbeit.

[90] Vgl. Kubicek /Organisationsforschung/ 46.

[91] Kubicek /Organisationsforschung/ 46.

[92] Die Begriffe Entdeckungszusammenhang und Exploration werden in dieser Arbeit synonym verwendet.

[93] Vgl. Müller-Böling /Organisationsforschung/ 1494.

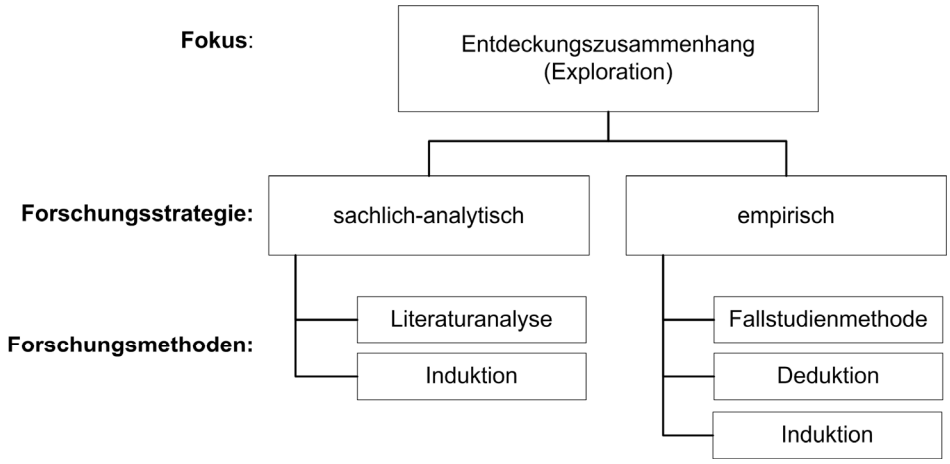


Abbildung 1-2: Forschungsmethodik der Arbeit

Sachlich-analytische Forschungsstrategie und Forschungsmethoden

Grochla beschreibt die *sachlich-analytische Forschungsstrategie* als „eine Art gedankliche Simulation der Realität mit dem Erkenntnisziel, die Beziehungen transparent zu machen und hieraus direkt Handlungsempfehlungen abzuleiten“[94]. Entwickelte Aussagen werden (zunächst) nicht empirisch überprüft, sondern durch Plausibilitätsüberlegungen und eventuelle empirisch festgestellte Teilzusammenhänge begründet.[95] Sie eignet sich für die Exploration, da sie vor allem die Suche nach neuen relevanten Größen und nach neuen Aussagen über mögliche Beziehungen zwischen verschiedenen Größen anstrebt.

Die sachlich-analytische Forschungsstrategie wird in dieser Arbeit eingesetzt, um Bezugsrahmen zu erarbeiten. Um zu gewährleisten, dass diese Forschungsergebnisse intersubjektiv überprüfbar sind, wird als Forschungsmethode das „*literature review*“ im Sinne von Webster und Watson angewandt.[96] Hierbei handelt es sich um eine systematische Analyse eines definierten Ausschnitts der Literatur mit Ziel der Synthese relevanter Aussagen. Deshalb wird diese Forschungsmethode im Folgenden kurz als *systematische Literaturanalyse* bezeichnet. Damit durch die systematischen Literaturanalysen möglichst *bewährte*[97] *Aussagen* in die gedanklichen Bezugsrahmen einfließen, wird die Forschungsmethode jeweils ausschließlich auf Literaturquellen angewandt, die empirische Untersuchungen zum Gegenstand haben.[98] Die berücksichtigten empirischen Literaturquellen werden zudem systematisch beurteilt.

---

[94] Grochla /Organisationstheorie/ 72.

[95] Vgl. hierzu und zum nächsten Satz Grochla /Organisationstheorie/ 72 f.

[96] Vgl. hierzu und zum nächsten Satz Webster, Watson /Literature review/ xiii-xxiii.

[97] Vgl. zum Begriff der Bewährung Gadenne /Bewährung/ 125-143.

[98] Entsprechende Quellen werden im Folgenden der Einfachheit halber kurz als empirische Literaturquellen bezeichnet.

Innerhalb der sachlich-analytischen Forschungsstrategie wird in dieser Arbeit als eine weitere Forschungsmethode die *Induktion* eingesetzt. Die Induktion ist ein heuristisches Schlussverfahren, bei dem aufgrund bestimmter Einzelerfahrungen von speziellen Aussagen auf eine allgemeine Aussage geschlossen wird.[99] Hierbei übersteigt der Gehalt der allgemeinen Aussagen den Umfang der Annahmen, welche zu den speziellen Aussagen getroffen wurden. Deshalb zeichnen sich durch die Induktion gewonnenen Aussagen durch einen hohen sachlichen Neuigkeitsgehalt, jedoch zugleich durch einen hohen Grad an Unsicherheit über ihren Wahrheitsgrad aus.

Die Induktion wird hier eingesetzt, um aus empirischen Befunden auf theoretische Aussagen für gedankliche Bezugsrahmen zu schließen (Entdeckungszusammenhang). Aufgrund ihrer Merkmale ist die Induktion nicht dafür geeignet, aufgrund von Beobachtungsdaten auf den Bestätigungsgrad von Theorien oder Bezugsrahmen zu folgern (Begründungszusammenhang).[100]

Empirische Forschungsstrategie und Forschungsmethoden

Die *empirische Forschungsstrategie* bzw. kurz *Empirie* ist eine methodengestützte Vorgehensweise, bei der gedankliche Konzepte aus einer Beobachtung der Realität gewonnen und die in diesen Konzepten enthaltenen Aussagen an der Realität überprüft werden.[101] Sie wird in dieser Arbeit wie die sachlich-analytische Forschungsstrategie eingesetzt, um das erstmalige Erkennen von Zusammenhängen zu ermöglichen.[102] Im Gegensatz zur sachlich-analytischen Forschungsstrategie erfolgt die Exploration in der empirischen Forschungsstrategie jedoch durch die Konfrontation mit der Realität.

Als Forschungsmethode innerhalb der empirischen Forschungsstrategie wird die Fallstudienmethode gewählt. Die *Fallstudienmethode* untersucht ein aktuelles Phänomen innerhalb seines realen Kontextes.[103] Sie eignet sich insbesondere dann, wenn die Grenzen zwischen dem Phänomen und seinem Kontext nicht direkt einsichtig sind.

Die Fallstudienmethode kann mit unterschiedlichen Zielsetzungen angewandt werden, so dass verschiedene Anwendungen der Fallstudienmethode unterschieden werden können.[104] In die-

---

[99] Vgl. zu diesem Absatz Heinrich /Wirtschaftsinformatik/ 98.

[100] Vgl. Gadenne /Rationalismus/ 10. Für eine umfassende Kritik der Induktion siehe Schurz /Induktion/ 25-40.

[101] In Anlehnung an Kubicek /Organisationsforschung/ 34 und Müller-Böling /Organisationsforschung/ 1494.

[102] Die Induktion und das nachher beschriebene Gegenstück, die Deduktion, sind grundlegende Denkmethoden, die integraler Bestandteil einer wissenschaftlichen Argumentation sind. Folglich sind sie auch regelmäßig in dieser Arbeit wieder zu finden. Im Kontext der Darstellung der Forschungsmethoden beschränkt sich die Beschreibung der Anwendung dieser Denkmethoden auf inhaltlich zentrale Aspekte.

[103] Vgl. hierzu und zum folgenden Satz Yin /Case study research/ 13 f. und Yin /Case study crisis/ 58.

[104] Vgl. Yin /Case study research/ 3-7.

ser Arbeit wird die Fallstudienmethode mit einer explorativen Zielsetzung eingesetzt. Der Umstand, dass eine Untersuchungseinheit eingehend in ihrem realen Kontext analysiert wird, erfordert einen längeren Aufenthalt des Forschers am Ort der Untersuchung. Dies ermöglicht aber zugleich das Erhebungsprogramm kurzfristig bei Bedarf zu ändern. Hieraus ergibt sich das explorative Potenzial der Fallstudienmethode.[105]
Ziel der explorativen Fallstudie ist die *Durchführbarkeit des hergeleiteten Verfahrens* zu untersuchen. Die Untersuchungseinheit ist die Anwendung des Verfahrens in einem realen Softwareentwicklungsvorhaben. Es werden reale Fehler dieser Softwareentwicklung erfasst und analysiert, um Mängel im Prozess der Anforderungsanalyse dieses Entwicklungsvorhabens aufzudecken. Drei Fragen stehen hierbei im engeren Fokus: Welche Faktoren begünstigen oder erschweren die Einführung und Anwendung des Verfahrens? Welche Faktoren erzeugen bei der Einführung und Anwendung des Verfahrens Aufwand? Welche Qualität haben die Ergebnisse des Verfahrens?

Allgemeine Empfehlungen zur Fallstudienmethode im positivistischen Kontext finden sich insbesondere in den Arbeiten von Yin[106], Dubé und Paré[107] sowie Benbasat, Goldstein und Mead[108]. Auf die explorative Anwendung der Fallstudienmethode im Besonderen, ebenfalls im positivistischen Kontext, geht eine Arbeit von Paré[109] ein. Die vorliegende Arbeit orientiert sich an diesen Empfehlungen. In Kapitel 7.3 wird beschrieben, welche Methoden im Einzelnen im Rahmen der Fallstudienmethode eingesetzt werden, um Daten zu erfassen und auszuwerten.
Innerhalb der empirischen Forschungsstrategie wird in dieser Arbeit als eine weitere Forschungsmethode die *Deduktion* eingesetzt. Die Deduktion ist ein formal-logisches Schlussverfahren, das den Wahrheitsgehalt von allgemeinen Aussagen auf spezielle Aussagen überträgt.[110] Sie ist das Gegenstück zur Induktion. Im Unterschied zur Induktion weisen durch die Deduktion gewonnene Aussagen zwar einen geringeren Neuigkeitsgehalt, aber auch einen geringeren Grad an Unsicherheit über ihren Wahrheitsgehalt aus.

---

[105] Vgl. Kubicek /Organisationsforschung / 59 f.
[106] Vgl. Yin /Case study research/.
[107] Vgl. Dubé, Paré /Case research/.
[108] Vgl. Benbasat, Goldstein, Mead /Case research strategy/.
[109] Vgl. Paré /Case study research/
[110] Vgl. zu diesem Absatz Heinrich /Wirtschaftsinformatik/ 99.

Die Deduktion wird hier eingesetzt, um Beobachtungen bei der exemplarischen Anwendung des Verfahrens in der Fallstudie mit nomologischen Aussagen aus den entwickelten Bezugsrahmen zu erklären.
Neben der Deduktion kommt auch die Induktion in der empirischen Forschungsstrategie zum Einsatz. Die in der explorativen Fallstudie gemachten Beobachtungen werden genutzt, um das Verfahren zu verbessern. Folglich werden von Einzelbeobachtungen auf allgemeine Aussagen geschlossen.

Entdeckungs- und Verwertungszusammenhang von wissenschaftlichen Aussagen

In dieser Arbeit werden aufgrund des Stands der Forschung des zu untersuchenden Themas gedankliche Bezugsrahmen als Vorstufe einer erklärenden Theorie erarbeitet. Obwohl der Entdeckungszusammenhang im Vordergrund steht, können gedankliche Bezugsrahmen bereits praktische Gestaltungshilfe leisten.[111] Folglich hat diese Arbeit den Anspruch, sowohl theoretische Aussagen als auch *praxeologische Aussagen* zu formulieren, „die einem praktisch Handelnden unmittelbare Hilfestellung für seine Problemlösung zu bieten vermögen“[112].
Nach Kubicek bestehen praxeologische Aussagen aus vier Elementen. Diese Elemente und ihre Beziehungen zueinander werden in der Abbildung 1-3 dargestellt und im Folgenden beschrieben:[113]

- *Gestaltungsziele* steuern den Gestaltungsprozess. Sie beeinflussen zum einen die Problemidentifikation, da sie den Fokus auf die Zustände lenken, die geändert werden sollen. Zum anderen sind die für die Bewertung unterschiedlicher Gestaltungsmaßnahmen erforderlich.
- *Rahmenbedingungen der Gestaltung* erfassen situative Faktoren, welche einen Einfluss auf die Wirkung von Gestaltungsmaßnahmen haben. Diese Bedingungen müssen berücksichtigt werden, weil sie im Rahmen der organisatorischen Gestaltung nicht verändert werden können. Folglich können sie auch als Restriktionen der Gestaltung aufgefasst werden.
- Eine *Gestaltungsmaßnahme* ist eine Menge zusammenhängender organisatorischer Regelungen, die darauf abzielen, das Verhalten der Betroffenen in einer gegebenen Situation auf die Gestaltungsziele auszurichten.
- Eine *Gestaltungswirkung* beschreibt die Wirkung einer Gestaltungsmaßnahme auf die Verhaltensweise der Betroffenen. Sie kann zeitlich unterschieden werden als erwartete

[111] Vgl. hierzu die Ausführungen in Kapitel 1.3.4.1.
[112] Grochla /Organisationstheorie/ 70.
[113] Vgl. zu den nachfolgenden Punkten Kubicek /Organisationsforschung/ 17-21.

Wirkung, sofern die Durchführung der getroffenen Gestaltungsmaßnahme bevorsteht, oder als tatsächliche Wirkung, wenn die Maßnahme bereits durchgeführt wurde.

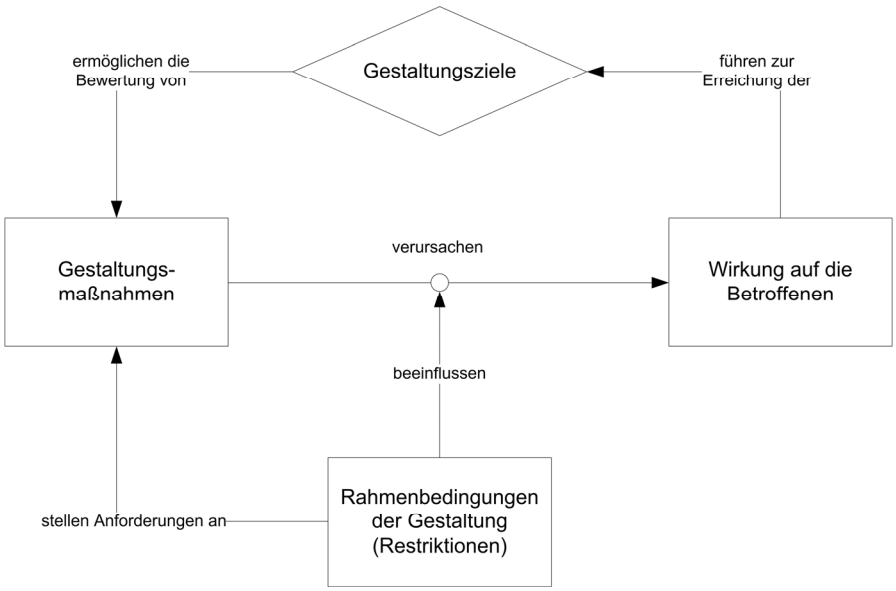


Abbildung 1-3: Beziehungen zwischen den Elementen praxeologischer Aussagen[114]

Im Sinne des dieser Arbeit zugrunde liegenden kritischen Realismus wird von einem *Fallibilismus* ausgegangen.[115] Dieser besagt, dass die Vernunft des Menschen fehlbar ist und folglich selbst durch die Anwendung der besten Forschungsmethoden erreichten Resultate sich als falsch erweisen können. Die Ergebnisse dieser Arbeit wie auch alle wissenschaftliche Ergebnisse müssen daher stets als revidierbar betrachtet werden, da eine Wahrheitsgarantie niemals erzielbar ist. Dies ist damit vereinbar, dass Ergebnisse sich trotzdem hochgradig bewähren können. Das Ziel dieser Arbeit ist daher, die Schwächen bisheriger Problemlösungen herauszuarbeiten und bessere Lösungen zu erzielen. Ihre Ergebnisse bleiben aber stets für eine Revision offen.

1.4 Aufbau und Vorgehensweise der Arbeit

Die vorliegende Arbeit besteht aus drei inhaltlich abgrenzbaren Teilen, sofern vom ersten Kapitel abgesehen wird. Im Einzelnen sind es die Grundlagen der Arbeit, die begründete Konzeption eines fehlerbasierten Verfahrens und die exemplarische Anwendung dieses Verfahrens. Die Abbildung 1-4 zeigt, wie die Arbeit aufgebaut ist. In dieser Abbildung ist zusätzlich

[114] In Anlehnung an Kubicek /Organisationsforschung/ 22.

[115] Vgl. zu diesem Absatz Albert /Wissenschaftstheorie/ 4677 und Gadenne /Rationalismus/ 8.

eingetragen, welche Zielfragen[116] durch welche Kapitel beantwortet werden. Im Anschluss an die Abbildung werden die drei Teile der Arbeit und jeweils die Vorgehensweise erörtert.

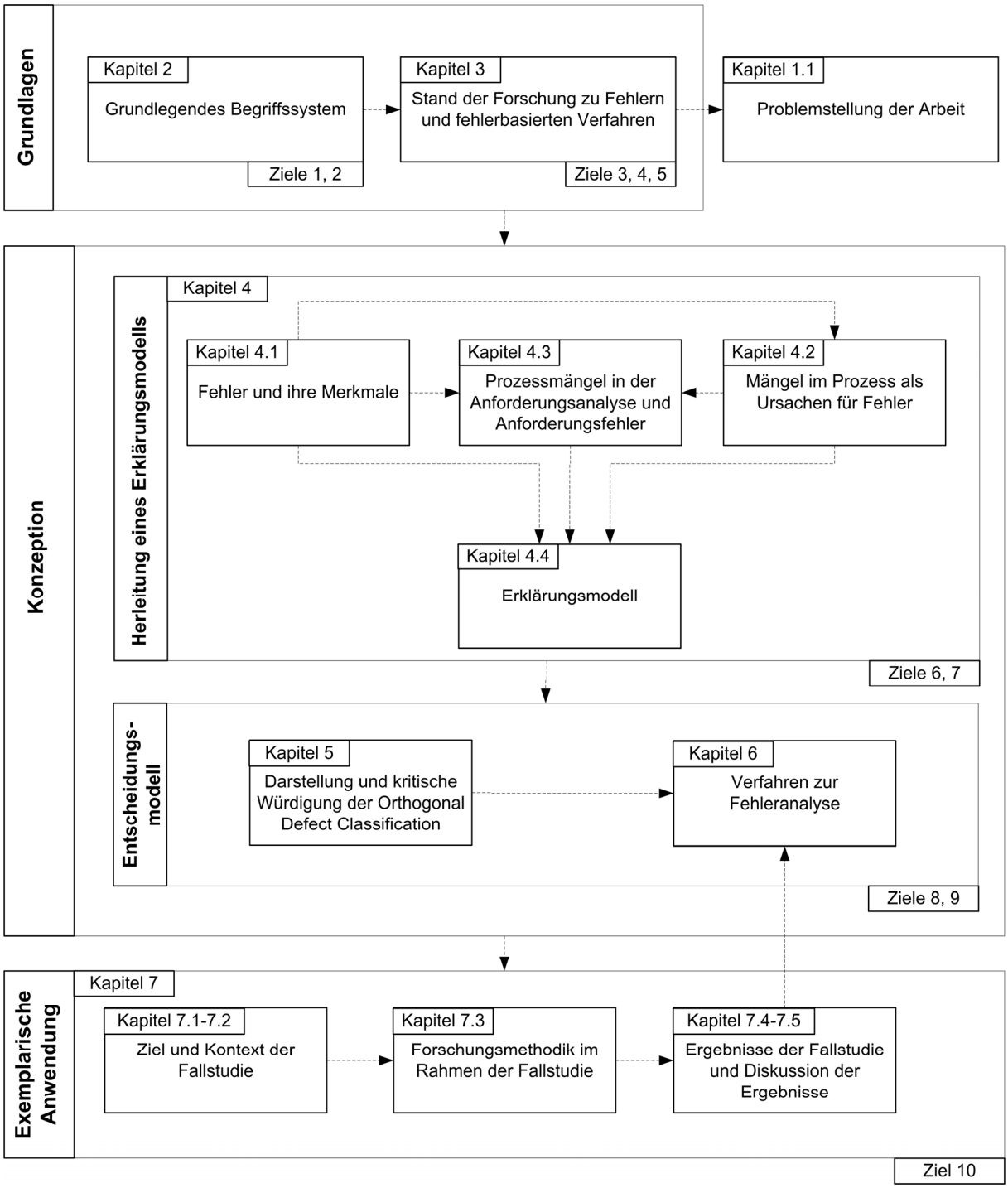


Abbildung 1-4: Aufbau der Arbeit

[116] Vgl. hierzu Kapitel 1.2.

Grundlagen der Arbeit

Kapitel 2 und 3 schaffen die Grundlagen der Arbeit. In Kapitel 2 wird ein grundlegendes Begriffssystem erarbeitet. Zentrale Begriffe dieser Arbeit wie Prozess, Fehler, Anforderungsanalyse und Qualitätssicherung werden dort definiert und für eine weitere Präzisierung oder Beschreibung realer Phänomene vorbereitet. Das Kapitel 3 stellt den Stand der Forschung zu Fehlern und fehlerbasierten Verfahren dar. Im Einzelnen wird ein Überblick über empirische Befunde zur Häufigkeit von Anforderungsfehlern und ihren Folgefehlern (Kapitel 3.2) sowie über den Aufwand zur Korrektur von Anforderungsfehlern (Kapitel 3.3) gegeben. Beide Unterkapitel untermauern mit empirischen Ergebnissen den Inhalt und die Relevanz des Praxisproblems dieser Arbeit.

Kapitel 3.4 stellt bestehende fehlerbasierte Verfahren vor, die den Anspruch erheben, auf die Ursachen von Fehler schließen zu können, indem Fehler erfasst und ausgewertet werden.

Konzeption des Verfahrens

In dieser Arbeit wird systematisch ein begründetes Verfahren hergeleitet, das Mängel im Prozess der Softwareentwicklung aufdeckt, indem es Fehler analysiert. In Kapitel 4 wird zunächst ein Erklärungsmodell herausgearbeitet, das erklärt, warum Anforderungsfehler und Folgefehler durch Mängel im Prozess der Anforderungsanalyse entstehen. Hierzu werden in Kapitel 4.1 Merkmale von Fehler beleuchtet und in Kapitel 4.2 das Konstrukt eines Prozessmangels erörtert. Anhand einer systematischen Literaturanalyse („literature review“) werden in Kapitel 4.3 empirische Befunde zu Zusammenhängen zwischen Prozessmängeln in der Anforderungsanalyse und dadurch verursachten Fehlern zusammengetragen. Mittels Induktion fließen die Erkenntnisse dieser drei Unterkapitel in Form von Gesetzeshypothesen in ein Erklärungsmodell ein, das Gegenstand von Kapitel 4.4 ist.

Aufbauend auf diesem Erklärungsmodell wird ein Entscheidungsmodell entwickelt. Hierzu wird zunächst die Orthogonal Defect Classification, kurz ODC, dargestellt und kritisch gewürdigt (Kapitel 5). Die ODC ist ein Verfahren zur Fehleranalyse, dass darauf abzielt, durch die Erfassung und Auswertung von Entwurfs- und Implementierungsfehlern auf die Ursachen ihrer Entstehung zu schließen. Nach der Darstellung in Kapitel 5.1, erfolgt eine kritische Würdigung anhand einer systematischen Literaturanalyse empirische Befunde zu ODC (Kapitel 5.2).

In Kapitel 6 wird schließlich ein Verfahren als Entscheidungsmodell entwickelt, das auf der ODC aufbaut. Die Darstellung orientiert sich an den einzelnen Schritten des Entscheidungsmodells.

Exemplarische Anwendung des Verfahrens

Die Durchführbarkeit des hergeleiteten Verfahrens wird in Kapitel 7 anhand einer explorativen Fallstudie untersucht.

Die Arbeit schließt mit einem Fazit in Kapitel 8 ab.

2 Grundlegendes Begriffssystem

2.1 Einführung in das Begriffssystem

In Kapitel 2 wird als gedanklicher Bezugsrahmen[117] ein Begriffssystem entwickelt, um reale Phänomene zu beschreiben.[118] Hierzu werden *begriffliche Aussagen* erarbeitet, die als relevant erachtete Phänomene problemadäquat erfassen sollen.[119] Entsprechende Aussagen haben zwei Funktionen. Erstens wird mit begrifflichen Aussagen festgehalten, welche Merkmale realer Erscheinungen betrachtet werden sollen, während andere Merkmale vernachlässigt werden sollen. Begriffliche Aussagen sind daher ein „unvollständiges Abbild realer Objekte“[120] und tragen damit zur Komplexitätsreduktion bei. Zweitens dienen sie dazu, Konventionen für den wissenschaftlichen Sprachgebrauch festzulegen. Sie sind eine Voraussetzung für die wissenschaftliche Beschäftigung mit realen Phänomenen.

Software soll Bedürfnisse der Kunden erfüllen und hat folglich keinen Selbstzweck. Aus diesen Bedürfnissen können Anforderungen an die Software abgeleitet werden. Nachfolgend werden in Kapitel 2.2.1 zunächst die Begriffe Software, Kunden der Software und Softwareanforderungen definiert. Bevor eine Software eingesetzt werden kann, muss sie entwickelt werden. Kapitel 2.2.2 hat begriffliche Aussagen zur Softwareentwicklung zum Gegenstand. In Kapitel 2.2.3 wird ein vereinfachtes Aufgabenmodell der Softwareentwicklung hergeleitet. Die Wartung einer entwickelten Software steht im Mittelpunkt des Kapitels 2.2.4.
Während eines Softwareentwicklungsprozesses können Anforderungsfehler entstehen, welche Folgefehler verursachen können. Fehler im Allgemeinen und Anforderungsfehler im Besonderen sowie ihre Ursachen im Entwicklungsprozess werden in Kapitel 2.2.5 beleuchtet. Anforderungsfehler entstehen in der Anforderungsanalyse. Sollen sie vermieden werden, ist folglich dort anzusetzen. Ist dies nicht möglich, sollte man bestrebt sein, sie und ggf. ihre Folgefehler möglichst früh zu finden. Dies steht wiederum im Mittelpunkt der Qualitätssicherung. Die Anforderungsanalyse und die Qualitätssicherung werden aufgrund ihrer besonderen Relevanz für diese Arbeit jeweils in den Kapiteln 2.3 und 2.4 einzeln behandelt.

---

[117] Vgl. hierzu Kapitel 1.3.4.1.

[118] Vgl. Grochla /Organisationstheorie/ 62 f. und Heinzl /Aktivitätsniveau/ 6.

[119] Vgl. zum Rest dieses Absatzes Grochla /Organisationstheorie/ 69 und Kubicek /Organisationsforschung/ 78, 80-93.

[120] Kubicek /Organisationsforschung/ 88 f.

2.2 Grundlagen der Softwareentwicklung

2.2.1 Software, Kunden und Softwareanforderungen

Software und Kunden der Software

Software wird im Rahmen dieser Arbeit verstanden als „Gesamtheit der Programme, die zum Betreiben einer bestimmten (Klasse von) Hardware und der Erfüllung einer bestimmten (Klasse von) Informationsaufgabe(n) benutzt wird“[121]. Sie kann unterschieden werden in Anwendungssoftware, Systemsoftware und systemnahe Software.[122] Eine *Anwendungssoftware* erfüllt oder unterstützt bestimmte Datenverarbeitungsaufgaben, die aus den Zielen des Anwenders abgeleitet werden.[123] In Abgrenzung hierzu umfasst *Systemsoftware* die Programme, „die für eine spezielle Hardware oder eine Hardwarefamilie entwickelt wurden, um den Betrieb und die Wartung dieser Hardware zu ermöglichen bzw. zu erleichtern“[124]. Folglich richtet sich eine Systemsoftware nach den Eigenschaften der Hardware, während bei einer Anwendungssoftware Probleme des Anwenders im Mittelpunkt stehen. *Systemnahe Software* orientiert sich bis zu einem gewissen Grad sowohl an Anwenderproblemen als auch an einer bestimmten Hardware(-klasse).[125] Hierzu zählen z. B. Netzwerkmanagement-Software, Datenbanken und so genannte „eingebettete“ Software, die in physische Produkte integriert wird. Ein *Anwendungssystem* besteht aus einer Anwendungssoftware und den zugehörigen Daten.[126] Zusätzlich umfasst es zum Betrieb der Anwendungssoftware erforderliche Hardware-Komponenten, systemnahe Software, Systemsoftware und Kommunikationseinrichtungen.

Anwendungssoftware wird für Anwender und Benutzer entwickelt und hat folglich keinen Selbstzweck. Als *Anwender* werden organisatorische Einheiten verstanden, die Software zur Erfüllung ihrer fachlichen Aufgaben einsetzen.[127] *Benutzer* sind jene Personen, die unmittelbar mit der Software arbeiten.[128] In dieser Arbeit wird der Begriff „*Kunde*“ als Oberbegriff für Anwender und Benutzer verwendet.[129]

---

[121] Seibt /Organisation/ 23.

[122] Für eine umfassende Klassifikation nach unterschiedlichen Kriterien siehe Gerhardt /Strategie/ 40-62.

[123] Vgl. Balzert /Software-Entwicklung/ 23.

[124] Balzert /Software-Entwicklung/ 23 .

[125] Vgl. Gerhardt /Strategie/ 48 f.

[126] Vgl. Stahlknecht, Hasenkamp /Wirtschaftsinformatik/ 226 f.

[127] Vgl. Balzert /Software-Entwicklung/ 24.

[128] Vgl. Balzert /Software-Entwicklung/ 24.

[129] Zur Kritik hinsichtlich einer ausschließlichen Fokussierung auf den Benutzer siehe auch Herzwurm /Softwareproduktentwicklung/ 207.

Softwareanforderungen

Anwendungssoftware soll den vorgesehenen Kunden einen Nutzen stiften, indem die Software Datenverarbeitungsaufgaben erfüllt oder unterstützt. Folglich werden die Bedürfnisse der Kunden durch die Frage konkretisiert, welche Daten auf welche Weise verarbeitet werden sollen. Aus diesen Kundenbedürfnissen werden Softwareanforderungen hergeleitet.[130]

Viele Autoren, die sich mit Softwareanforderungen befassen, definieren diesen Begriff dem allgemeinen Sprachgebrauch von „Anforderung“ folgend sehr umfassend. Nahezu jede Aussage eines Kunden, die in irgendeiner Weise in Bezug zu der zu erstellenden Software steht, ist dann eine Softwareanforderung. So überrascht es nicht, dass Klassifikationen von Softwareanforderungen neben Anforderungen hinsichtlich des funktionalen und qualitativen Verhaltens der Software auch beispielsweise Vorgaben für die Durchführung der Softwareentwicklung (Vorgehensmodelle, Methoden, Werkzeuge etc.), Vorgaben für die Qualitätssicherung, Einführung und Wartung sowie terminliche Vorgaben der Softwareerstellung enthalten.[131] Sicherlich sind die genannten Punkte für den Erfolg eines Softwareentwicklungsvorhabens wichtig. Es ist aber nicht hilfreich, völlig unterschiedliche Aspekte unter dem gleichen Begriff zusammenzufassen, wenn man konkrete Aussagen zu Softwareanforderungen machen möchte. In dieser Arbeit wird der Begriff der Softwareanforderung enger gefasst und im Folgenden herausgearbeitet.

Wird eine Software ausgeführt, weist sie *externe Merkmale* auf. Sie stellt einem Benutzer oder einer Hardware bestimmte Funktionen zur Verfügung.[132] Diese Funktionen bietet sie mit bestimmten Ausprägungen hinsichtlich qualitativer Eigenschaften an, wie z. B. Antwortzeitverhalten und Benutzbarkeit. Diese externen funktionalen und qualitativen Merkmale einer Software können durch einen Benutzer wahrgenommen bzw. im Fall von eingebetteter Software durch eine Hardware bestimmt werden. Sie werden deshalb als extern bezeichnet. *Interne Merkmale* einer Software dagegen betreffen ihre innere Struktur. Hierzu zählen strukturelle Merkmale wie Algorithmen, Prozeduren, interne Schnittstellen etc. Diese strukturellen Merkmale besitzen ebenfalls qualitative Eigenschaften wie beispielsweise Erweiterbarkeit, Wartbarkeit oder Wiederverwendbarkeit. Interne Merkmale, seien es strukturelle oder qualitative, können nur durch Einblick in den Quellcode, das Datenbankschema oder sonstigen software-

---

[130] Kundenbedürfnisse begründen Softwareanforderungen und sind folglich nicht miteinander zu verwechseln. Vgl. hierzu Sommerville /Software Engineering/ 48.

[131] Siehe hierzu zum Beispiel Partsch /Requirements-Engineering/ 22-26 oder Kotonya, Sommerville /Requirements Engineering/ 188-193.

[132] In Anlehnung an Mellis /Projektmanagement/ 65 f.

technischen Elementen bestimmt werden.[133] Die Ausprägungen interner Merkmale einer Software werden im Entwurf und in der Implementierung so festgelegt, dass die Software die gewünschten Ausprägungen der externen Merkmale in der geplanten Laufzeitumgebung zeigt.
Eine *Softwareanforderung*, kurz Anforderung, beschreibt ein gewünschtes externes Merkmal einer Software unabhängig von Entwurf und Implementierung dieses Merkmals. Funktionale Softwareanforderungen beschreiben hierbei externe funktionale Merkmale einer Software, d. h. auf welche Eingaben jeweils durch welche Ausgabe geplant reagiert werden soll. Qualitative Softwareanforderungen beschreiben die gewünschte Ausprägung einer qualitativen Eigenschaft einer oder mehrerer funktionaler Softwareanforderungen. Sie werden häufig auch sehr vage als nicht-funktionale Anforderungen bezeichnet.
Soll die zu erstellende Software z. B. auf einer bestimmten Hardware-Plattform laufen oder mit bestehender Software zusammenarbeiten, sind solche *Entwicklungseinschränkungen* gemäß der Definition keine Softwareanforderungen, weil sie Aspekte der Implementierung festlegen. Da Entwicklungseinschränkungen jedoch die Möglichkeiten der technischen Umsetzung von Anforderungen begrenzen, müssen sie möglichst früh mit den Anforderungen erkannt werden.

2.2.2 Softwareentwicklung als Aufgabe und als Prozess

Aufgabe versus Prozess

Nach Kosiol setzt eine *Aufgabe* ein Ziel für zweckbezogene menschliche Handlungen.[134] Sie beschreibt „eine durch physische oder geistige Aktivitäten zu verwirklichende Soll-Leistung“[135]. In diesem Sinne ist die Softwareentwicklung eine Aufgabe mit dem Sachziel bzw. Zweck, eine Software für Kunden zu erstellen, die den Bedürfnissen dieser Kunden genügt.
Neben dem Sachziel muss für eine Aufgabe geklärt werden, wie sie durchgeführt werden soll, um dieses Sachziel zu erreichen. Unter einem *Prozess* wird hier die Durchführung einer Aufgabe verstanden. Folglich beschreibt ein *Softwareentwicklungsprozess*, kurz *Softwareprozess*, wie die Aufgabe einer Softwareentwicklung durchgeführt werden soll (Soll-Prozess) oder wird (Ist-Prozess).
Ein Prozess übernimmt das Sachziel der ihm zugrunde liegenden Aufgabe. Ihm können zudem *Formalziele* zugeordnet werden, die jeweils Vorstellungen über erwünschte Zustände

---

[133] Für eine Beschreibung von Qualitätsattributen aus Entwicklersicht siehe Raasch /Systementwicklung/ 30-37.
[134] Vgl. hierzu und zu nachfolgendem Satz Kosiol /Aufgabenanalyse/ 68.
[135] Vgl. Hoffmann /Aufgabe/ 200.

oder Vorgehensweisen beschreiben, wie das Sachziel umgesetzt werden soll.[136] Dies können beispielsweise Vorgaben hinsichtlich der Entwicklungsdauer, der Entwicklungskosten, der Effizienz des Prozesses oder der Stabilität des Prozesses sein.[137]

Gestaltung eines Softwareentwicklungprozesses

Gestaltungsmaßnahmen sind Mittel, mit deren Hilfe ein Softwareentwicklungsprozess gestaltet werden kann.[138] Ein *Gestaltungsbereich* ist eine Zusammenfassung inhaltlich zusammenhängender Gestaltungsmaßnahmen. Es können grundsätzlich organisatorische und fachlich-technische Gestaltungsmaßnahmen unterschieden werden. *Fachlich-technische Gestaltungsmaßnahmen* beantworten Fragen dahingehend, welche Tätigkeiten mit welchen Methoden und Werkzeugen zu erledigen sind. *Organisatorische Gestaltungsmaßnahmen* haben dagegen zum einen die Differenzierung der Gesamtaufgabe in Teilaufgaben (Arbeitsteilung) und zum anderen die Abstimmung voneinander abhängiger Teilaufgaben im Blickpunkt.

Der in der vorliegenden Arbeit verwandte Begriff des Prozesses beschränkt sich nicht wie im ursprünglichen Sinne auf den Ablauf. Vielmehr kann er sich auf jeden organisatorischen oder fachlich-technischen Gestaltungsbereich erstrecken, der die Durchführung der Aufgabe beschreibt, so z. B. auf den Einsatz von Methoden und Techniken, die Kommunikation zwischen Aufgabenträgern, Anforderungen an die Aufgabenträger etc. Ein Prozess kann folglich auch als eine Menge von Maßnahmen aus verschiedenen Gestaltungsbereichen aufgefasst werden.[139]

Prozessorientiertes Softwarequalitätsmanagement

Der Begriff des Softwareprozesses wird häufig einseitig verknüpft mit dem Capability Maturity Model (CMM). Modelle wie das CMM geben Richtlinien für die Gestaltung von Softwareentwicklungsprozessen. Mit diesen Richtlinien werden die Formalziele verfolgt, Prozesse planbar, kontrollierbar und steuerbar zu machen. Mellis und Stelzer fassen das CMM und vergleichbare Modelle unter dem Begriff des prozessorientierten Softwarequalitätsmanagements, kurz PSQM, zusammen.[140] Neben dem genannten Sachziel der Softwareentwicklung, eine

---

[136] In Anlehnung an Staehle /Management/ 438. Staehle verwendet das Begriffspaar Zweck und Ziele statt Sachziel und Formalziele.

[137] Vgl. Stelzer /Software-Qualitätsmanagements/87-97.

[138] Vgl. zu diesem Absatz Stelzer /Software-Qualitätsmanagements/97-124

[139] Im Kontext des Capability Maturity Model (CMM) wird im Einklang mit obiger Auffassung ein Softwareprozess beispielsweise wie folgt definiert: „a software process can be defined as a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products.“ Paulk u. a. /Capability Maturity Model/ 3.

[140] Vgl. Mellis, Stelzer /Rätsel/ 31-33.

Software für Kunden zu erstellen, die den Bedürfnissen dieser Kunden genügt, wird beim PSQM der Begriff des Prozesses mit weiteren Formalzielen assoziiert. Mellis und Stelzer zeigen auf, dass diese Formalziele nicht für alle Unternehmen angemessen sind.[141] In dieser Arbeit wird deshalb der Begriff des Prozesses lediglich mit dem genannten Sachziel verknüpft und ist unabhängig von Formalzielen. Folglich wird der Prozess hier allgemeiner verstanden als im Sinne des PSQM. Somit geben auch zum PSQM alternative Ansätze wie das Extreme Programming Empfehlungen für die Gestaltung eines Softwareentwicklungsprozesses, auch wenn der Begriff des Prozesses bisweilen in diesem Zusammenhang in der Literatur vermieden wird. Sie betonen zum Teil andere Gestaltungsbereiche oder legen andere Gestaltungsmaßnahmen nahe.[142]

2.2.3 Aufgabenmodell der Softwareentwicklung

In dieser Arbeit wird der Prozess der Anforderungsanalyse untersucht. Dieser ist wiederum ein Teilprozess einer Softwareentwicklung. Da viele Vorschläge für Softwareentwicklungsprozesse existieren und diese sich erheblich unterscheiden können, ist es hilfreich, von einem konkreten Prozess zu abstrahieren. Hierzu wird ausgehend von der Softwareentwicklung als Aufgabe eine verrichtungsorientierte Aufgabenanalyse durchgeführt.[143] Das Ergebnis ist ein vereinfachtes Modell der Entwicklungsaufgaben einer Softwareentwicklung, von welchem als gemeinsamem Nenner ausgegangen werden kann. Dies ermöglicht zudem die Anforderungsanalyse als Teilaufgabe der Softwareentwicklung besser einzuordnen und abzugrenzen.

Wie bereits oben eingeführt verfolgt die Softwareentwicklung das Ziel, eine Software für Kunden zu erstellen, die den Bedürfnissen dieser Kunden genügt. Folglich kann die Softwareentwicklung als das Lösen eines Problems aufgefasst werden, bei dem die Bedürfnisse der Kunden der Ausgangspunkt sind (siehe Abbildung 2-1).[144] Das Lösen des Problems schließt mit ein, dass die Kundenbedürfnisse identifiziert werden, die durch die Nutzung einer Software befriedigt werden sollen. Ausgehend von den Kundenbedürfnissen wird ein Konzept für die Lösung erstellt. Mit der Umsetzung dieses Konzepts liegt dann im Idealfall eine Lösung vor, welche die Kundenbedürfnisse zufrieden stellt.

---

[141] Vgl. Mellis, Stelzer /Rätsel/ 33-38.

[142] Vgl. z. B. Beck /Change/ 70-77.

[143] Vgl. Frese /Aufgabenanalyse/ 208-214 und Kosiol /Aufgabenanalyse/ 66-77.

[144] Vgl. zu diesem Absatz Roman /Requirements engineering/ 14.

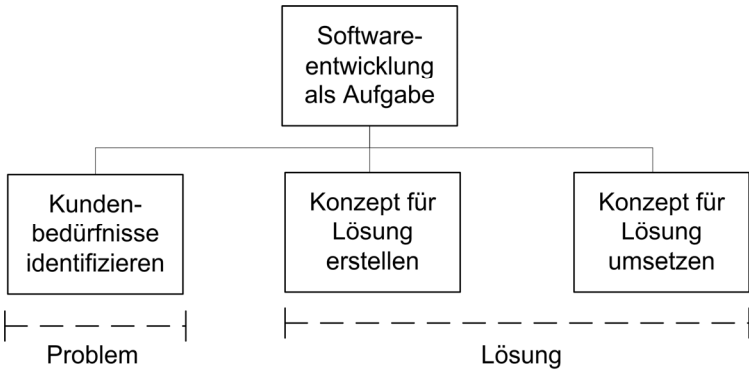


Abbildung 2-1: Vereinfachtes Aufgabenmodell der Softwareentwicklung

Die Softwareentwickler lassen in das Konzept für die Lösung einfließen, wie sie die Kundenbedürfnisse wahrnehmen und verstehen. Das Konzept beinhaltet zum einen Softwareanforderungen. Diese externen Merkmale der Software werden gemäß Definition technikneutral beschrieben. Zum anderen erfolgt während der Erstellung des Lösungskonzepts auch der Softwareentwurf.[145] Im Rahmen des Grobentwurfs bzw. Architekturentwurf wird die Architektur der Software abgeleitet und dokumentiert. Hierzu werden Komponenten der Software identifiziert und festgelegt, was diese leisten sollen, sowie die Beziehungen und Schnittstellen zwischen den Komponenten bestimmt. Im Feinentwurf wird die Struktur einzelner Komponenten als Vorbereitung für ihre Umsetzung beschrieben. Hierzu zählen beispielsweise verwendete Datenstrukturen oder Algorithmen in einer Komponente. Der Softwareentwurf hat folglich interne Merkmale der Software zu Gegenstand und ist daher technikabhängig. Das Konzept umzusetzen bedeutet schließlich die Software zu implementieren. Der Übergang zwischen dem Softwareentwurf und der Implementierung ist zum Teil fließend und nicht trennscharf.

Die Abbildung 2-2 fasst das Aufgabenmodell der Softwareentwicklung zusammen.

[145] Die Ausführungen zum Softwareentwurf sind angelehnt an IEEE /SWEBOK/ [3-1].

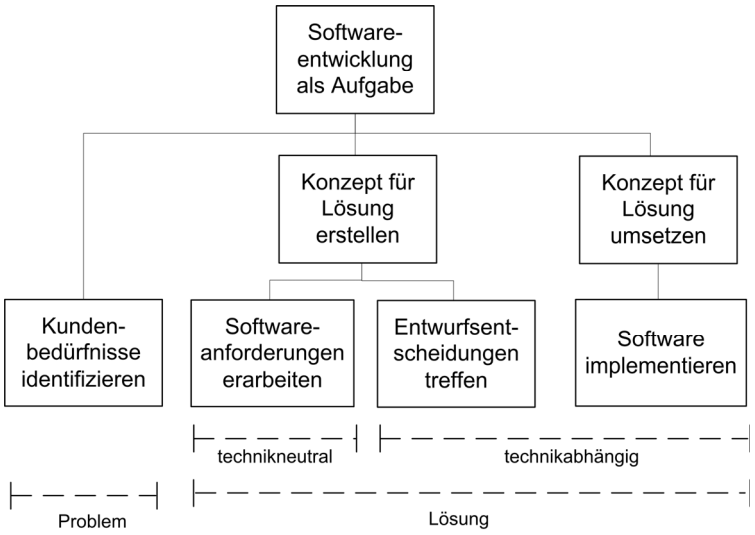


Abbildung 2-2: Erweitertes Aufgabenmodell der Softwareentwicklung

Das vereinfachte Aufgabenmodell der Softwareentwicklung berücksichtigt nur Entwicklungsaufgaben[146], die unmittelbar auf die Erreichung des Sachziels der Softwareentwicklung ausgerichtet sind. Die Qualitätssicherung unterstützt die Entwicklungsaufgaben bei Ihrer Zielerreichung.[147] Als Unterstützungsaufgabe hat sie das Ziel, die Qualität der produzierten Software sicherzustellen. Der Begriff der Softwareentwicklung wird in dieser Arbeit so verwendet, dass er sowohl die Entwicklungsaufgaben als auch die Qualitätssicherung umfasst.

Ausgehend vom Aufgabenmodell kann eine Softwareanforderung als eine Entscheidung in der Anforderungsanalyse aufgefasst werden. Als solche ist sie die Grundlage für eine oder mehrere Entscheidungen im Entwurf. Eine Entwurfsentscheidung schafft wiederum die Voraussetzung für eine oder mehrere Entscheidungen in der Implementierung. Folglich kann die Softwareentwicklung als eine Vielzahl von Entscheidungsketten verstanden werden, mit abhängigen Entscheidungen aus der Anforderungsanalyse, dem Entwurf und der Implementierung. Dieser Zusammenhang zwischen einer Anforderung und mehreren Entwurfs- und Implementierungsentscheidungen ist grundlegend und unterscheidet eine professionelle Softwareentwicklung von einer unprofessionellen. Er gilt unabhängig davon, wie die Anforderungsanalyse, der Entwurf und die Implementierung im Prozess der Softwareentwicklung zeitlich relativ zueinander ablaufen. Dieser Zusammenhang liegt folglich sowohl in sequenziellen Entwicklungsansätzen wie dem V-Modell als auch in agilen Entwicklungsansätzen wie dem Extreme Programming vor.[148]

---

[146] Vgl. hierzu und im Folgenden Stelzer /Software-Qualitätsmanagements/ 98-104.

[147] Die Qualitätssicherung wird in Kapitel 2.4 erörtert.

[148] Vgl. Trittmann /Agile Softwareprojekte/ 69-71.

Bezeichnungen für Aufgabenträger der Softwareentwicklung

Die Tabelle 2-1 führt in dieser Arbeit verwandte Bezeichnungen für Aufgabenträger der Softwareentwicklung auf.

Aufgabe	Bezeichnung für Augabenträger
Anforderungsanalyse[149]	Systemanalytiker
Softwareentwurf	Softwaredesigner
Implementierung	Programmierer
Softwareentwicklung	Softwareentwickler

Tabelle 2-1: Bezeichnungen für Aufgabenträger der Softwareentwicklung

Einer Person können auch mehrere Teilaufgaben übertragen werden, so dass sie beispielsweise Systemanalytiker und Softwaredesigner zugleich ist. Die Bezeichnung des Softwareentwicklers wird hier als Oberbegriff für die übrigen Bezeichnungen verwandt. Wird sie im Plural eingesetzt, umfaßt sie alle Aufgabenträger einer Softwarewicklung gemäß dem Aufgabenmodell. Jeder Systemanalytiker, Softwaredesigner und Programmierer ist also zugleich ein Softwareentwickler.

Ein Entwicklungsteam umfasst alle Projektmitarbeiter, die Entwicklungsaufgaben oder die Qualitätssicherung[150] durchführen.

2.2.4 Softwarelebenszyklus

Der *Softwarelebenszyklus* ist der Zeitraum, der die Entwicklung, den Betrieb und die Wartung der Software umfasst.[151] Er endet, wenn die Software nicht mehr betrieben wird.

Mit Abschluss der Softwareentwicklung kann eine Anwendungssoftware in Betrieb genommen werden. Während des Betriebszeitraums der Software können sich Kundenbedürfnisse ändern oder neue hinzukommen. Folglich können sich auch Softwareanforderungen ändern oder neue aufkommen. Unter der *Wartung* einer Software versteht man, dass die Software nach ihrer Inbetriebnahme aufgrund geänderter Anforderungen angepasst oder aufgrund neuer Anforderungen weiterentwickelt wird.[152] In der Regel werden mit der Wartung auch entdeckte Fehler in der Software korrigiert. Sofern sich Änderungen an der Software ausschließ-

[149] Vgl. zu dieser Teilaufgabe Kapitel 2.3.

[150] Die Qualitätssicherung wird in Kapitel 2.4 erörtert.

[151] Vgl. zu diesem Absatz Stahlknecht, Hasenkamp /Wirtschaftsinformatik/ 236 und Seibt /Systemlebenszyklus/ 456 f.

[152] Vgl. IEEE /Software Maintenance/ 4 und Stahlknecht, Hasenkamp /Wirtschaftsinformatik/ 235 f.

lich auf die Korrektur von Fehler beschränken, wird statt von der Wartung von der *Pflege* einer Software gesprochen.[153]

Sofern die Wartung nicht ad hoc durchgeführt wird, ist sie vergleichbar mit einer Softwareentwicklung.[154] Statt jedoch eine Software von Grund auf neu zu entwickeln, hat die Wartung eine neue *Version* (synonym *Release*) der Software zum Ergebnis. Die Wartung als Aufgabe weist dann die dargestellten Entwicklungsaufgaben einer Softwareentwicklung auf: Anforderungsanalyse, Entwurf und Implementierung.

2.2.5 Fehler in der Softwareentwicklung und ihre Ursachen

Fehler können während der gesamten Softwareentwicklung entstehen. Sie sind erkennbar als unbefriedigende Merkmale eines sichtbar existierenden Arbeitsergebnisses wie beispielsweise eines Anforderungsdokuments oder Softwarecodes. Konkret ist ein *Fehler* ein unzureichendes Merkmal oder ein erwartetes, jedoch fehlendes Merkmal eines Arbeitsergebnisses der Softwareentwicklung, sofern es eine Änderung in diesem Ergebnis notwendig macht.[155] In der englischen Literatur wird ein Fehler häufig als „error", „defect" oder „bug" bezeichnet und inhaltlich sehr unterschiedlich definiert und abgegrenzt.[156]

Es können Anforderungs-, Entwurfs- und Implementierungsfehler voneinander abgegrenzt werden, je nachdem, ob Entscheidungen in der Anforderungsanalyse, dem Entwurf oder der Implementierung betrachtet werden. Insbesondere werden hier also unter einem Fehler nicht ausschließlich Implementierungsfehler aufgefasst.

Aufgrund des Zusammenhangs zwischen Entscheidungen der verschiedenen Teilaufgaben der Softwareentwicklung kann sich ein Anforderungsfehler als Entwurfs- und Implementierungsfehler fortsetzen. Diese besonderen Entwurfs- und Implementierungsfehler sind also *Folgefehler*. Ist ein Fehler kein Folgefehler, wird er nachfolgend als *originärer Fehler* bezeichnet.

Die *Fehlerverfolgung* ist eine Aktivität, bei der bestimmte Merkmale von Fehlern erfasst und verwaltet werden.[157] Ein Fehlerbericht beinhaltet die Merkmale einschließlich der Ausprägungen eines konkreten Fehlers. Die Fehlerverfolgung wird meist unterstützt durch eine Software, welche nachfolgend als Fehlerverfolgungssoftware bezeichnet wird.[158] Die uneinheitliche Verwendung der englischen Begriffe „error", „defect" und „bug" setzt sich auch für die

---

[153] Vgl. Stahlknecht, Hasenkamp /Wirtschaftsinformatik/ 236.

[154] Vgl. zu diesem Absatz Seibt /Systemlebenszyklus/ 457.

[155] In Anlehnung an der Definition des Begriffs „defect" in Chillarege /Defect/ 362 f.

[156] Vgl. Fenton, Pfleeger /Software metrics/156 f. und Fenton, Neil /Critique/ 678 f.

[157] Vgl. IEEE /SWEBOK/ [5-10].

[158] Fehlerverfolgungssoftware werden umfassend in Kaner, Falk, Nguyen /Computer Software/ 87-120 erörtert und dort als „problem tracking system" bezeichnet.

Fehlerverfolgung fort. Sie wird entsprechend als „error tracking“, „defect tracking“ oder „bug tracking“ bezeichnet.

Ein Prozess beschreibt, wie eine Aufgabe durchgeführt wird.[159] Fehler entstehen während der Durchführung von Aufgaben. Anforderungsfehler entstehen folglich in einem Prozess der Anforderungsanalyse. Die Gestaltungsmaßnahme eines Prozesses, die Fehler und ggf. Folgefehler verursacht, wird nachfolgend als ein *Mangel im Prozess* oder kurz *Prozessmangel* bezeichnet. Entsprechend werden Anforderungsfehler und ihre Folgefehler durch Mängel im Prozess der Anforderungsanalyse hervorgerufen.

2.3 Anforderungsanalyse

2.3.1 Definition der Anforderungsanalyse

Obwohl sich die Anforderungsanalyse ungefähr seit Mitte der siebziger Jahre als eigenständige Aufgabe der Softwareentwicklung etabliert hat und seitdem zahlreiche Veröffentlichungen erschienen sind,[160] gibt es bis heute keinen Konsens bei der Definition grundlegender Begriffe.[161] Dies zeigt sich vor allem an dem Begriff der Anforderungsanalyse beziehungsweise dem englischen Begriff „*Requirements Engineering*“.

Häufig wird die Anforderungsanalyse definiert als eine Aufgabe mit dem Ziel, ein Anforderungsdokument zu erstellen. So schreiben zum Beispiel Sommerville und Sawyer: „A requirements engineering process is a structured set of activities which are followed to derive, validate and maintain a systems requirements document.“[162] Die Dokumentation von Anforderungen ist jedoch ein Gestaltungsbereich der Anforderungsanalyse. Als solches kann ein Anforderungsdokument zum Beispiel hinsichtlich des Detaillierungs- und Formalisierungsgrades unterschiedliche Ausprägungen haben, die sich einander ausschließen können.

Folglich ist das Anforderungsdokument keine Konstante der Anforderungsanalyse und somit als Ziel ungeeignet. Vielmehr ist es zweckmäßig die Anforderungsanalyse als Aufgabe mit Zielen zu definieren, die unabhängig von bestimmten Gestaltungsbereichen sind. In diesem Zusammenhang grenzen Ross und Schoman bereits 1977 die Anforderungsanalyse mit ihrer Definition gut ab: „*requirements definition is a careful assessment of the needs, based on*

---

[159] Vgl. hierzu Kapitel 2.2.2.

[160] Vgl. Boehm /Requirements Engineering/ 255.

[161] Vgl. beispielsweise Davis /Software requirements/ 21 f.

[162] Sommerville, Sawyer /Requirements Engineering/ 9. Sommerville und Sawyer fassen unter dem Begriff „system requirements specification“ das Dokument zusammen, das neben Softwareanforderungen und –restriktionen zusätzlich die Hardwareanforderungen festlegt.

current or foreseen conditions, which may be internal operations or an external market. It must say what system features will serve and satisfy this context. And it must say how the system is to be constructed. “[163]

In Anlehnung an diese Definition wird in dieser Arbeit unter Anforderungsanalyse eine Teilaufgabe der Softwareentwicklung verstanden, die die Ziele hat

1. die Kundenbedürfnisse zu identifizieren, zu verstehen und abzustimmen, die die zu erstellende Software erfüllen soll und somit den zu erbringenden Nutzen zu definieren,
2. bestehende Entwicklungseinschränkungen[164] des Kunden zu erkennen,
3. Softwareanforderungen zu erheben, so dass die zu erstellende Software den geplanten Nutzen leisten kann und
4. die identifizierten Anforderungen so aufzubereiten, dass sie angemessen in den übrigen Aufgaben der Softwareentwicklung verwertet werden können.

Die Abbildung 2-3 stellt grafisch dar, wie sich die Anforderungsanalyse gemäß der Definition in das Aufgabenmodell der Softwareentwicklung[165] einordnen lässt.

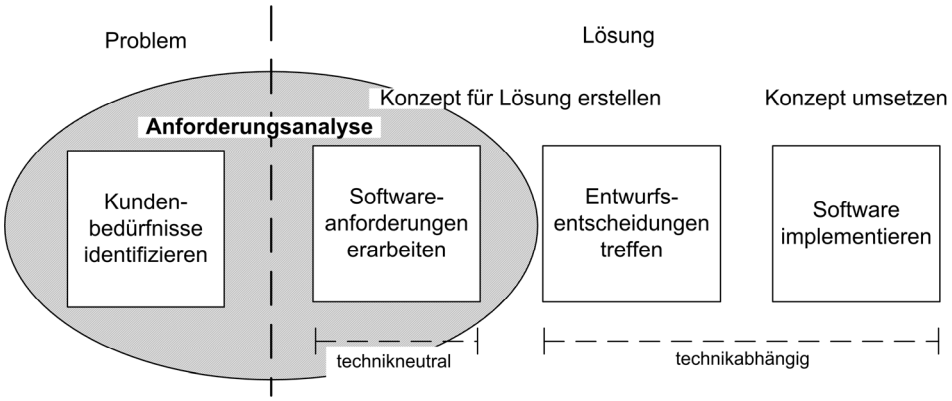


Abbildung 2-3: Einordnung der Anforderungsanalyse

Die Einordnung der Anforderungsanalyse zeigt die Besonderheiten dieser Aufgabe auf. Im Gegensatz zum Softwareentwurf und der Implementierung erstreckt sich die Anforderungsanalyse sowohl auf die Domäne des Problems und damit der Domäne der Kunden, als auch die der Lösung und damit die Domäne der Softwareentwickler. Hierdurch erwachsen besondere Herausforderungen, da diese Domänen beispielsweise durch unterschiedliche Begriffswelten und Perspektiven gekennzeichnet sind. In der Regel sind diese Domänen auch unterschiedlichen Organisationen oder Organisationseinheiten zugeordnet.

---

[163] Ross, Schoman /Structured Analysis/ 6 f.

[164] Siehe zu diesem Begriff Kapitel 2.2.1.

[165] Vgl. hierzu Kapitel 2.2.3.

2.3.2 Teilaufgaben der Anforderungsanalyse

Genauso wie die Softwareentwicklung kann auch die Anforderungsanalyse anhand einer verrichtungsorientierten Aufgabenanalyse in Teilaufgaben zerlegt werden.[166] In der Literatur werden häufig vier Teilaufgaben unterschieden: Anforderungen erheben, dokumentieren, valdieren und verwalten.[167]

Um Softwareanforderungen zu *erheben*, müssen Kundenbedürfnisse und Entwicklungseinschränkungen verschiedener Kundengruppen erkannt und verstanden werden. Aus den Kundenbedürfnissen werden dann Softwareanforderungen abgeleitet.

Damit auf der Grundlage von Softwareanforderungen Entscheidungen getroffen werden können, müssen sie einschließlich der Entwicklungseinschränkungen[168] *dokumentiert* werden. Das Ergebnis wird häufig vereinfacht als Anforderungsdokument, Anforderungsspezifikation oder kurz Spezifikation bezeichnet. Praktisch gesehen können es aber mehrere Dokumente sein, die jeweils Teilbereiche der Software beschreiben. Ebenso können Anforderungen beispielsweise auch in einer Datenbank „dokumentiert" werden. Insofern ist es präziser, ein Anforderungsdokument für eine Software als eine Menge dokumentierter Anforderungen für diese Software zu verstehen. Der Ausdruck „spezifizierte Anforderungen" wird synonym für „dokumentierte Anforderungen" verwendet.

Anforderungsdokumente können sich von Projekt zu Projekt hinsichtlich mehrerer Merkmale deutlich voneinander unterscheiden. So können beispielsweise Anforderungen unterschiedlich formal beschrieben sein. Das Spektrum reicht von einer natürlich-sprachlichen Beschreibung über eine semi-formale Beschreibung[169] bis zu einer formalen Beschreibung. Ebenso können Anforderungsdokumente sich dadurch abgrenzen, wie detailliert die einzelnen Anforderungen beschrieben werden. So kann z. B. für funktionale Softwareanforderungen jeweils einzeln angegeben sein, für welche Eingaben welche Ausgaben erzeugt werden. Oder aber Softwareanforderungen werden grob über die Kundenbedürfnisse dokumentiert, d. h. streng genommen werden Softwareanforderungen rudimentär oder gar nicht dokumentiert. Dies ist beispielsweise bei agilen Entwicklungsansätzen verbreitet, wo Kundenbedürfnisse in Form von Anwendungsfällen dokumentiert werden. Nach Cusumano und Selby geht auch Microsoft mit seinem Ansatz der aktivitätsbezogenen Planung auf diese Weise vor.[170]

---

[166] Vgl. hierzu Kapitel 2.2.3.

[167] Vgl. hierzu und den folgenden Absätzen Pohl /Requirements engineering/ 16-21 und IEEE /SWEBOK/ [2-4] - [2-9].

[168] Vgl. Kapitel 2.2.1.

[169] Die Unified Modeling Language, kurz UML, ist eine verbreitete semiformale Beschreibungsmethode.

[170] Vgl. Cusumano, Selby /Microsoft-Methode/ 230-237.

Die Existenz einer Anforderung gründet sich auf einem Kundenbedürfnis. Eine Anforderung ohne ein zugehöriges Kundenbedürfnis impliziert, dass für ein Merkmal der Software, das vom Kunden nicht gewünscht wird, Ressourcen eingesetzt werden. Folglich wird die Umsetzung einer derartigen Anforderung auch vom Kunden nicht honoriert, weil sie ihm keine Nutzensteigerung bringt. Um solche Fälle zu vermeiden, müssen Anforderungen *validiert* werden, d. h. sie müssen hinsichtlich ihrer Eignung zur Befriedigung der Kundenbedürfnisse untersucht werden.

Spätesten wenn Softwareanforderungen freigegeben werden, also als Voraussetzung für Entscheidungen in den übrigen Aufgaben der Softwareentwicklung zur Verfügung stehen, ist es zweckmäßig, Änderungen bestehender Softwareanforderungen und die Aufnahme neuer Softwareanforderungen zu *verwalten*. Diese Teilaufgabe der Anforderungsanalyse wird häufig als Anforderungsmanagement bezeichnet.[171] Definierte Verfahren sollen vermeiden, dass Änderungen und Neuaufnahmen von Anforderungen zu Widersprüchen zwischen Anforderungen führen und ggf. erforderliche Änderungen anderer Ergebnisse der Softwareentwicklung übersehen werden.

Die beschriebenen Teilaufgaben können in einem Prozess der Anforderungsanalyse nicht klar voneinander abgrenzt werden, da sie in hohem Maße ineinander greifen.[172] Häufig werden sie iterativ und parallel für verschiedene Teilmengen von Anforderungen durchgeführt.

2.4 Qualitätssicherung

2.4.1 Definition der Qualitätssicherung

Das vereinfachte Aufgabenmodell der Softwareentwicklung aus Kapitel 2.2.3 berücksichtigt nur Entwicklungsaufgaben[173], die unmittelbar auf die Erreichung des Sachziels der Softwareentwicklung ausgerichtet sind. Die *Qualitätssicherung* unterstützt die Entwicklungsaufgaben bei Ihrer Zielerreichung. Als Unterstützungsaufgabe hat sie das Ziel, die Qualität der produzierten Software sicherzustellen.[174]

Was genau ist aber die Qualität einer Software? Sowohl in der Theorie als auch Praxis gibt es bei der Beantwortung dieser Frage keinen Konsens. Zum einen gibt es verschiedene Perspektiven auf die Qualität eines Produktes.[175] Diese können zwei grundlegenden Kategorien

---

[171] Vgl. Mead /Requirements Management/ 4-8.

[172] Vgl. Houdek, Pohl /Requirements Engineering/ 987.

[173] Vgl. hierzu und im Folgenden Stelzer /Software-Qualitätsmanagements/ 98-104.

[174] Vgl. Mellis /Projektmanagement/ 181 f.

[175] Vgl. hierzu und für den folgenden Satz Hoyer, Hoyer /Quality/ 53-62.

zugeordnet werden: Qualität als Übereinstimmung eines Produktes mit seinen spezifizierten Anforderungen („conformance to specifications“) oder Qualität als Eignung für den vorgesehenen Anwendungszweck („fitness for purpose“). Zum anderen ist die Qualität selbst ein mehrdimensionaler Begriff mit Dimensionen wie z. B. Sicherheit, Zuverlässigkeit oder Laufzeiteffizienz.[176] Die Wichtigkeit einzelner Dimensionen kann sich in verschiedenen Projekten stark unterscheiden und in manchen sogar vernachlässigbar gering sein.[177] Folglich ist es nach Liggesmeyer für ein Softwareentwicklungsvorhaben zweckmäßig, statt der „besten“ Qualität eine „richtige“ Qualität anzustreben.

Die Fehlerarmut ist ein Merkmal einer Software, der sowohl in der Theorie als auch Praxis eine besondere Beachtung geschenkt wird.[178] Sie kann einerseits als eine Dimension der Qualität betrachtet werden. Andererseits kann bei einer geeigneten Definition des Fehlerbegriffs die Mehrdimensionalität der Softwarequalität abgebildet werden.[179] So definiert beispielsweise Dromey einen Qualitätsfehler („quality defect“) als ein Merkmal eines Ergebnisses, das eine gewünschte Qualitätsdimension dieses Ergebnisses beeinträchtigt.[180] Aufgrund der variierenden Bedeutung einer Qualitätsdimension in verschiedenen Projekten kann so ein Merkmal in einem Projekt als ein Fehler wahrgenommen werden und in einem anderen wiederum nicht. Im Folgenden wird daher unter der Qualität eines Arbeitsergebnisses, sei es ein Zwischenergebnis oder die Software selbst, seine Fehlerarmut verstanden. Dies ist auch im Einklang mit empirischen Untersuchungen, welche die Produktqualität als das Verhältnis zwischen dem Produktumfang und der Fehleranzahl messen.[181]

Aufgrund der vorhergehenden Überlegungen kann die Definition der Qualitätssicherung präzisiert werden. Die Qualitätssicherung hat das Ziel, Fehler der Software oder ihrer Zwischenergebnisse zu erkennen und zu lokalisieren.[182]

2.4.2 Teilaufgaben der Qualitätssicherung

Mittels einer verrichtungsorientierten Aufgabenanalyse[183] kann die Qualitätssicherung in die zwei Teilaufgaben Prüfen und Testen zergliedert werden.

---

[176] Vgl. Mellis /Projektmanagement/ 181 f.

[177] Vgl. hierzu und im Folgenden Liggesmeyer /Software-Qualität/ 1.

[178] Vgl. Mellis /Projektmanagement/ 182.

[179] Siehe zu Begriff des Fehlers Kapitel 4.1.

[180] Vgl. Dromey /Software product quality/ 146 f.

[181] Vgl. Harter, Krishnan, Slaughter /Process maturity/ 454, 461-464 und Harter, Slaughter /Quality improvement/ 786, 791-797.

[182] Vgl. Wallmüller /Software-Qualitätssicherung/ 24.

[183] Vgl. Frese /Aufgabenanalyse/ 208-214 und Kosiol /Aufgabenanalyse/ 66-77.

Mit der Teilaufgabe des *Prüfens* versucht man, Fehler in einem Prüfobjekt zu erkennen und zu lokalisieren, ohne es auszuführen.[184] Vielmehr wird das Prüfobjekt in der Darstellungsform untersucht, in der es erstellt wird (z. B. Programmcode, natürlich-sprachlicher Text, Diagramme). In diesem Sinne ist das Prüfen statisch. Sofern das Prüfobjekt wie im Fall eines Anforderungsdokuments kein Programmcode ist, besteht ohnehin nicht die Möglichkeit der Ausführung. Bekannte Prüftechniken sind zum Beispiel Inspektionen und Reviews.[185]
Softwareanforderungen können beispielsweise dahingehend geprüft werden, ob aufgrund unterschiedlicher Vorstellungen von Kundengruppen Anforderungen einander widersprechen.
Im Unterschied dazu beschreibt die Teilaufgabe des *Testens*, dass eine Software ausgeführt wird, um ein mögliches Fehlverhalten[186] der Software zu erkennen. In diesem Sinne ist das Testen dynamisch. Ein Fehlverhalten ist dabei ein Symptom eines oder mehrerer Fehler. Folglich versucht man über die Analyse von Fehlverhalten Fehler zu erkennen und zu lokalisieren. Zu den verbreiteten Testtechniken zählen zum Beispiel der Anweisungsüberdeckungstest oder die funktionale Äquivalenzklassenbildung.[187]
Die Teilaufgabe des Testens kann weiter differenziert werden in einzelne Teststufen. Eine Teststufe ist eine Gruppe von Testaktivitäten, die gemeinsam ausgeführt und verwaltet werden.[188] Teststufen unterscheiden sich durch Testobjekt, Testumgebung oder Testziel.[189]

2.5 Zusammenfassung

Im Kapitel 2 wurde das grundlegende Begriffssystem der Arbeit herausgearbeitet. Seine wesentlichen Bestandteile werden nun kurz zusammengefasst.
Die Softwareentwicklung ist eine Aufgabe mit dem Sachziel, eine Software für Kunden zu erstellen, die den Bedürfnissen dieser Kunden genügt. Diese Aufgabe kann zerlegt werden in die grundlegenden Entwicklungsaufgaben der Anforderungsanalyse, des Entwurfs und der Implementierung. Mit der Anforderungsanalyse verfolgt man das Ziel, die Kundenbedürfnisse und darauf aufbauend die Softwareanforderungen zu erheben und schließlich die Anforderungen für weitere Teilaufgaben aufzubereiten. Die Qualitätssicherung unterstützt die Entwicklungsaufgaben, indem sie das Ziel verfolgt, Fehler der Software oder ihrer Zwischenergebnisse zu erkennen und zu lokalisieren. Unter der Wartung einer Software versteht man, dass die

[184] Vgl. zu diesem Absatz Müller /Prüf- und Testprozesse/ 20 und die dort aufgeführten Quellen.
[185] Vgl. hierzu zum Beispiel Liggesmeyer /Software-Qualität/ 285-300.
[186] Der Begriff des Fehlverhaltens wird in Kapitel 4.1 definiert.
[187] Vgl. hierzu zum Beispiel Liggesmeyer /Software-Qualität/ 47-53, 81-83.
[188] Vgl. International Software Testing Qualification Board /Standard glossary/ 31.
[189] Vgl. Spillner, Linz /Softwaretest/ 42-64.

Software nach ihrer Inbetriebnahme aufgrund geänderter Anforderungen angepasst oder aufgrund neuer Anforderungen weiterentwickelt wird

Ein Prozess beschreibt, wie eine Aufgabe durchgeführt wird. So kann man beispielsweise vom Prozess der Anforderungsanalyse oder der Qualitätssicherung sprechen, sofern festgehalten werden soll, wie in einem konkreten Fall die Anforderungsanalyse oder die Qualitätssicherung durchgeführt werden oder werden sollen. Für ein bestimmtes Projekt mit den gegebenen Rahmenbedingungen kann eine Maßnahme im Prozess einer Entwicklungsaufgabe einen Prozessmangel darstellen, sofern sie das Entstehen von Fehlern begünstigt. Werden diese nicht entdeckt, können zudem im weiteren Verlauf Folgefehler entstehen. Fehler sind dabei als unzureichende oder fehlende Merkmale eines Arbeitsergebnisses der Softwareentwicklung zu verstehen, das aufgrund dessen einer Änderung bedarf.

3 Stand der Forschung zu Fehlern und fehlerbasierten Verfahren

3.1 Einführung

In Kapitel 3 wird der Stand der Forschung herausgearbeitet, der für das Verständnis des Ursache-Wirkungs-Zusammenhangs zwischen Mängeln im Prozess der Anforderungsanalyse (Ursache) sowie den Anforderungsfehlern und ihren Folgefehlern (Wirkung) relevant ist. Hierzu werden zunächst empirische Erkenntnisse zu den Thesen des Praxisproblems aufgezeigt, das dieser Arbeit zugrunde liegt. Diese Thesen lauten:[190]

> *These 1 (Inhalt des Praxisproblems): Mängel in einem Prozess der Anforderungsanalyse verursachen viele Anforderungsfehler und Folgefehler.*
>
> *These 2 (Relevanz des Praxisproblems): Je später ein Anforderungsfehler entdeckt und behoben wird, desto aufwändiger wird die Korrektur.*

Die Häufigkeit von Anforderungsfehlern und ihren Folgefehlern (These 1) ist Gegenstand des Kapitels 3.2 und der Aufwand für die Korrektur von Anforderungsfehlern (These 2) wird in Kapitel 3.3 behandelt.

Das Kapitel 3.4 stellt Vorschläge für fehlerbasierte Verfahren in der wissenschaftlichen Literatur vor. Diese Verfahren sind mit dem Anspruch verbunden, dass auf die Ursachen von Fehlern geschlossen werden kann, indem Fehler erfasst und ausgewertet werden.

3.2 Häufigkeit von Anforderungsfehlern und ihren Folgefehlern

Ein zentrales Problem kennzeichnet die Praxis der Anforderungsanalyse:

> *Mängel in einem Prozess der Anforderungsanalyse verursachen viele Anforderungsfehler und Folgefehler.*

Empirische Arbeiten aus dem Zeitraum von Beginn der siebziger Jahre bis heute stützen die These des Praxisproblems.[191] Zum einen ist diese Erkenntnis also nicht neu. Zum anderen

---

[190] Vgl. Kapitel 1.1.2.

[191] Die These für den Inhalt des Praxisproblems ist kausaler Natur. Kausale Aussagen beschreiben, dass zwei oder mehrere Größen sich beeinflussen und in welche Richtung der Einfluss jeweils wirkt. Im Kapitel 3.2 erfolgt der erste Schritt zur Begründung der These. Anhand empirischer Befunde wird gezeigt, dass ein *statistischer* Zusammenhang zwischen Prozessen der Anforderungsanalyse und Fehlern besteht. Ein statistischer Zusammenhang ist zwar notwendig, aber nicht hinreichend für einen *kausalen* Zusammenhang. Zwei Größen können auch dann statistisch zusammenhängen, wenn dies inhaltlich nicht gerechtfertigt ist. Um einen kausalen Zusammenhang zu begründen, müssen sachlogische Überlegungen herangezogen werden und die Richtung der Wirkung erklärt werden. Dies ist Gegenstand des Erklärungsmodells in Kapitel 4 (insbesondere Kapitel 4.2).

fällt auf, dass sich das Praxisproblem beständig über die Jahrzehnte halten konnte. Entscheidend aber ist, dass es *weiterhin* besteht. Es kann deshalb treffend als ein chronisches Problem bezeichnet werden.[192]

Nachfolgend werden die Ergebnisse veröffentlichter empirischer Arbeiten zur Häufigkeit von Anforderungsfehlern und ihren Folgefehlern chronologisch nach dem Jahr der Veröffentlichung dargestellt.[193]

Betrachtet man die Ergebnisse in ihrer Gesamtheit, können *zwei Folgerungen* gezogen werden.[194] Erstens untermauern alle Studien unabhängig voneinander die These des Praxisproblems, obwohl sie unterschiedliche Rahmenbedingungen haben. Zweitens stellen Anforderungsfehler und ihre Folgefehler in den meisten beschriebenen Studien die größte Fehlergruppe dar.

Empirische Befunde aus den siebziger Jahren

Bereits in den siebziger Jahren erscheinen erste empirische Arbeiten, in denen Fehlerhäufigkeiten getrennt nach verschiedenen Fehlerarten ermittelt werden. Einen Beitrag leisten in diesem Zusammenhang insbesondere die Untersuchungen von Rubey, Endres sowie von Bell und Thayer.

Rubey teilt 1202 gefundene Fehler aus 11 Testaktivitäten von verschiedenen Organisationen in Kategorien ein.[195] Unvollständige und fehlerhafte Spezifikationen bilden mit insgesamt 340 Fehlern die Fehlerkategorie mit der häufigsten Nennung. 28% der Fehler haben demnach ih-

---

[192] Vgl. Avcı, Wagner /Anforderungsanalyse/ 279.

[193] Empirische Arbeiten mit quantitativen Daten über Häufigkeiten von Fehlern im Allgemeinen und über Anforderungsfehler im Besonderen gibt es nur wenige. Ausgehend von einer Literaturrecherche in wissenschaftlichen Zeitschriften (vgl. hierzu Kapitel Tabelle 4-1) und verschiedenen Literaturdatenbanken (EBSCOhost, ACM Digital Library, IEEE Xplore digital library, Google Scholar und CiteSeer Scientific Literature Digital Library) wurden Beiträge gesucht, die entsprechende empirische Untersuchungen zum Gegenstand haben oder auf solche verweisen. Ergebnis der Literaturrecherche sind 11 empirische Beiträge, von denen 4 in Konferenzbänden, 4 in wissenschaftlichen Zeitschriften und 3 in anderer Form veröffentlich worden sind.

Beiträge, denen keine systematische empirische Untersuchung zugrunde liegt, wurden in diesem Kapitel nicht berücksichtigt. So z. B. Tavolato, Vincena /Prototyping/ 435 f. und Sheldon u. a. /Reliability measurement/ 19. Leider fallen auch interessante Arbeiten von Capers Jones in diese Kategorie (z. B. Jones /Software Costs/ 425).

[194] Beim Vergleich dieser Ergebnisse ist zu berücksichtigen, dass die Autoren der empirischen Studien unterschiedliche Ansätze verfolgen, um einen Fehler zu definieren und damit auch zu zählen. Folglich sind die Fehlerhäufigkeiten verschiedener Arbeiten nicht direkt miteinander vergleichbar. Ebenso können die Ergebnisse der einzelnen empirischen Arbeiten nicht verallgemeinert werden, weil die Anzahl der untersuchten Projekte jeweils klein ist. Dies schließt jedoch nicht die Möglichkeit aus, bestimmte Schlüsse aus der gemeinsamen Betrachtung aller empirischen Befunde zu folgern.

[195] Vgl. Rubey /Software validation/ 248-249.

ren Ursprung in der Anforderungsanalyse. Nach Endres können sogar 46% von 432 in einem Betriebssystem gefundenen Fehlern der Anforderungsanalyse zugeordnet werden.[196]
Bell und Thayer werten die Fehlerberichte einer Echtzeitsoftware für ein Raketenabwehrsystem aus.[197] Die Anforderungsspezifikation führt insgesamt 8248 Softwareanforderungen auf. Aus den betrachteten Fehlerberichten können die Autoren 972 Anforderungsfehler bestimmen. Folglich liegt durchschnittlich ein Fehler pro 8,5 Anforderungen vor.

Empirische Befunde aus den achtziger Jahren

Zu ähnlichen Ergebnissen wie die oben aufgeführten drei empirischen Arbeiten kommen auch Untersuchungen, die in den achtziger Jahren durchgeführten worden sind. Basili und Perricone analysieren Änderungsanträge für eine Software zur Planung von Satellitenmissionen.[198] Der überwiegende Teil der Änderungsanträge (62%) sind Anträge für Fehlerkorrekturen. 48% der auf diese Weise identifizierten Fehler haben ihre Ursache in fehlerhaften oder falsch interpretierten Softwareanforderungen[199].
Ostrand und Weyuker führen bei der Entwicklung eines kommerziellen Editors einen Änderungsbericht ein, um Fehlerdaten zu sammeln und anschließend auszuwerten.[200] 12% von 171 gefundenen Fehlern sind den Autoren nach eine Folge von unklaren, unvollständigen oder widersprüchlichen Spezifikationen. Diese Fehlerursache wird in der Untersuchung als zweithäufigste aufgeführt.
McGarry u. a. berichten von 11 Softwareentwicklungen für die NASA aus den Jahren 1985 bis 1990 mit einem Gesamtaufwand von 65 Personenjahren.[201] Das Software Engineering Laboratory[202] hat diese Projekte näher untersucht und insbesondere über 2000 Fehler klassifiziert sowie deren Ursache bestimmt. Demnach haben 20% der Fehler ihren Ursprung in der Anforderungsanalyse.

---

[196] Endres verwendet nicht die Bezeichnung „requirements engineering“, sondern schreibt: „Almost half of errors (46%) are found in the area of understanding the problem, of problem communication, of the knowledge of possibilities and procedures for problem solving.“ Vgl. Endres /Analysis/ 331.

[197] Vgl. Bell, Thayer /Software requirements/ 61-68.

[198] Vgl. Basili, Perricone /Software errors/ 42-52.

[199] In Basili, Perricone /Software errors/ 45 heißt es: „[…] 48 percent of errors was attributed to incorrect or misinterpreted functional specifications or requirements“. Da die Autoren es versäumen, die Begriffe „funktionale Spezifikationen“ und „Anforderungen“ abzugrenzen, wird hier vereinfacht „Softwareanforderung“ als Oberbegriff verwendet.

[200] Vgl. Ostrand, Weyuker /Software Error Data/ 290-298.

[201] Vgl. McGarry u. a. /Software Process Improvement/ 21-25. Die untersuchten Projekte werden auf der S. 22 der Quelle aufgelistet. Die Autoren führen weitere Hinweise zu den Projekten in der Tab. B-1 auf S. 60 auf.

[202] Das Software Engineering Laboratory (SEL) hat von der Gründung im Jahre 1976 bis zur Auflösung im Jahre 2001 zahlreiche empirische Untersuchungen bei Softwareentwicklungen für die NASA durchgeführt. Ziel dieser Untersuchungen ist die Verbesserung der Prozesse gewesen. Einen Überblick über die Geschichte des SEL geben Basili u.a. /Lessons/ 69-79.

Empirische Befunde aus den neunziger Jahren

Auch mit Arbeiten aus den neunziger Jahren lässt sich die Reihe empirischer Untersuchungen fortsetzen, die mit quantitativen Ergebnissen die These des Praxisproblems untermauern.

Nakajo und Kume unterziehen 508 Fehler einer detaillierten Analyse, die während der Entwicklung zweier kommerzieller Softwareprodukte aufgetreten sind.[203] Bei den untersuchten Produkten handelt es sich um Software, die in technischen Geräten eingebettet ist und diese steuert. 169 Fehler (33%) sind Fehler in Modulfunktionen. Von diesen Funktionsfehlern können 45% auf falsch verstandene Softwareanforderungen zurückgeführt werden. Folglich haben 76 von 508 Fehlern (15%) ihre Ursache in der Anforderungsanalyse.

Lutz baut auf dem Konzept von Nakajo und Kume auf und untersucht 387 Fehler, die während der Integration und dem Systemtest in der eingebetteten Software der Raumsonden Voyager und Galileo entdeckt werden.[204] Die größte Teilmenge unter den 387 Fehlern machen mit einer Häufigkeit von 286 die Funktionsfehler aus, wobei auf Voyager 87 und auf Galileo 199 Fehler entfallen. Von diesen Funktionsfehlern werden 144 Fehler als sicherheitskritisch eingestuft (Voyager: 48 Fehler, Galileo: 96 Fehler). 62% der sicherheitskritischen Funktionsfehler bei Voyager und 79% bei Galileo sind durch übersehene oder falsch verstandene Softwareanforderungen verursacht.

Vinter u. a. prüfen die in einem Unternehmen verfügbaren Fehlerberichte mehrerer vorangegangener Projekte, in denen eingebettete Echtzeitsoftware entwickelt wurde.[205] Die häufigste Ursache für Fehler sind Probleme mit Softwareanforderungen. Den Autoren nach sind 36% von insgesamt 1100 Fehlern eine Folge dieser Probleme.

Aktuelle empirische Befunde

Auch aktuellere, nach dem Jahr 2000 erschienene Studien weisen auf den beschriebenen Missstand hin. So werten Lauesen und Vinter 200 Fehlerberichte einer Anwendungssoftware nach ihrer Auslieferung aus.[206] Insgesamt 54% dieser Fehlerberichte (107) haben Unzulänglichkeiten von Softwareanforderungen zum Gegenstand.

Im Rahmen einer weiteren Studie wurden Softwareunternehmen in Amerika befragt, die Software für die Fertigungsindustrie des Transportwesens (Automobil, Luftfahrt und Bahn) oder für den Finanzdienstleistungssektor entwickeln.[207] Diese Studie wurde durch das National

---

[203] Vgl. Nakajo, Kume /Case history analysis/ 830-838.

[204] Vgl. Lutz /Software requirements errors/ 126-133.

[205] Vgl. Vinter u.a. /Prevention/ 16-18.

[206] Vgl. Lauesen, Vinter /Requirement defects/ 37-50.

[207] Vgl. Tassey /Software Testing/ [6-1] bis [7-29].

Institute of Standards and Technology[208], kurz NIST, in Auftrag gegeben und 2002 veröffentlicht. Gemäß der Befragung bei 10 Softwareunternehmen entstehen bei Softwareentwicklungen für die Fertigungsindustrie des Transportwesens[209] insgesamt 15,6% der Fehler in der Anforderungsanalyse und dem Architekturentwurf.[210] Die analoge Befragung bei 4 Softwareunternehmen, die eingebettete Software und Anwendungssoftware für den Finanzdienstleistungssektor entwickeln, kommt sogar zum Ergebnis, dass 30,3% der Fehler in der Anforderungsanalyse und dem Architekturentwurf entstehen.[211]

Die Tabelle 3-1 fasst die aufgeführten quantitativen Ergebnisse zusammen.

[208] Das National Institute of Standards and Technology, kurz NIST, ist eine Bundesbehörde, die dem amerikanischen Handelsministerium unterstellt ist. Gemäß Eigendarstellung hat diese Behörde das Ziel, in Zusammenarbeit mit der Industrie Techniken, Messverfahren und Standards zu entwickeln und anzuwenden, um das wirtschaftliche Wachstum zu fördern. Insbesondere entwickelt das NIST Richtlinien für die nationale informationstechnische Infrastruktur. Für weitere Einzelheiten siehe http://www.nist.gov.

[209] Bei der entwickelten Software handelt es sich um folgende Arten: computer-aided design (CAD), computer-aided manufacturing (CAM), computer-aided engineering (CAE) und product data management (PDM). Für eine Erläuterung dieser Softwarearten siehe Tassey /Software Testing/ [B-1] bis [B-24].

[210] Vgl. Tassey /Software Testing/ [6-9] f. In der Tabelle 6-2 auf Seite [6-10] wird die Fehlerhäufigkeit von 15,6% aus Platzgründen ausschließlich der Anforderungsanalyse zugeordnet. Ein Blick auf den beigefügten Fragebogen auf den Seiten [C-8] und [C-9] verdeutlicht jedoch, dass nach „requirements gathering and analysis/architectural design“, also der Anforderungsanalyse im Sinne dieser Arbeit inklusive Architekturentwurf, gefragt wurde.

[211] Vgl. Tassey /Software Testing/ [7-8] bis [7-11]. Analog wie in der Fußnote 210 bereits erwähnt, muss die Fehlerhäufigkeit entgegen der Tabelle 7-4 auf Seite [7-11] in der Quelle nicht nur der Anforderungsanalyse, sondern der Anforderungsanalyse einschließlich Architekturentwurf zugeordnet werden, weil in dem in der Studie verwendetem Fragebogen entsprechend gefragt wurde (siehe Seiten [E-7] bis [E-9]).

Kurztitel der Studie	Veröffentlichungsjahr	Ergebnis
Rubey /software validation/	1975	28% der Fehler haben ihren Ursprung in der Anforderungsanalyse.
Endres /analysis/	1975	46% der Fehler haben ihren Ursprung in der Anforderungsanalyse.
Bell, Thayer /Software requirements/	1976	1 Fehler pro 8,5 Anforderungen.
Basili, Perricone /Software errors/	1984	48% der Fehler haben ihren Ursprung in der Anforderungsanalyse.
Ostrand, Weyuker /Software Error Data/	1984	12% der Fehler haben ihren Ursprung in der Anforderungsanalyse.
McGarry u. a. /Software Process Improvement/	1994[212]	20% der Fehler haben ihren Ursprung in der Anforderungsanalyse.
Nakajo, Kume /case history analysis/	1991	15% der Fehler haben ihren Ursprung in der Anforderungsanalyse.
Lutz /software requirements errors/	1993	62% bzw. 79% der sicherheitskritischen Funktionsfehler haben ihren Ursprung in der Anforderungsanalyse.
Vinter u. a. /prevention/	1996	36% der Fehler haben ihren Ursprung in der Anforderungsanalyse.
Lauesen, Vinter /requirement defects/	2001	54% der Fehler haben ihren Ursprung in der Anforderungsanalyse.
Tassey /Software Testing/	2002	15,6% der Fehler in Softwareentwicklungen für die Fertigungsindustrie des Transportwesens haben ihren Ursprung in der Anforderungsanalyse (hier inkl. Architekturentwurf). 30,3% der Fehler in Softwareentwicklungen für den Finanzdienstleistungssektor haben ihren Ursprung in der Anforderungsanalyse (hier inkl. Architekturentwurf).

Tabelle 3-1: Empirische Arbeiten zu Häufigkeiten von Anforderungsfehlern und Folgefehlern

Die Ergebnisse der empirischen Befunde aus Tabelle 3-1 stützen die These des Praxisproblems, das dieser Arbeit zugrunde liegt:

> *Mängel in einem Prozess der Anforderungsanalyse verursachen viele Anforderungsfehler und Folgefehler.*

3.3 Aufwand für die Korrektur von Anforderungsfehlern

Je später ein Anforderungsfehler entdeckt und behoben wird, desto aufwändiger ist seine Korrektur. Diese These zeigt, wie relevant das Praxisproblem ist. Sie wird zunächst theoretisch begründet und anschließend durch empirische Befunde belegt.

[212] Die Arbeit von McGarry u. a. /Software Process Improvement/ wurde 1994 veröffentlich. Die quantitativen Daten zu Fehlern beziehen sich aber auf 11 Softwareentwicklungsprojekte für die NASA aus den Jahren 1985 bis 1990.

Akkumulative Effekte eines Fehlers

Wie kann theoretisch begründet werden, dass die Korrektur eines Anforderungsfehlers aufwändiger wird, je später er entdeckt und korrigiert wird? Hierzu ist es zweckmäßig, eine Softwareanforderung als eine Entscheidung aufzufassen; eine Entscheidung, die eine Voraussetzung für weitere Entscheidungen darstellt.[213] So werden auf der Grundlage dieser Voraussetzung beispielsweise eine oder mehrere Entscheidungen im Entwurf getroffen. Auf diesen Entwurfsentscheidungen aufbauend werden wiederum Entscheidungen für die Implementierung gefasst. Das Dilemma liegt auf der Hand: je später ein Anforderungsfehler entdeckt wird, desto mehr falsche Entwicklungsentscheidungen werden als Folgefehler getroffen und umgesetzt (vgl. Abbildung 3-1). Mizuno bezeichnet dies als die akkumulativen Effekte eines Fehlers.[214]

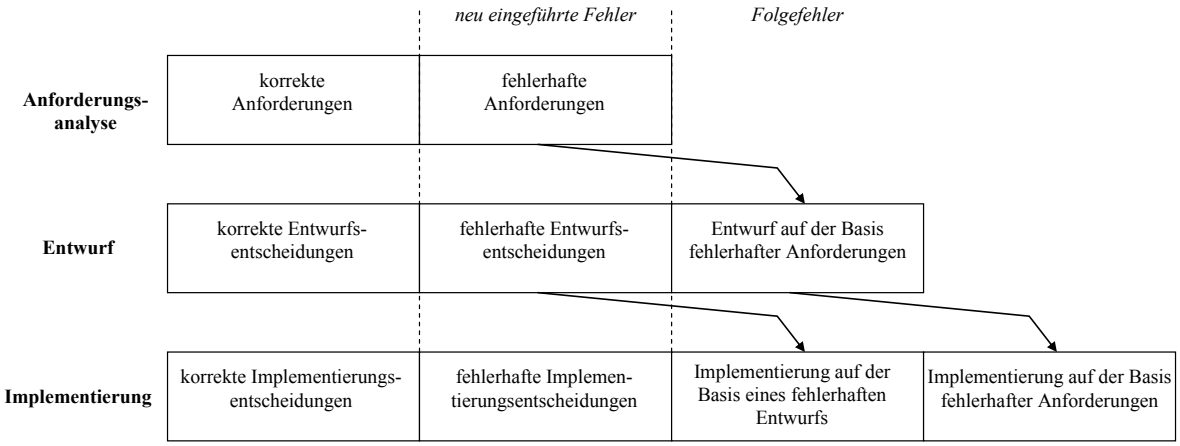


Abbildung 3-1: Akkumulative Effekte von Fehlern[215]

Der Aufwand für die Korrektur der Folgefehler muss dem Aufwand für die Korrektur des entsprechenden Anforderungsfehlers zugeordnet werden, weil die Folgefehler durch den Anforderungsfehler verursacht wurden. Dieser Zusammenhang kann auch verallgemeinert werden: Korrigiert man einen originären Fehler, muss auch der Aufwand für die Korrektur der entstandenen Folgefehler dem Korrekturaufwand des originären Fehlers zugerechnet werden, da sie durch ihn verursacht wurden.

Die Folgen der akkumulativen Effekte eines Anforderungsfehlers treten insbesondere dann deutlich auf, wenn aufgrund fehlender oder sonstiger fehlerhafter Anforderungen Änderungen erst nach Inbetriebnahme der Software umgesetzt werden. Dies ist in der Regel mit großen

[213] Vgl. hierzu auch Kapitel 2.2.3.

[214] Vgl. Mizuno /Software Quality/ 70.

[215] In Anlehnung an Mizuno /Software Quality/ 70.

Überarbeitungen verbunden. Der Aufwand für diese Überarbeitungen ist dem Entwicklungsprojekt zuzuordnen, sofern es sich nicht um Anforderungen handelt, die nach dem Abschluss des Entwicklungsprojekts neu aufgekommen sind. So liefern beispielsweise Stark, Skillicorn und Ameele Daten aus acht Freigaben von Softwareüberarbeitungen im Rahmen der Wartung.[216] Diese Freigaben umfassen 243 Änderungen und erzeugen einen Gesamtaufwand von 591 Personentagen. Obwohl nur 26 von 243 Änderungen durch Softwareanforderungen verursacht werden, beanspruchen diese Änderungen 20% des Gesamtaufwands und stellen somit den größten Aufwandsanteil.

Empirische Befunde zum Korrekturaufwand von Anforderungsfehlern

Dass die Korrektur eines Anforderungsfehlers aufwändiger wird, je später er entdeckt und behoben wird, belegen auch empirische Befunde. Hierbei sind insbesondere Studien interessant, die beschreiben, wie sich Faktoren für den Aufwand relativ ändern, falls die Korrektur eines Fehlers verzögert wird. Ein relativer Faktor beschreibt folglich, um welche Größe sich der Korrekturaufwand vervielfacht, wenn ein Fehler zu einem späteren Zeitpunkt korrigiert wird. Wichtiger als die konkreten Werte der relativen Faktoren ist der Trend, den sie aufzeigen: je später ein Fehler gefunden und behoben wird, desto aufwändiger wird die Korrektur. Alle dem Verfasser in diesem Zusammenhang bekannten empirischen Arbeiten werden nachfolgend kurz beschrieben.[217]

Leider wird in empirischen Literaturquellen häufig nicht geklärt, welche Aktivitäten zur Korrektur eines Fehlers gezählt werden. Dies ist aber unabdingbar, wenn man Aussagen zum Aufwand der Korrektur macht. Schließt die Korrektur eines Fehlers auch Prüf- und Testaktivitäten zum Finden des Fehlers ein? Oder wird nur die Lokalisierung eines Fehlers berücksichtigt, nachdem er wahrgenommen wurde? Umfasst die Korrektur auch erforderliche Maßnahmen, um den Fehler zu verstehen und Alternativen der Behebung zu erarbeiten? Diese Fragen bleiben meist unbeantwortet. Dieser Umstand muss bei der Interpretation und dem Vergleich der Ergebnisse berücksichtigt werden.

---

[216] Vgl. Stark, Skillicorn, Ameele /Examination/ 13.

[217] Empirische Arbeiten mit quantitativen Daten über Korrekturaufwand von Fehlern im Allgemeinen und von Anforderungsfehlern im Besonderen gibt es nur wenige. Ausgehend von einer Literaturrecherche in wissenschaftlichen Zeitschriften (vgl. hierzu Kapitel Tabelle 4-1) und verschiedenen Literaturdatenbanken (EBSCOhost, ACM Digital Library, IEEE Xplore digital library, Google Scholar und CiteSeer Scientific Literature Digital Library) wurden Beiträge gesucht, die entsprechende empirische Untersuchungen zum Gegenstand haben oder auf solche verweisen. Ergebnis der Literaturrecherche sind 4 empirische Beiträge. Beiträge, denen keine systematische empirische Untersuchung zugrunde liegt, wurden in diesem Kapitel nicht berücksichtigt.

Ebenso werden die Begriffe Aufwand und Kosten meist nicht angemessen differenziert in den entsprechenden Literaturquellen genutzt. Unter Aufwand wird hier der mengenmäßige Verbrauch personeller Arbeitsleistung zur Erfüllung einer Aufgabe verstanden.[218] Er wird gemessen als die Summe der eingesetzten Arbeitszeit der Personen. Wird die personelle Arbeitsleistung zum Beispiel durch die Multiplikation mit einem Verrechnungssatz monetär bewertet, erhält man als Wertgröße die Kosten.[219] Den Hauptteil der Entwicklungskosten einer Anwendungssoftware bilden die Personalkosten. Andere Kostengrößen wie z. B. Kosten für Entwicklungswerkzeuge und Hardware sind im Verhältnis zu den Personalkosten in der Regel vernachlässigbar. Nachfolgend werden insbesondere Aussagen darüber aufgegriffen, wie sich der Aufwand oder die Kosten einer Fehlerkorrektur *im Zeitverlauf relativ* ändern. Aufgrund der obigen Überlegungen wird unterstellt, dass relative Größenfaktoren für die Kosten einer Korrektur gleichermaßen für den Aufwand gelten.

Damm, Lundberg und Wohlin bestimmen auf der Grundlage von Expertenbefragungen den durchschnittlichen relativen Aufwand für Fehlerkorrekturen in verschiedenen Teilaufgaben und Teststufen.[220] Hierzu werden zwei unterschiedliche Wartungsprojekte bei Ericsson untersucht, in denen jeweils eine neue Version einer Software für mobile Netzwerke entwickelt wird. Der Aufwand für eine Fehlerkorrektur umfasst Tätigkeiten zur Verfolgung[221], Behebung und Nachtesten eines Fehlers. Die Tabelle 3-2 gibt einen Überblick über die durchschnittlichen relativen Aufwandsfaktoren. Der Aufwand für die Fehlerkorrektur steigt demnach vom Grobentwurf bis zum Betatest durchschnittlich um das 20-fache.

	Grobentwurf	**Feinentwurf und Implementierung**	**Integrationstest**	**Funktionstest**	**Systemtest**	**(einjähriger) Betatest**
Aufwandsfaktor	1x	1,2x	3x	9,8x	16x	20x

Tabelle 3-2: Durchschnittliche relative Aufwandsfaktoren für Fehlerkorrekturen bei Ericsson

[218] Vgl. zu diesem Absatz Noth, Kretschmar /Aufwandsschätzung/ 4 und Balzert /Software-Technik/ 74.

[219] Die hier verwendeten Begriffe Kosten und Aufwand unterscheiden sich vom Verständnis in der Betriebswirtschaftslehre (vgl. dazu Wöhe, Döring /Betriebswirtschaftslehre/ 1006-1024).

[220] Vgl. zu diesem Absatz Damm, Lundberg, Wohlin /Improvement potential/ 143 f.

[221] Vgl. hierzu Kapitel 2.2.5.

Die bereits erwähnte NIST-Studie[222] aus dem Jahr 2002 nennt die durchschnittlich erforderliche Anzahl von Personenstunden zum Finden und Beheben eines Fehlers, der in der Anforderungsanalyse eingeführt wurde.[223] Wie in der Tabelle 3-3 dargestellt, betrachtet die Studie Softwareentwicklungen für die Fertigungsindustrie des Transportwesens (Automobil, Luftfahrt und Bahn) und für den Finanzdienstleistungssektor getrennt. Der Aufwand zum Finden und Beheben eines Anforderungsfehlers steigt zum Beispiel gemäß dieser Studie nach Auslieferung der Software um das 5-fache in der Fertigungsindustrie des Transportwesens im Vergleich zum Finden und Beheben dieses Fehlers in der Anforderungsanalyse (einschließlich dem Architekturentwurf). Für den Finanzdienstleistungssektor steigt der Aufwand sogar um das 16-fache.

		Anforderungsanalyse/ Architekturentwurf	Implementierung/ Modultest	Integrations- und Systemtest[224]	Beta-Test	nach Auslieferung
I	**Anzahl Stunden***	2	4	6	8	10
	Aufwandsfaktor**	1x	2x	3x	4x	5x
II	**Anzahl Stunden***	1,2	8,8	14,8	15	18,7
	Aufwandsfaktor**	1x	7x	12x	13x	16x

I = Softwareentwicklungen für die Fertigungsindustrie des Transportwesens
II = Softwareentwicklungen für den Finanzdienstleistungssektor

* durchschnittliche Anzahl von Stunden zum Finden und Beheben eines Fehlers, der in der Anforderungsanalyse (inkl. Architekturentwurf) eingeführt wurde.

** relativer Aufwandsfaktor zum Finden und Beheben eines Fehlers im Vergleich zur Phase Anforderungsanalyse (inkl. Architekturentwurf)

Tabelle 3-3: Aufwand zum Finden und Beheben von Anforderungsfehlern

Daly berichtet 1977 von zwei Entwicklungsvorhaben, der Entwicklung einer Echtzeitsoftware mit einem Umfang von 111 000 Anweisungen und eines Software-Werkzeugs mit einem Umfang von 12 000 Anweisungen.[225] Dem Autor nach steigen die Kosten (und damit der Aufwand) zum Finden und Beheben eines Fehlers nach der Implementierungsphase,

- um das 4-fache, falls der Fehler während der Testphase entdeckt wird und

[222] Siehe hierzu auch S. 48.

[223] Vgl. Tassey /Software Testing/ [6-10] und [7-12].

[224] In Tassey /Software Testing/ wird der Integrationstest in der Tabelle 6-3 auf Seite [6-10] sowie in der Tabelle 7-5 auf Seite [7-12] aus Platzgründen allein aufgeführt. Ein Blick auf die beigefügten Fragebögen auf den Seiten [C-10] f. und [E-10] f. verdeutlicht jedoch, dass nach Integrations- und Systemtest gefragt wurde.

[225] Vgl. Daly /Software development/ 238 und 241.

- um das 15-fache, falls der Fehler während der Wartung, also nach der Auslieferung und Inbetriebnahme, entdeckt wird.

Daly berücksichtigt in seiner Arbeit nicht, dass Fehler zu verschiedenen Zeitpunkten der Softwareentwicklung (Anforderungsanalyse, Entwurf, Implementierung) entstehen können. Insbesondere nennt er Anforderungsfehler nicht explizit. Trotzdem sind diese Aufwandsfaktoren auch für Fehler in der Anforderungsanalyse relevant. Erfordert das Finden und Beheben eines Anforderungsfehlers beispielsweise in der Implementierung 1 Personenstunde, so würde nach Daly die Korrektur dieses Fehlers in der Testphase 4 Personenstunden und während der Wartung sogar 60 Personenstunden beanspruchen. Folglich untermauert auch Dalys Arbeit nachdrücklich die These für die Relevanz des Praxisproblems.

Nach Boehm steigen die Kosten (und damit der Aufwand) zum Finden und Beheben eines Fehlers mit jeder Entwicklungsphase exponentiell an.[226] Die Korrektur eines Fehlers nach Auslieferung der Software sei demnach sogar um das 100-fache aufwändiger als die Korrektur in der Phase der Anforderungsanalyse und des Entwurfs. Diese Aussage Boehms wird vielfach wiedergegeben, seit der Veröffentlichung in einem Artikel im Jahr 1976.[227] Hierbei ist aber zu berücksichtigen, dass sie auf nicht veröffentlichten Projektdaten oder Arbeitspapieren der zwei amerikanischen Unternehmen TRW und IBM basiert. Die Aussage wird in Beiträgen, die nach 2000 erschienen sind, von mehreren Experten (einschließlich Boehm) bekräftigt.[228] Jedoch wird ihre Gültigkeit auf schwerwiegende Fehler in großen Softwareprojekten eingeschränkt. Leider wird dabei nicht konkretisiert, was schwerwiegende Fehler und große Softwareprojekte sind. Eine Untersuchung von Westland aus dem Jahr 2002 stützt empirisch Boehms These des exponentiellen Wachstums der Korrekturkosten (und damit des Korrekturaufwands).[229]

Boehm findet in einer weiteren Untersuchung Anfang der Achtziger heraus, dass die Kosten (und damit der Aufwand) zum Finden und Beheben von Fehlern in kleinen Softwareentwicklungsprojekten von der Anforderungsanalyse bis zum Abnahmetest um das 4-fache steigen.[230] Unter einem kleinen Softwareentwicklungsprojekt versteht er in seiner Untersuchung, dass

---

[226] Vgl. Boehm /Software/ 1227 f.

[227] Boehm nennt nur den Kostenfaktor 100 für Korrekturen nach Auslieferung der Software explizit in seinen Veröffentlichungen. Ansonsten stellt er eine Abbildung zur Verfügung, welche den Trend verdeutlicht, dass die Korrektur eines Fehlers teurer wird, je später sie erfolgt. Vgl. hierzu Boehm /Software/ 1227 f. und Boehm /Economics/ 39 f. Einige Autoren nennen weitere konkrete Zahlen für Kostenfaktoren mit Rückgriff auf diese Abbildung, obwohl diese hierfür nicht geeignet ist. Deshalb wird Boehm manchmal mit unterschiedlichen Kostenfaktoren zitiert.

[228] Vgl. Shull u. a. /Defects/ 251 und Boehm, Basili /Software defect/ 135.

[229] Vgl. Westland /Errors/ 1-9.

[230] Vgl. Boehm /Experiment/ 489.

Projektteams bestehend aus 6 Personen Software in einem Umfang von ungefähr 2000 Codezeilen entwickeln.

Die Tabelle 3-4 gibt eine Übersicht über die bisher diskutierten relativen Faktoren des Aufwands (bzw. der Kosten) zum Finden und Beheben von Softwarefehlern.

	Damm u. a., 2004	**Tassey, 2002**		**Daly, 1977**	**Boehm, 1976 und 1981**	
		I	**II**		schwere Fehler in großen Projekten	Fehler in kleinen Projekten
Anforderungsanalyse	-	1x	1x	-	1x	1x
Architekturentwurf	1x			-		-
Feinentwurf	1,2x	-	-	-		-
Implementierung		2x	7x	1x	-	-
Modultest				4x	-	-
Integrationstest	3x	3x	12x		-	-
Funktionstest	9,8x				-	-
Systemtest	16x				-	-
Beta-Test	20x	4x	13x	-	-	-
Abnahmetest	-	-	-	-	-	4x
nach Auslieferung	-	5x	16x	15x	100x	-

I = Softwareentwicklungen für die Fertigungsindustrie des Transportwesens
II = Softwareentwicklungen für den Finanzdienstleistungssektor

Tabelle 3-4: Relative Faktoren des Aufwands (bzw. der Kosten) zur Korrektur von Fehlern

Wie teuer die Korrektur eines Fehlers konkret ausfällt, hängt von vielen Einflussgrößen ab. So ist es beispielsweise wahrscheinlich, dass eine flexible Architektur Kostenfaktoren reduzieren kann.[231] Ebenso werden das Ausmaß der Codemodularität und der Erfahrungsstand der Entwickler die Korrektur eines Fehlers begünstigen oder erschweren.

Dies ist auch Gegenstand der Kritik der Vertreter agiler Entwicklungsansätze[232] an den empirischen Studien zu relativen Kostenfaktoren, die um die achtziger Jahre im Umfeld großer Softwareprojekte durchgeführt worden sind. Sie kritisieren vor allem Boehms These des exponentiellen Anstiegs der Korrekturkosten. Nach ihrer Ansicht führt die These zu folgenden falschen Schlussfolgerungen: [233]

- Um exponentiell steigende Kosten zu vermeiden, sind Fehler stets zu vermeiden.

[231] Vgl. Boehm, Basili /Software defect/ 135.

[232] Zu den agilen Entwicklungsverfahren zählen u. a. Adaptive Software Development, die Crystal-Methodenfamilie und Extreme Programming. Für einen kurzen Überblick über diese Verfahren siehe Coldewey /Entwicklung/ 240-245. Für eine differenzierte Darstellung der Gestaltungsempfehlung von Extreme Programming in Abgrenzung zu sequentiellen Ansätzen siehe Mellis /Softwareentwicklungsprozess/.

[233] Vgl. hierzu und im Folgenden z B. Coldewey /Entwicklung/ 238-240 und Coldewey /Wandel/ 78-83.

- Um Fehler zu vermeiden, ist insbesondere in der Anforderungsanalyse und dem Entwurf auf Dokumentation und Redundanz zu setzen. Das heißt, dass Zwischenergebnisse in den frühen Phasen der Entwicklung vollständig und genau zu dokumentieren sind (Dokumentation) und mehrere Modelle der Software zu erstellen sind, die unterschiedliche Sichtweisen der gleichen Software abbilden (Redundanz).

Diese Schlussfolgerungen werden in Frage gestellt, da sie implizit auf Annahmen basieren, die nicht immer gültig seien. Diese Kritik ist nicht unberechtigt. So können beispielsweise in einem dynamischen Umfeld, in dem sich Anforderungen zum Teil erst in der Auseinandersetzung mit lauffähigen Zwischenergebnissen der Software ergeben oder sich Kundenbedürfnisse während des Projekts ändern, Fehler oder Änderungen nicht grundsätzlich vermieden werden.[234] Außerdem werden die geschlussfolgerten Entwicklungsstrategien zur Fehlervermeidung kritisiert. Schließlich würden Dokumentation und Redundanz selbst maßgeblich dafür sorgen, dass die Kosten für späte Änderungen und somit auch Fehlerkorrekturen hoch ausfallen.[235] Die Lösung werde also zum Bestandteil des Problems.

Kritiker der Boehmschen These argumentieren letztendlich, dass der These im dynamischen Umfeld eine falsche Kausalvermutung zugrunde liegt. Die Hervorhebung der Fehlervermeidung gekoppelt mit der Dokumentation und Redundanz baue eine selbsterfüllende Prognose auf: trotz umfassender und redundanter Dokumentation träten Fehler auf, deren Korrektur aber gerade aufgrund der Dokumentation und Redundanz zu aufwändigen Überarbeitungen führten. Dies werde fälschlicherweise als Bestätigung der These einschließlich der Schlussfolgerungen verstanden, so dass noch mehr auf Dokumentation und Redundanz gesetzt wird.

Entscheidend ist, dass auch die agilen Ansätze auf der Überlegung aufbauen, dass die Kosten und damit der Aufwand der Korrektur zunehmen, je später ein Fehler gefunden und behoben wird. Während hinsichtlich dieser These also weitgehend Konsens herrscht, gehen die Meinungen lediglich bei der Frage auseinander, in welchem Maße die Kosten und damit der Aufwand über die Zeit steigen. Die These zur Relevanz des Praxisproblems verliert folglich auch aus dieser Perspektive nicht an Gültigkeit: der Aufwand für die Korrektur eines Anforderungsfehlers steigt, je später er gefunden und behoben wird.

---

[234] Vgl. Mellis u. a. /Software development/ 4-12.

[235] Vgl. Coldewey /Entwicklung/ 238-240 und Coldewey /Wandel/ 78-83.

3.4 Vorschläge für fehlerbasierte Verfahren

3.4.1 Merkmale von fehlerbasierten Verfahren

In einer Softwareentwicklung auftretende Fehler können erfasst und ausgewertet werden. Die Art und Häufigkeit von Fehlern erlauben Rückschlüsse auf den Prozess.[236] Insbesondere können bereits entwicklungsbegleitend die Ursachen von Fehlern im Entwicklungsprozess identifiziert werden. Fehlerbasierte Verfahren verfolgen diesen Anspruch und stellen folglich einen Ausgangspunkt für eine systematische Prozessverbesserung da.

Empirische Untersuchungen zu Fehlern beinhalten in der Regel Ausführungen dahingehend, wie Fehler im Rahmen der Untersuchung erfasst und ausgewertet wurden.[237] Entsprechende Ausführungen verfolgen jedoch nicht den Anspruch, einen praktischen Vorschlag für ein fehlerbasiertes Verfahren zur Prozessverbesserung zu entwickeln. Die Zahl solcher Vorschläge ist in der Literatur überschaubar. Hierzu zählen insbesondere die Hewlett-Packard-Fehleranalyse, der IEEE-Standard zu Softwareanomalien und die Orthogonal Defect Classification.[238]

Im Mittelpunkt solcher Vorschläge steht jeweils ein *Klassifikationsschema* zur Beschreibung einzelner Fehler. Klassifikationsschemata bestehen aus einer Menge von Attributen. Ein Attribut ist ein als relevant erachtetes Merkmal eines Fehlers.[239] Wird ein Klassifikationsschema angewandt, also Fehler klassifiziert, werden für jeden einzelnen Fehler die Attribute mit in der Regel vorgegebenen Werten gemessen. Die Attribute sind nominal oder ordinal skaliert.[240] Nominale Attribute haben unterschiedliche Werte, die keine Beziehung untereinander implizieren. Die Werte ordinaler Attribute können dagegen geordnet werden.

Im Folgenden werden der Vorschlag der IEEE zur Klassifikation von Softwareanomalien (Kapitel 3.4.2), die Hewlett-Packard-Fehleranalyse (Kapitel 3.4.3) und die Orthogonal Defect Classification (Kapitel 3.4.4) kurz dargestellt und in Kapitel 3.4.5 verglichen. Die Darstellung der fehlerbasierten Verfahren ist nach folgenden Fragen strukturiert:

- Was ist der Anwendungsbereich des Verfahrens?
- Welche Vorgehensweise wird vorgeschlagen?
- Wie ist das Klassifikationsschema gestaltet?
- Wie sollen klassifizierte Fehler ausgewertet werden?

---

[236] Vgl. hierzu zum Beispiel Bhandari u. a. /Case study/, Bhandari u. a. /Improvement/, Leszak, Perry, Stoll /Case study/ und Mays u. a. /Defect prevention/.

[237] Vgl. hierzu die in Kapitel 3.2 aufgeführten empirischen Studien.

[238] Vgl. zur Auswahl Freimut /Defect classification schemes/ 6-12.

[239] Vgl. Fenton, Pfleeger /Software metrics/ 5.

[240] Vgl. Fenton, Pfleeger /Software metrics/ 47-49.

Zusätzlich wird jeder Vorschlag kurz dahingehend kritisch gewürdigt

- inwiefern er praktisch anwendbar ist,
- inwiefern er theoretisch fundiert ist und
- inwiefern empirische Erkenntnisse vorliegen.

3.4.2 IEEE Standard Classification for Software Anomalies

Anwendungsbereich

Der IEEE Standard 1044-1993 mit der Bezeichnung „IEEE Standard Classification for Software Anomalies“ beinhaltet einen Vorschlag für ein fehlerbasiertes Verfahren.[241] Es definiert ein Klassifikationsschema, das sich nicht nur darauf beschränkt Fehler zu beschreiben. Vielmehr wird der Anspruch erhoben, dass das Klassifikationsschema auf jeden Zustand im Softwarelebenszyklus anwendbar ist, der von bestimmten Erwartungen abweicht. Diese Erwartungen können auf dokumentierten Zwischenergebnissen der Softwareentwicklung wie einem Anforderungsdokument, Entwurfsdokument aber auch Wahrnehmungen und Erfahrungen beruhen. Entsprechende Abweichungen werden als *Softwareanomalien* bezeichnet. Der Begriff der Anomalie ist in diesem Zusammenhang ungewöhnlich. Dessen Wahl wird von den Autoren des Standards dadurch begründet, dass er im Vergleich zu Begriffen wie z. B. Fehler, Defekt und Problem weniger inhaltlich „vorbelastet“ sei.

Vorgehensweise

Der Standard 1044-1993 unterscheidet vier grundlegende, sequenzielle Arbeitsschritte im Lebenszyklus einer Anomalie:[242]

- ihre Erkennung (recognition),
- ihre Untersuchung (investigation),
- Gegenmaßnahmen (action) und
- den Abschluss der Anomalie (disposition).

Alle vier Arbeitsschritte haben die Gemeinsamkeit, dass sie jeweils in drei Verwaltungsschritten unterteilt werden können. Zunächst werden Information gesammelt, die eine Klassifizierung unterstützen oder sogar erst ermöglichen (recording). Hierfür wird für jeden Arbeitsschritt vorgeschlagen, welche Informationen relevant sein können. Darauf aufbauend erfolgt dann die für den jeweiligen Arbeitsschritt relevante Klassifizierung gemäß dem Klassifi-

---

[241] Vgl. zu diesem Absatz IEEE /Software Anomalies/ 1-3. Zu dem IEEE-Standard 1044-1993 wurde zusätzlich eine Anleitung veröffentlicht, welche einen Nutzer des Standards dabei unterstützt, nach eigenen Bedürfnissen anzupassen. Vgl. hierzu IEEE /Guide/.

[242] Vgl. zu diesem Absatz IEEE /Software Anomalies/ 4-6.

kationsschema (classifying). Schließlich werden in jedem Arbeitsschritt die Auswirkungen der Anomalie festgehalten (identifying impact). Die Abbildung 3-2 fasst die Vorgehensweise des IEEE-Standards zusammen.

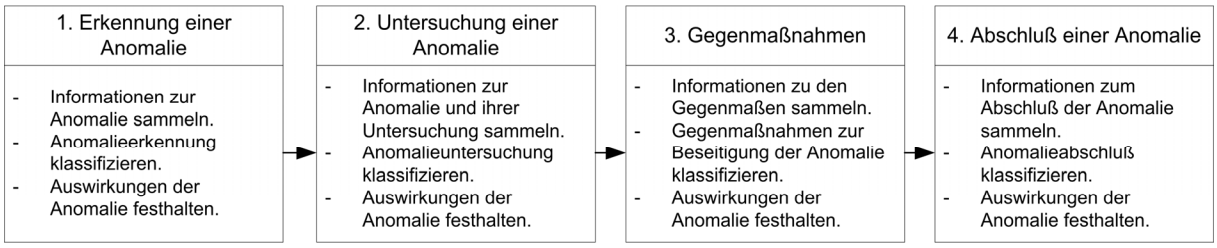


Abbildung 3-2: Vorgehensweise beim IEEE-Standard 1044-1993

Klassifikationsschema

Im Folgenden wird ein Überblick über die Attribute des Klassifikationsschemas gegeben und jeweils mögliche Werte als Beispiel aufgeführt. Im IEEE-Standard sind die Attribute mit ihren Werten tabellarisch aufgelistet. Die Werte der Attribute werden im Standard nur genannt, jedoch nicht beschrieben. Gleiches gilt für die Attribute, sofern von einleitenden Anmerkungen abgesehen wird. Die Autoren des Standards verweisen auf die Begriffsdefinitionen im IEEE-Standard 610.12-1990[243].[244] Abweichend vom Standard 1044-1993 werden die Inhalte der Attribute nachfolgend in Form von einfachen Fragen wiedergegeben, um ihr Verständnis zu erleichtern. Die Attribute und ihre Werte werden sinngemäß und nicht wortwörtlich ins Deutsche übersetzt. Bei Attributen wird zusätzliche ihre englische Bezeichnung in Klammern aufgeführt.

Das Klassifikationsschema ist gegliedert nach den oben beschriebenen vier Arbeitsschritten. Die möglichen Attributwerte sind hierarchisch geordnet. So kann beispielsweise für das Attribut Projektphase der Wert „Anforderungen“ gewählt werden oder mögliche Verfeinerungen dieses Wertes wie z. B. Konzeptevaluation, Systemanforderungen oder Softwareanforderungen.

Im ersten Arbeitschritt, der Erkennung einer Anomalie, sind folgende Attribute bei einer gegebenen Anomalie zu klassifizieren:[245]

- Projektaktivität (project activity): Bei welcher Projektaktivität wurde die Anomalie aufgedeckt? Mögliche Werte sind zum Beispiel „Testen“ oder „Inspektion“.

[243] Vgl. IEEE /Glossary/.

[244] Vgl. IEEE /Software Anomalies/ 2 f.

[245] Vgl. zum Folgenden IEEE /Software Anomalies/ 7-10.

- Projektphase (project phase): In welcher Projektphase ist die Anomalie aufgetreten? Mögliche Werte sind z. B. „Anforderungen“, „Entwurf“ oder „Betrieb und Wartung“.
- Vermutete Ursache (suspected cause): Welche Ursache wird hinter der Anomalie vermutet? Mögliche Werte sind z. B. „Testsystem“, „Plattform“ oder „Benutzer“.
- Wiederholbarkeit (repeatability): Ist die Anomalie wiederholbar? Mögliche Werte sind z. B. „einmaliges Auftreten“ oder „reproduzierbar“.
- Symptom (symptom): Was sind die Symptome der Anomalie? Mögliche Werte sind z. B. „Eingabeproblem“ oder „Programmabsturz“.
- Produktstatus (product status): Inwiefern beeinträchtigt die Anomalie die Nutzung der betroffenen Software? Mögliche Werte sind z. B. „unbrauchbar“ oder „eingeschränkt nutzbar“.

Abgesehen von den Attributen Projektaktivität und Projektphase sind die übrigen Attribute optional.

Nachdem eine Anomalie erkannt und klassifiziert worden ist, folgt ihre Untersuchung. Das Ergebnis der Untersuchung soll gemäß dem IEEE-Standard anhand folgender Attribute klassifiziert werden:[246]

- Tatsächliche Ursache (actual cause): Was ist die Ursache der Anomalie? Mögliche Werte sind z. B. „Testsystem“, „Plattform“ oder „Benutzer“.
- Arbeitsergebnis (source): Welches Arbeitsergebnis ist von der Anomalie betroffen? Mögliche Werte sind z. B. Code oder Datenbank.
- Typ (type): Um was für einen Typ von Anomalie handelt es sich? Mögliche Werte sind z. B. logisches Problem oder Datenproblem.

Alle drei Attribute müssen zwingend für eine gegebene Anomalie mit einem Wert belegt werden. Das Attribut Typ hat im Vergleich zu den allen übrigen des Klassifikationsschemas eine Vielzahl an möglichen Werten, die im Standard auf über zwei Seiten dargestellt werden.
Mit dem Ende der Untersuchung einer Anomalie sind ihre Ursachen bekannt. Im nächsten Arbeitsschritt können Gegenmaßnahmen zur Beseitigung der Anomalie vorbereitet werden. Im IEEE-Standard sind diesbezüglich folgende Attribute relevant:[247]

- Lösung (resolution): Wann soll die Anomalie gelöst werden? Mögliche Werte sind z. B. „umgehend“ oder „verschieben“.

---

[246] Vgl. zum Folgenden IEEE /Software Anomalies/ 11-16.
[247] Vgl. zum Folgenden IEEE /Software Anomalies/ 16-18.

- korrigierende Maßnahme (corrective action): Welche Maßnahmen sind erforderlich, um die Anomalie zu beseitigen? Mögliche Werte sind z. B. „Maßnahmen auf Abteilungsebene“ oder „Maßnahmen auf Unternehmensebene“.

Das erste Attribut muss belegt werden, das zweite ist optional.

Nach der Festlegung der Gegenmaßnahmen kann die Anomalie abgeschlossen werden. Hierzu muss ein Attribut berücksichtigt werden, das den gleichen Namen hat wie der zugehörige Arbeitsschritt:

- Abschluss (disposition): Wie wird die Anomalie abgeschlossen? Mögliche Werte sind z. B. „beendet“ oder „mit einem anderen Problem zusammengeführt“.

In jedem der vier Arbeitsschritte sind die Auswirkungen der jeweils betrachteten Anomalie festzuhalten bzw. zu aktualisieren. Hierzu schlägt der IEEE-Standard ebenfalls eine Reihe von Attributen vor:

- Grad der Schwere (severity): Wie schwerwiegend ist die Anomalie?
- Priorität (priority): Welche Priorität hat die Beseitigung der Anomalie?
- Kundennutzen (customer value): Welchen Nutzen hat der Kunden, wenn die Anomalie beseitigt wird?
- Sicherheit der Mission (mission safety): Welche Auswirkung hat die Anomalie auf die Sicherheit der Mission?
- Projektzeitplan (project schedule): Welche Auswirkung hat die Anomalie auf den Projektzeitplan?
- Projektkosten (project cost): Welche Auswirkung hat die Anomalie auf die Projektkosten?
- Projektrisiko (project risk): Welche Auswirkung hat die Anomalie auf das Projektrisiko?
- Projektqualität und Zuverlässigkeit (project quality / reliability): Welche Auswirkung hat die Anomalie auf die Projektqualität und die Zuverlässigkeit?
- Gesellschaftliche Auswirkungen (societal): Welche gesellschaftlichen Auswirkung hat die Anomalie?

Alle Attribute sind ordinal und haben jeweils bis zu maximal sechs Werte, die in der Regel das Spektrum von „hohe Auswirkung“ bis „keine Auswirkung“ abdecken. Abgesehen von den Grad der Schwere, Projektzeitplan und Projektkosten sind die übrigen Attribute als optional angegeben.

Auswertung klassifizierter Fehler

Der IEEE-Standard 1044-1993 beinhaltet keine Ausführungen darüber, wie klassifizierte Fehler ausgewertet werden können oder sollen.

Kritische Würdigung des Vorschlags

Die praktische Anwendbarkeit des IEEE-Standards wird dadurch eingeschränkt, dass die Attribute und ihre Wertemenge nicht definiert und beschrieben werden. Zugleich werden keinerlei Angaben darüber, wie eine Auswertung erfolgen soll.

Das Klassifikationsschema versucht, den gesamten Softwarelebenszyklus und jegliche Abweichungen von irgendwelchen Erwartungen zu berücksichtigen. Dies hat jedoch zur Folge, dass ganz verschiedene Sachverhalte als Anomalien erfasst werden: z. B. die Fehlbedienung eines Benutzers, ein Implementierungsfehler oder falsche Testdaten. Spezifische Merkmale von Fehlern, die entscheidungsrelevant sein können, werden daher nahezu nicht berücksichtigt. Im Wesentlichen ermöglicht nur das Attribut Typ (type), einen Fehler genauer zu charakterisieren. Es stellt sich deshalb die Frage, ob eine ausschließliche Auswertung der Fehler überhaupt Einblicke über mögliche Ursachen im Prozess erlauben kann.

Der Vorschlag wird theoretisch nicht fundiert. Dem Autor sind keine Veröffentlichungen bekannt, die empirische Befunde oder Erfahrungsberichte zu diesem Standard beinhalten.

3.4.3 Hewlett-Packard-Fehleranalyse

Mitte der achtziger Jahre wird bei Hewlett-Packard (HP) der Bedarf für ein fehlerbasiertes Verfahren gesehen.[248] Zeitgleich ist ein IEEE-Standard für die Fehlerklassifikation in Arbeit, der später zum IEEE Standard 1044-1993 wird. Arbeitsergebnisse des damals noch nicht abgeschlossenen Standards werden bei der Entwicklung eines eigenen Verfahrens berücksichtigt. Robert B. Grady beschreibt das Verfahren und die aus der Anwendung bei HP gewonnenen Erfahrungen in mehreren Veröffentlichungen.[249] Das Verfahren wird nachfolgend als Hewlett-Packard-Fehleranalyse oder kurz HP-Fehleranalyse bezeichnet.

Anwendungsbereich

Nach Grady ist der Zweck der HP-Fehleranalyse, dass Fehler einheitlich dokumentiert und analysiert werden.[250] Dies soll in Anstrengungen münden, Fehler und ihre Hauptursachen zu

---

[248] Vgl. zu diesem Absatz Grady /Software metrics/ 127.

[249] Vgl. inbesondere Grady /Software metrics/127-131, 137-157, 223-227, Grady /Software failure analysis/ 155-162d und Grady /Software quality/ 65-67.

[250] Vgl. zu diesem Absatz Grady /Software metrics/ 223.

eliminieren. Im Fokus stehen Fehler, die im gesamten Softwarelebenszyklus auftreten können. Dabei versteht Grady unter einem Fehler[251], einen Mangel in Ergebnissen der Anforderungsanalyse, des Entwurfs oder der Implementierung einer Software. Einen Fehler grenzt er von der Erweiterung oder Verbesserung einer Software ab, da sie im Gegensatz zu Fehlern beide auf neue oder geänderte Kundenbedürfnisse zurückzuführen sind.

Vorgehensweise

Grady unterscheidet drei Arbeitsschritte bei der HP-Fehleranalyse.[252] Zunächst sind anhand eines vorgeschlagenen Klassifikationsschemas Fehlerdaten zu sammeln. Die klassifizierten Fehler werden dann ausgewertet. Beide Arbeitsschritte bereiten die Ursachenanalyse vor.
Er stellt drei unterschiedliche Vorgehensweisen vor, die schrittweise einen Übergang in einen kontinuierlichen Prozessverbesserungszyklus ermöglichen sollen. Alle drei Varianten setzen eine Fehlerverfolgung[253] voraus und umfassen jeweils alle drei beschriebenen Arbeitsschritte.

Variante 1: einmalige Ursachenanalyse (one-shot root-cause analysis)[254]

Die erste Variante richtet sich an Organisationen, die zwar eine Fehlerverfolgung bereits durchführen, jedoch bisher keine Erfahrungen mit dem Klassifikationsschema der HP-Fehleranalyse haben.
Grady empfiehlt für diesen Fall, Berichte von 50 bis 75 Fehlern aus der Fehlerverfolgungssoftware zufällig auszuwählen und nachträglich nur den Fehlertyp jedes Fehlers zu bestimmen. Der Fehlertyp beschreibt, was genau fehlerhaft ist. Mögliche Werte sind z. B. Benutzeroberfläche, Logik oder Softwareschnittstelle. Anschließend kann die relative und die absolute Häufigkeit jedes Fehlertyps berechnet werden.
Aufbauend auf diesem Ergebnis sind zwei Fehlertypen auszuwählen, die näher analysiert werden sollen. Für jeweils diese beiden Fehlertypen sind die Ursachen der entsprechenden Fehler zu ermitteln. Dieser Schritt soll methodisch unterstützt werden, indem Ursache-Wirkungsdiagramme nach Ishikawa[255] erstellt werden. Mit der Kenntnis der gefundenen Ursachen können Maßnahmen erarbeitet werden, um den Entwicklungsprozess zu verbessern.

---

[251] Grady bezeichnet einen Fehler mit dem Begriff „defect“. Die fehlerbasierte Vorgehensweise zur Prozessverbesserung nennt er aber dazu inkonsistent „software failure analysis“. Vgl. z. B. Grady /Software metrics/ 137, 223.

[252] Vgl. zu diesem Absatz Grady /Software metrics/ 124-129 und Grady /Software failure analysis/ 158-162c.

[253] Vgl. hierzu Kapitel 2.2.5.

[254] Vgl. Grady /Software failure analysis/ 158 f.

[255] Vgl. Ishikawa /Quality Control/ 18-29.

Variante 2: umfassende Ursachenanalyse (post-project root-cause analysis)[256]

Sofern Organisationen bereits die HP-Fehleranalyse einsetzen und auf der Grundlage des Klassifikationsschemas gefundene Fehler dokumentieren, können sie eine umfassende Ursachenanalyse durchführen.

Im ersten Arbeitsschritt werden die betriebswirtschaftlichen Ziele der Organisation geklärt. Sie ermöglichen im weiteren Verlauf Prioritäten zu setzen, indem sie Anhaltspunkte dafür geben, welchen Fehlern eine besondere Aufmerksamkeit zuteil kommen muss.

Nach Abschluss eines Projektes werden im zweiten Schritt klassifizierte Fehler ausgewertet. Zum einen ist hierzu ist die relative Häufigkeit für jede Fehlerquellen zu ermitteln. Die Fehlerquelle gibt für einen Fehler an, welches Arbeitsergebnis jeweils von diesem Fehler betroffen ist (z. B. Ergebnis der Anforderungsanalyse, Entwurfergebnis, Code etc.). Zum anderen werden die relativen Häufigkeiten der Fehlertypen bestimmt.

Die relativen Häufigkeiten der Fehlerquellen können schließlich mit relativen Aufwandsfaktoren gewichtet werden. Diese relativen Aufwandsfaktoren können aus dem eigenen Projekt stammen oder empirischen Befunden wie in Kapitel 3.3 zugrunde liegen. Grady führt in diesem Zusammenhang folgendes Beispiel auf:[257]

> Klassifizierte Fehler aus dem Systemtest eines Projektes zeigen unter anderem, dass 25,5 % der Fehler als Fehlerquelle Ergebnisse der Anforderungsanalyse aufzeigen. Der Gewichtungsfaktor von 14,5 für entsprechende Fehler besagt, dass die Korrektur dieser Fehler im Systemtest um das 14,5-fache aufwändiger ist als in der Anforderungsanalyse. Grady multipliziert die relativen Häufigkeiten der Fehlerquellen jeweils mit den entsprechenden relativen Aufwandsfaktoren und normalisiert das Ergebnis wieder auf 100%. Fehler in den Ergebnissen der Anforderungsanalyse haben dann sogar eine gewichtete relative Häufigkeit von 52,9%.

Ausgehend von den betriebswirtschaftlichen Zielen der Organisation werden im nächsten Schritt zwei Fehlertypen ausgewählt, deren Fehler unter der Fragestellung näher untersucht, wie sie zukünftig vermieden oder früher gefunden werden können. Für eine Auswahl von Fehlern werden die Ursachen ermittelt und Lösungen erarbeitet. Dieser Schritt soll methodisch unterstützt werden, indem Ursache-Wirkungsdiagramme nach Ishikawa[258] erstellt werden.

Im letzten Schritt werden die geplanten Änderungen im Entwicklungsprozess festgehalten und bei Bedarf die Vorgehensweise der Ursachenanalyse überarbeitet.

---

[256] Vgl. Grady /Software failure analysis/ 159-162a und Grady /Software Metrics/126-131

[257] Vgl. Grady /Software Metrics/ 126-131.

[258] Vgl. Ishikawa /Quality Control/ 18-29.

Variante 3: kontinuierlicher Prozessverbesserungszyklus (continous process improvement cycle) [259]

Der kontinuierliche Prozessverbesserungszyklus unterscheidet sich von der umfassenden Ursachenanalyse dadurch, dass die HP-Fehleranalyse in die Entwicklungsprozesse integriert und organisatorisch institutionalisiert ist. Das Klassifikationsschema ist Bestandteil der Fehlerverfolgung. In regelmäßigen Zeitabständen oder zur Einführung einer neuen Version einer Software, werden klassifizierte Fehler ausgewertet und ihre Ursachen analysiert. Kleine Teams sollen dieses Verfahren betreuen und kontinuierlich die Entwicklungsprozesse durch viele kleine, aber nachhaltige Maßnahmen verbessern.

Klassifikationsschema

Grady fordert, dass nicht ausschließlich Fehlerdaten gesammelt werden, um Fehler schneller lokalisieren und beheben.[260] Vielmehr sollen zusätzlich solche Daten berücksichtigt werden, die es erlauben, aus Fehlern zu lernen und den Entwicklungsprozess zu verbessern. Dieser Forderung begegnet er mit einem Klassifikationsschema, das aus nur drei Attributen mit zugehörigen Werten besteht. Das Klassifikationsschema erhebt nicht den Anspruch alle Merkmale eines Fehlers aufzuführen, die im Rahmen einer Fehlerverfolgung erfasst werden sollen. Es ist vielmehr ein Vorschlag wie eine Fehlerverfolgung *ergänzt* werden kann, um seiner Forderung gerecht zu werden.

Die drei Attribute haben jeweils folgende Bedeutung:[261]

- Fehlerquelle (defect origin): Welches Arbeitsergebnis ist der Ausgangspunkt des Fehlers? Mögliche Werte sind z. B. Ergebnisse der Anforderungsanalyse und des Entwurfs.
- Fehlertyp (defect type): Was genau ist im Arbeitsergebnis fehlerhaft? Mögliche Werte sind z. B. Funktionalität oder Logik.
- Fehlermodus (defect mode): Warum handelt es sich um einen Fehler? Mögliche Werte sind z. B. fehlend oder falsch.

Die drei Attribute sind sequentiell mit Werten zu belegen, da das Attribut Fehlertyp abhängig ist vom Attribut Fehlerquelle. Indem ein Wert für die Fehlerquelle festgelegt wird, kann der Fehlertyp nur noch mit Werten belegt werden, die für diese Fehlerquelle sinnvoll sind (vgl. Abbildung 3-3). Für die Fehlerquelle Dokumentation sieht das Klassifikationsschema keine Werte im Attribut Fehlertyp vor.

---

[259] Vgl. Grady /Software failure analysis/

[260] Vgl. zu diesem Absatz Grady /Software Metrics/ 124-126.

[261] Vgl. zum Folgenden Grady /Software Metrics/ 223-227.

Die Attribute Fehlerquelle und Fehlertyp weisen eine Ähnlichkeit mit den Attributen Arbeitsergebnis (source) und Typ (type) des IEEE-Standards 1044-1993 auf.[262] Das HP-Klassifikationsschema hat gegenüber dem IEEE-Standard den Vorteil, dass trotz der großen Menge an auswählbaren Werten beim Attribut Fehlertyp ein einfacher Überblick möglich ist, da die möglichen Werte nach Fehlerquellen gruppiert sind (vgl. Abbildung 3-3).

Grady definiert alle Attribute einschließlich ihrer Werte. Jedes Attribut hat zusätzlich den Wert „andere“. Dies sorgt für den Fall vor, dass das Klassifikationsschema bei bestimmten Fehlern nicht anwendbar ist, weil die Attributwerte den realen Fehler nicht abbilden. So wird zum einen vermieden, dass ein Attribut mit einem unzutreffenden Wert belegt wird. Wenn der Wert „andere“ häufig auftritt, kann das zum anderen ein Hinweis darauf sein, dass das Klassifikationsschema angepasst werden muss.

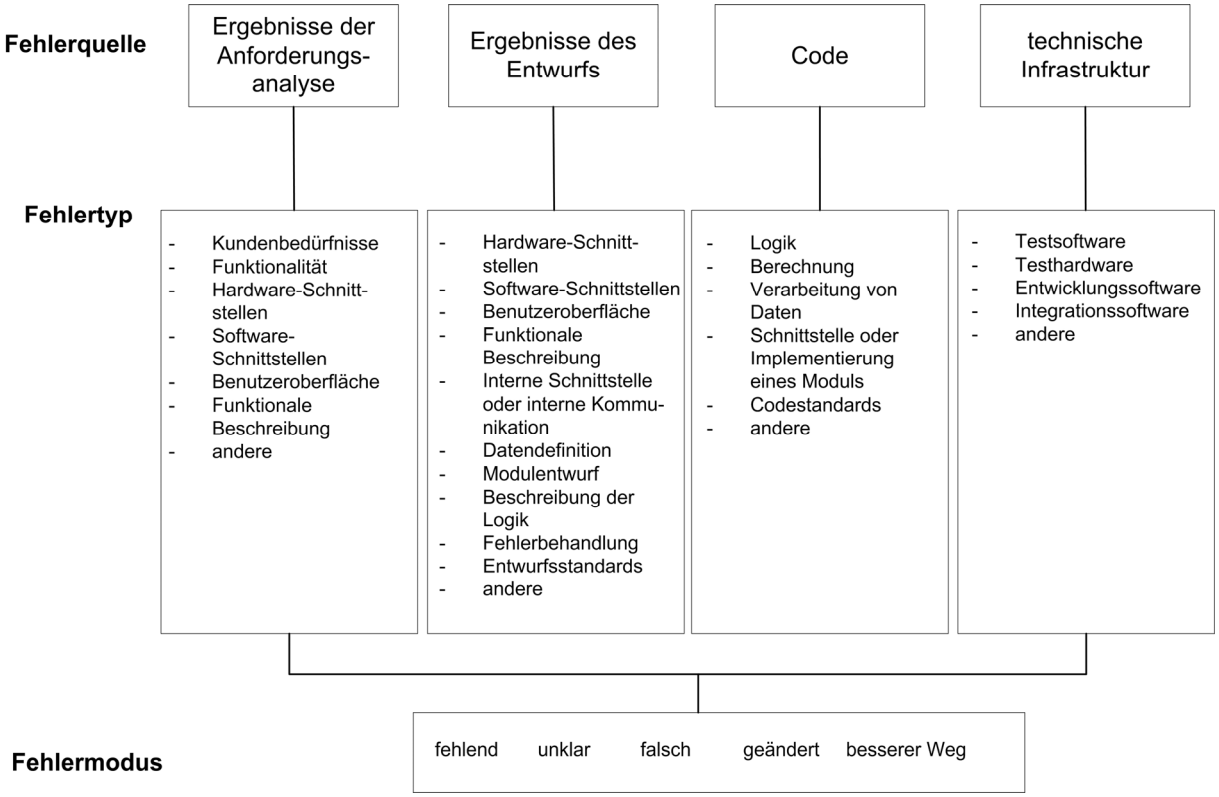


Abbildung 3-3: Hewlett-Packard -Klassifikationsschema[263]

Auswertung klassifizierter Fehler

Das HP-Klassifikationsschema hat mit drei Attributen einen schnell überschaubaren Umfang. Infolgedessen sind auch die Möglichkeiten begrenzt, klassifizierte Fehler quantitativ auszu-

[262] Vgl. Kapitel 3.4.2.

[263] In Anlehnung an Grady /Software failure analysis/ 157. Die Werte „Dokumentation“ und „andere“ für das Attribut Fehlerquelle wurden nicht in die Abbildung aufgenommen, da das Klassifikationsschema für entsprechende Fehler keine Werte im Attribut Fehlertyp vorsieht.

werten. Im Wesentlichen werden absolute und relative Häufigkeiten der Werte eines Attributs berechnet. Grady legt mit mehreren Beispielen aus dem Unternehmen HP ausführlich dar, wie entsprechende Ergebnisse aussehen und interpretiert werden können.[264]
Grady führt nicht aus, wie das Attribut Fehlermodus ausgewertet werden soll. Er empfiehlt lediglich, Ursachen für die Fehler eines bestimmten Fehlertyps gruppiert nach den Werten des Fehlermodus zu ermitteln und darzustellen.[265] Die Fehler werden folglich danach gruppiert, ob z. B. etwas fehlt, unklar oder falsch ist.

Kritische Würdigung des Vorschlags

Grady beschreibt sehr präzise die Vorgehensweise der HP-Fehleranalyse, was die praktische Anwendbarkeit unterstützt. Das Klassifikationsschema ist kompakt und konzentriert sich nur auf Fehler in der Softwareentwicklung. Obwohl jeder Attributwert einzeln definiert wird, fällt es bisweilen schwer, bestimmte Werte klar voneinander abzugrenzen. Beispielsweise bleibt eine genaue Abgrenzung der Werte „requirements or specifications“, „functionality“ und „functional description“ für Fehler in Ergebnissen der Anforderungsanalyse aus.
Implizit legt Grady nahe, dass das Klassifikationsschema nur auf originäre Fehler angewandt werden soll. Folgefehler werden nicht erfasst, da ein originärer Fehler mit seinen Folgefehlern als ein Fehler aufgefasst wird.[266] Bereits bei der Erfassung von Fehlern muss also eine einzelne Person implizit eine Ursachenanalyse durchführen. Dies setzt jedoch voraus, dass diese Person alle betroffenen Arbeitsergebnisse gut kennen muss. Ansonsten besteht die Gefahr, dass die erfassten Daten nicht korrekt sind. Grady stellt hierzu selbst fest:

> *„[...] two different HP divisions reported that many defects that were labeled as 'coding defects' were really caused by changed requirements or design. The incorrect labeling occurred because the defects were discovered or fixed during the coding phase.“*[267]

Die HP-Fehleranalyse sieht für die Ermittlung der Ursachen vor, dass Fehler jeweils einzeln untersucht werden. Aufgrund dessen wird immer nur eine Auswahl von Fehlern in die Ursachenanalyse einbezogen wird, um den Aufwand zu begrenzen. Welche Fehler ausgewählt werden, beeinflusst folglich maßgeblichen welche Ursachen entdeckt werden können. Da die Anzahl der Attribute im Klassifikationsschema beschränkt ist, können Muster in der Fehlermenge unentdeckt bleiben und bei der Fehlerauswahl unberücksichtigt bleiben.

---

[264] Vgl. Grady /Software metrics/ 137-156.
[265] Vgl. Grady /Software metrics/ 147.
[266] Vgl. Grady /Software metrics/ 126.
[267] Grady /Software metrics/ 126.

Der Vorschlag wird theoretisch nur rudimentär fundiert. Abgesehen von den veröffentlichten Erfahrungen bei HP sind dem Autor keine empirische Befunde oder Erfahrungsberichte zur HP-Fehleranalyse bekannt.

3.4.4 Orthogonal Defect Classification

Ram Chillarege entwickelt Anfang der neunziger Jahre bei IBM einen Vorschlag für ein fehlerbasiertes Verfahren: die Orthogonal Defect Classification, kurz ODC.[268] Seit dieser Zeit wird das Verfahren stetig weiterentwickelt.[269] In mehreren Fallstudien wird über die erfolgreiche Anwendung von ODC in verschiedenen Unternehmen berichtet.

Anwendungsbereich

ODC erhebt den Anspruch ein Klassifikationsschema anzubieten, anhand dessen Entwurfs- und Implementierungsfehler erfasst und ausgewertet werden können, um die Ursachen für diese Fehler zu bestimmen.[270] Es werden sowohl Fehler berücksichtigt, die während der Softwareentwicklung entdeckt werden, als auch Fehler, die nach Auslieferung der Software entdeckt werden. Das Klassifikationsschema ist auf Anforderungsfehler nicht anwendbar.

Vorgehensweise

In den Publikationen von Chillarege und seinen Kollegen werden implizit folgende drei Arbeitsschritte unterschieden:[271]

- Erfassung von Fehlern anhand des ODC-Klassifikationsschemas.
- Auswertung der klassifizierten Fehler mit dem Ziel, Fehlermuster zu identifizieren. Fehlermuster sind Teilmengen von Fehlern, die hinsichtlich ausgewählter Attribute die gleichen Ausprägungen haben.
- Bestimmung der Ursachen im Softwareentwicklungsprozess, die dazu geführt haben, dass die Fehler eines Fehlermusters entstanden oder spät entdeckt worden sind.

Klassifikationsschema

Das ODC-Klassifikationsschema gruppiert die zu erfassenden Attribute für Fehler nach zwei grundlegenden Zeitpunkten:[272]

---

[268] Vgl. Chillarege /Defect/ 359-400. Ausgangspunkt von ODC war die empirische Untersuchung Chillarege, Kao, Condit /Defect type/ 246-255.

[269] Vgl. hierzu und zum folgenden Satz die systematische Literaturanalyse zu ODC in Kapitel 5.2.

[270] Vgl. zu diesem Absatz Chillarege u. a. /In-process measurements/ 944 f.

[271] Vgl. z. B. Bhandari u. a. /Case study/ 1159-1163. Für eine detaillierte Darstellung aller Verfahrensschritte siehe 5.1.3.

[272] Vgl. zu Folgendem IBM Research /ODC/ o. S.

1. Attribute, deren Ausprägungen frühestens bestimmt werden können, nachdem ein Fehler *entdeckt* worden ist.
2. Attribute, deren Ausprägungen frühestens bestimmt werden können, nachdem ein Fehler *behoben* worden ist.

Die ODC-Attribute, die nach einer Fehlerentdeckung ermittelt werden, haben jeweils folgende Bedeutung:[273]

- Aktivität (activity): In welcher Aktivität wurde der Fehler gefunden? Der Wertebereich dieses Attributs ist projektspezifisch. Mögliche Werte in einem Projekt können z. B. sein: Codeinspektion oder Modultest.
- Auslöser (trigger): Welche Bedingungen beziehungsweise Kriterien haben dazugeführt, dass der Fehler entdeckt wurde? Mögliche Werte sind z. B. Abdeckung von einzelnen Funktionen oder Kontroll- und Datenfluss.
- Auswirkung (impact): Welches Fehlverhalten verursacht bzw. kann der Fehler verursachen? Mögliche Werte sind z. B. Einschränkungen hinsichtlich des Zeitverhaltens oder der Wartbarkeit der Software.

Die ODC-Attribute, die nach einer Fehlerbehebung ermittelt werden, haben jeweils folgende Bedeutung:[274]

- Ziel (target): Welche Art von Arbeitsergebnissen enthielt den Fehler? Mögliche Werte sind z. B. Entwurfsdokument oder Code.
- Fehlertyp (defect type): Was musste geändert werden, um den Fehler zu beheben? Mögliche Werte sind z. B. Zuweisung/Initialisierung oder interne Schnittstelle.
- Kennzeichner (qualifier): Was zeichnet den Fehler aus? Mögliche Werte sind z. B. fehlende, falsche oder überflüssige Implementierung.
- Ursprung (source): Woher stammt der fehlerbehaftete Teil des Arbeitsergebnisses ursprünglich? Mögliche Werte sind z. B. Eigenentwicklung oder Wiederverwendung aus einer Standard-Bibliothek.
- Alter (age): Welche Vorgeschichte weist der Fehler aus? Mögliche Werte sind z. B. alter Fehler oder neuer Fehler.

[273] Vgl. zu Folgendem IBM Research /ODC/ o. S.
[274] Vgl. zu Folgendem IBM Research /ODC/ o. S.

Auswertung klassifizierter Fehler

Die mit ODC klassifizierten Fehler werden mit dem Ziel ausgewertet, um Fehlermuster aufzuzeigen.[275] Fehlermuster sind dabei Teilmengen von Fehlern, die hinsichtlich ausgewählter Attribute die gleichen Ausprägungen haben.

Bhandari u. a. stellen mit „Attribute Focusing“ eine Methode vor, um die Suche nach Fehlermustern zu automatisieren.[276] Diese Methode zielt darauf ab, schwer erkennbare Zusammenhänge in großen Datenbeständen aufzuspüren und ist daher eine Data-Mining-Methode.[277]

In den Veröffentlichungen von Chillarege und seinen Kollegen werden ansonsten allgemeine Hinweise zur Suche nach Fehlermustern gegeben und beispielhafte Fehlermuster aus Fallstudien dargestellt.[278] Die Beschreibung einer methodischen Vorgehensweise bleibt aus. Zu den wichtigsten Hinweisen zählen folgende Aussagen:[279]

- Fehlermuster zeigen sich insbesondere bei mehrdimensionalen Auswertungen mit zwei oder mehreren Attributen.
- Ein- oder mehrdimensionale Auswertungen, in denen das Attribut Auslöser (trigger) einbezogen wird, erlauben insbesondere Rückschlüsse über die Qualitätssicherungsmaßnahmen.
- Ein- oder mehrdimensionale Auswertungen, in denen das Attribut Fehlertyp (defect type) einbezogen wird, erlauben insbesondere Rückschlüsse über die durchgeführten Entwicklungsaufgaben und die Qualität einer Software.

Kritische Würdigung des Vorschlags

Das ODC-Klassifikationsschema ist ein etabliertes Klassifikationsschemata für Fehler. Es wurde insbesondere auf der Grundlage empirische und praktischer Erfahrungen ständig weiterentwickelt.[280] Eine systematische Literaturanalyse liefert 27 Literaturquellen, die empirische Untersuchungen zu ODC zum Gegenstand haben. Tenor dieser Literaturquellen ist nahezu übereinstimmend, dass ODC in der Praxis erfolgreich anwendbar ist.

ODC ist jedoch nur rudimentär theoretisch fundiert. Es geht von mehreren impliziten und expliziten Annahmen aus, deren systematische Bewertung noch aussteht.

---

[275] Vgl. Chillarege u. a. /In-process measurements/ 944 f.

[276] Vgl. Bhandari u. a. /Improvement/ 186-190. Für eine allgemeine Darstellung dieser Methode siehe Bhandari /Attribute Focusing/.

[277] Vgl. Mertens, Wieczorrek /Strategien/ 18 f.

[278] Vgl. hierzu die in Kapitel 5.2.2.1 aufgeführten empirischen Untersuchungen zu ODC.

[279] Vgl Chillarege u. a. /In-process measurements/ 944 f., Bhandari u. a. /Improvement/ 184 und Chillarege /Defect/ 393-396, 374-376, 384-393.

[280] Vgl. hierzu und im Folgenden Kapitel 5.2.

3.4.5 Vergleich der Vorschläge

Das Kapitel 3.4 hat einen Überblick über drei vorgeschlagene fehlerbasierte Verfahren zur Prozessverbesserung gegeben. Diese waren im Einzelnen:

- IEEE-Standard 1044-1993 zur Klassifikation von Softwareanomalien (Kapitel 3.4.2),
- die Hewlett-Packard-Fehleranalyse (Kapitel 3.4.3) und
- die Orthogonal Defect Classification (Kapitel 3.4.4).

Die Tabelle 3-5 fasst die Ergebnisse der Darstellung vergleichend zusammen.

	IEEE Standard 1044-1993	**HP-Fehleranalyse**	**ODC**
Anwendungsbereich	Softwareanomalien im gesamten Softwarelebenszyklus	Originäre Fehler im gesamten Softwarelebenszyklus	Entwurfs- und Implementierungsfehler während der Entwicklung und des Betriebs einer Software
Anwendbarkeit	▪ Standard beschreibt keine Vorgehensweise ▪ Elemente des Klassifikationsschema werden nicht definiert ▪ Auswertung von Fehlern und die Ursachenanalyse werden nicht berücksichtigt.	▪ genaue Beschreibung der Vorgehensweise ▪ Elemente des Klassifikationsschemas werden definiert, aber nicht präzise abgegrenzt. ▪ kleines Klassifikationsschema. ▪ wenige Möglichkeiten der Auswertung. ▪ In der Ursachenanalyse wird nur eine Auswahl von Fehlern berücksichtigt.	▪ Eine durchgehende Beschreibung der Vorgehensweise fehlt; viele Beispiele und Hinweise. ▪ Elemente des Klassifikationsschemas werden definiert und weitgehend präzise abgegrenzt. ▪ Zahlreiche Möglichkeiten der Auswertung. ▪ Methoden zur manuellen und automatischen Auswertung von Fehlern. ▪ Mit dem Klassifikationsschema ist eine detaillierte Beschreibung von Mustern in einer Fehlermenge möglich.
theoretische Fundierung	nicht theoretisch fundiert	nicht theoretisch fundiert	rudimentär theoretisch fundiert
empirische Befunde	keine bekannt	nur von Grady veröffentlichten Erfahrungen aus HP bekannt	mehrere empirische Untersuchungen mit verschiedenen Zielsetzungen

Tabelle 3-5: Vergleich der fehlerbasierten Verfahren

3.5 Zusammenfassung

Das Kapitel 3 hatte den Stand der Forschung zu Fehlen und fehlerbasierten Verfahren zum Gegenstand. Zwei zentrale Ergebnisse wurden dabei herausgearbeitet.

> *Ergebnis 1: Mängel in der Anforderungsanalyse verursachen viele Fehler, deren nachträgliche Korrektur aufwändig ist.*

Erschwerend zu dieser Feststellung kommt hinzu, dass häufig missverstandene oder übersehene Anforderungen die Anforderungsfehler ausmachen.[281] Hierbei handelt es sich um jene Feh-

[281] Vgl. hierzu die für den Inhalt des Praxisproblems beschriebenen Ergebnisse der Studien in Kapitel 1.1.2.

ler, die erst entdeckt werden, wenn die Software vor Benutzern ausgeführt wird oder zumindest die Ausführung simuliert wird. Die Anforderungsspezifikation als Grundlage für das Testen ist in diesem Falle nicht hilfreich. Folglich sind es Fehler, die mit großer Wahrscheinlichkeit spät entdeckt werden, sofern nicht vorbeugende Maßnahmen ergriffen werden. Die Folgen sind bekannt: Softwareprojekte werden zu spät ausgeliefert, die Kosten für die Entwicklung übersteigen häufig das ursprünglich Geplante um ein Vielfaches und die ausgelieferte Software erfüllt die Kundenbedürfnisse nicht.[282]

Ergebnis 2: Die Orthogonal Defect Classification ist ein etabliertes fehlerbasiertes Verfahren, um die Ursachen von Fehlern zu bestimmen.

In der Literatur werden drei fehlerbasierte Verfahren vorgeschlagen: der IEEE-Standard 1044-1993, die Hewlett-Packard-Fehleranalyse und die Orthogonal Defect Classification nach Chillarege, kurz ODC. Auf der Grundlage der zur Verfügung stehenden Literatur und empirischer Erkenntnisse ist ODC im Vergleich zu den übrigen zwei Verfahren ausgereifter und in Forschung und Praxis etablierter. Jedoch hat ODC ebenso deutliche Schwächen. ODC ist wie die beiden anderen fehlerbasierten Verfahren nicht ausreichend theoretisch begründet und unterstützt die Identifikation und Analyse von Prozessmängeln in der Anforderungsanalyse nur rudimentär.

Trotz der Schwächen der ODC wird sie in dieser Arbeit als ein Ausgangspunkt gewählt. Sie wird überarbeitet und weiterentwickelt, so dass als Ergebnis ein begründetes Verfahren vorliegt, um Fehler zu erfassen und auszuwerten mit dem Ziel, entwicklungsbegleitend Mängel im Prozess der Anforderungsanalyse zu finden.

[282] Vgl. The Standish Group /CHAOS/ 1-9.

4 Erklärungsmodell für Anforderungsfehler

In Kapitel 4 wird ein Erklärungsmodell herausgearbeitet, das erklärt, warum Anforderungsfehler und Folgefehler durch Mängel im Prozess der Anforderungsanalyse entstehen. Hierzu werden zunächst in Kapitel 4.1 Fehler und ihre Merkmale beleuchtet und in Kapitel 4.2 das Konstrukt eines Prozessmangels erörtert. Anhand einer systematischen Literaturanalyse werden in Kapitel 4.3 empirische Befunde zu Zusammenhängen zwischen Prozessmängeln in der Anforderungsanalyse und verursachten Fehlern zusammengetragen. Mittels Induktion fließen die Erkenntnisse dieser drei Unterkapitel in Form von Gesetzeshypothesen in ein Erklärungsmodell ein, das Gegenstand von Kapitel 4.4 ist.

4.1 Fehler und ihre Merkmale

Am 9. September 1945 dringt eine Motte in einen Computer ein und führt zu dessen Fehlfunktion.[283] Der Motte ist sicherlich nicht bewusst, dass dies der Auslöser dafür wird, mehr als 60 Jahre einen Softwarefehler im Englischen auch als „bug“[284] zu bezeichnen. Und dies, obwohl sie streng genommen weniger die Software als mehr die Hardware des Computers beeinträchtigt.

Ein Fehler wird sowohl in der Literatur als auch in der Praxis mit sehr unterschiedlichen Inhalten verknüpft. Es stellt sich also Frage, was genau ein Fehler ist. Nachfolgend wird der Begriff des Fehlers aufbauend auf der Definition aus Kapitel 2.2.5 weiter konkretisiert und von anderen Begriffen abgegrenzt.

Definition und Abgrenzung des Fehlerbegriffs

Der IEEE-Standard 610.12-1990 unterscheidet die Begriffe Irrtum (engl. error oder mistake), Fehler (engl. fault) und Fehlverhalten (engl. failure),[285] welche hier konzeptionell erweitert

[283] Vgl. Zuse /Software measurement/ 628 f.

[284] „bug“ ist die englische Begriff für einen Insekt bzw. Käfer.

[285] Die Begriffe Irrtum, Fehler, Fehlverhalten sind angelehnt an den Definition von error, fault und failure in IEEE /Glossary/ 29 f. Vgl. zum Begriff Irrtum die vierte Definition zu „error“ in IEEE /Glossary/ 29, zum Begriff Fehler die zweite Definition von „fault“ in IEEE /Glossary/ 30 und zum Begriff Fehlverhalten die Definition von „failure“ in IEEE /Glossary/ 30. Die Übersetzungen der Begriffe error, fault und failure wurden übernommen aus Liggesmeyer /Software-Qualität/ 7. Der IEEE Standard 610.12-1990 versucht die uneinheitliche Verwendung der Begriffe „error“, „fault“ und „failure“ durch mehrere Definitionsvorschläge abzubilden.

werden. Die Begriffe werden im Folgenden präzisiert und in Abbildung 4-1 zusammenfassend dargestellt.[286]

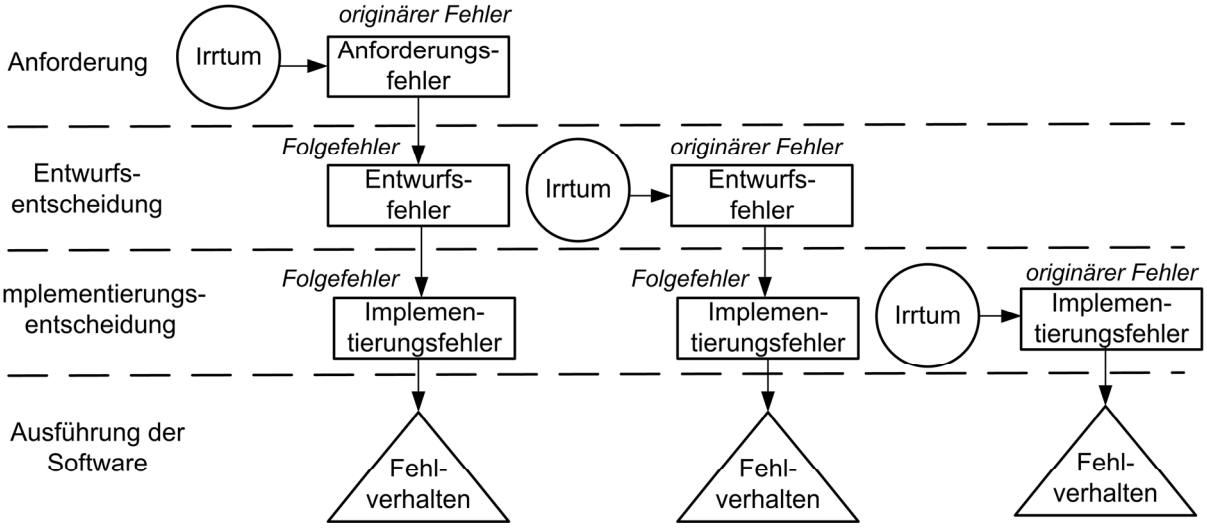


Abbildung 4-1: Entstehung von Fehlern und ihre Auswirkungen

Ein Projektmitarbeiter kann sich irren, während er ein Arbeitsergebnis erstellt, d. h. Softwareanforderungen erarbeitet, Entwurfsentscheidungen trifft oder die Software implementiert. Sein *Irrtum* kann dazu führen, dass das bearbeitete Ergebnis mit einem oder mehreren Fehlern behaftet ist.[287] Dabei sind unter einem Irrtum alle Ereignisse zu verstehen, in denen eine geplante Abfolge von geistigen oder körperlichen Aktivitäten nicht zum beabsichtigten Ergebnis führt, sofern dies nicht einem fremden Einwirken zugeschrieben werden kann.[288] Ein *Fehler* ist ein unzureichendes Merkmal oder ein erwartetes, jedoch fehlendes Merkmal eines Arbeitsergebnisses der Softwareentwicklung, sofern es eine Änderung in diesem Ergebnis notwendig macht.[289] Es können Anforderungs-, Entwurfs- und Implementierungsfehler voneinander abgegrenzt werden, je nachdem, ob Entscheidungen in der Anforderungsanalyse, dem Entwurf oder der Implementierung betrachtet werden. Insbesondere werden hier also unter einem Fehler nicht ausschließlich Implementierungsfehler aufgefasst.

Da einzelne Entscheidungen der Teilaufgaben zusammenhängen,[290] kann sich ein Anforderungsfehler als Entwurfs- und Implementierungsfehler fortsetzen. Diese besonderen Ent-

---

[286] Es sei darauf hingewiesen, dass sowohl in der Praxis als auch in der Literatur die Begriffe im Umfeld eines Fehlers uneinheitlich verwendet werden. Zu dem Problem der uneinheitlichen Begriffsverwendung siehe z. B. Fenton, Pfleeger /Software metrics/156 f. und Fenton, Neil /Critique/ 678 f.

[287] Vgl. Munson, Nikora /Definition/ 388.

[288] Vgl. Reason /Human error/ 5-9.

[289] Ein Fehler liegt auch dann vor, wenn eine eigentlich notwendige Änderung nicht durchgeführt wird. In Anlehnung an Chillarege /Defect/ 362 f. Chillarege zieht "defect" der Bezeichnung "fault" vor.

[290] Vgl. hierzu die Ausführungen in Kapitel 2.2.3.

wurfs- und Implementierungsfehler sind also *Folgefehler*. Ein *originärer Fehler* liegt dagegen vor, wenn er infolge eines Irrtums neu eingeführt wird und damit unabhängig von einem Fehler einer vorgelagerten Entscheidung ist. Immer wenn ein Projektmitarbeiter einem *Irrtum* unterliegt, finden originäre Fehler Eingang in ein Ergebnis. Eine Software kann ein *Fehlverhalten* offenbaren, wenn sie einen Implementierungsfehler enthält und ausgeführt wird (siehe Abbildung 4-1). Dieses Fehlverhalten ist also die Wirkung eines Implementierungsfehlers. Tritt ein Fehlverhalten auf, folgt daraus ausschließlich, dass die Software mindestens einen Implementierungsfehler hat. Die Kenntnis eines Fehlverhaltens allein erlaubt jedoch keine Rückschlüsse darüber, welche oder wie viele Implementierungsfehler vorhanden sind, da mehrere Implementierungsfehler gemeinsam als ein Fehlverhalten wahrgenommen werden können.

Sofern ein Implementierungsfehler vor der Freigabe der Software für die Nutzung seitens der Kunden gefunden wird, liegt ein *Entwicklungsfehler* vor. Wird er hingegen nach der Freigabe gefunden, wird er im Folgenden als *Produktionsfehler* bezeichnet. Letztere werden in der englischsprachigen Literatur häufig als „field defects“ bezeichnet.[291]

Merkmale eines Fehlers

Ein Fehler kann aus drei unterschiedlichen Perspektiven beleuchtet werden: aus Sicht der Kunden, des Fehlermanagements und des Entwicklungsteams.[292] Anhand dieser drei Perspektiven besteht die Möglichkeit, Merkmale eines Fehlers einer der drei Perspektiven zuzuordnen und so die Merkmale zu gruppieren (vgl. hierzu Abbildung 4-2).

[291] Vgl. zum Beispiel Kan /Metrics/ 91.

[292] In Anlehnung an Grady /Software metrics/ 122 f.

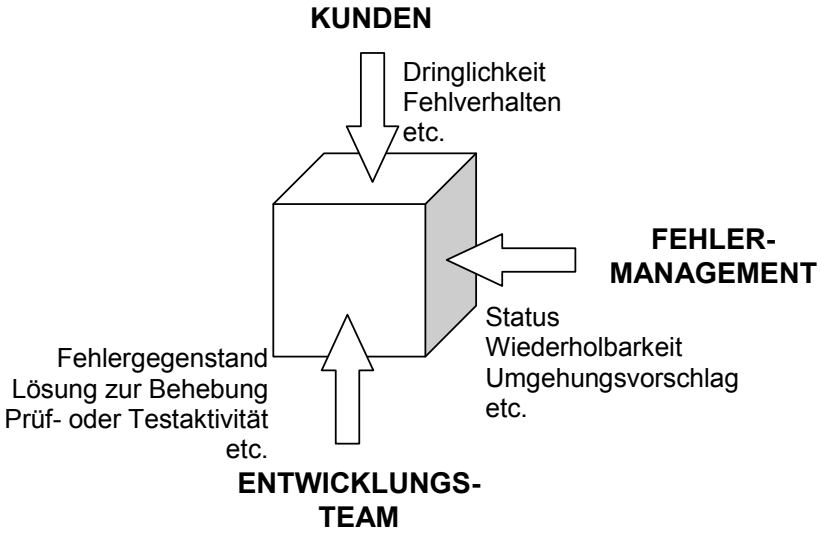


Abbildung 4-2: Fehlerwürfel: drei Perspektiven auf einen Fehler[293]

Merkmale aus Kundensicht betreffen die Auswirkungen eines Fehlers auf die Nutzung der Software und damit verbundene Einschränkungen in Arbeitsabläufen, die durch die Software unterstützt werden sollen. Beispiele für Merkmale in diesem Zusammenhang sind die Beschreibung des eingetretenen oder möglichen Fehlverhaltens und die Dringlichkeit der Behebung.

Das Fehlermanagement wird hier als Schnittstelle zwischen Kunden und Entwicklungsteam einerseits und Teilen des Entwicklungsteams andererseits verstanden. Zum einen bereitet das Fehlermanagement Informationen zu einem Fehler auf, damit das Entwicklungsteam ihn schneller lokalisieren kann. Merkmale hierfür sind zum Beispiel die Version des betroffenen Arbeitsergebnisses und die Information, ob das Fehlverhalten wiederholbar ist oder nur sporadisch auftritt. Zum anderen liefert das Fehlermanagement beispielsweise für das gesamte Entwicklungsteams und den Kunden den aktuellen Stand einer Fehlerbearbeitung. Bei Implementierungsfehlern kann es Kunden Möglichkeiten zur Umgehung eines Fehlers vorschlagen.

Das Entwicklungsteam[294] kann Informationen zu Merkmalen eines Fehlers beisteuern, die sich auf seine Entdeckung oder Behebung beziehen. Zum Beispiel: In welcher Prüf- oder Testaktivität wurde er entdeckt? Was konkret ist falsch? Wie wurde der Fehler behoben?

[293] In Anlehnung an Grady /Software metrics/ 123.

[294] Vgl. zum Begriff des Entwicklungsteams das Kapitel 2.2.1.

4.2 Prozessmängel als Ursachen für Fehler

Vorüberlegungen

Fehler werden durch einen Softwareentwicklungsprozess verursacht. Diese These ist eine kausale Aussage. Kausale Aussagen beschreiben, dass zwei oder mehrere Größen sich beeinflussen und in welche Richtung der Einfluss jeweils wirkt.[295] Entsprechende Aussagen sind ein wesentlicher Bestandteil von gedanklichen Bezugsrahmen.

Um die zu Beginn dieses Kapitels aufgeführte kausale Aussage zu präzisieren, werden zwei Schritte durchgeführt.[296] Zunächst werden Verbundenheitsannahmen formuliert und damit die Frage geklärt, welche Größen sich wie beeinflussen. Ausgehend von einer linearen Kausalität[297] werden im zweiten Schritt Asymmetrieannahmen gebildet. D. h., es wird bestimmt, in welcher Richtung die Beeinflussung zwischen zwei Größen verläuft.

Auf Grundlage dieser kausalen Aussage wird anschließend das Konstrukt eines Prozessmangels begründet. Ausgehend von dem Verständnis, dass ein Prozess ein Bündel von Gestaltungsmaßnahmen ist, ist ein Prozessmangel eine Gestaltungsmaßnahme eines Prozesses, die menschliche Irrtümer begünstigt, so dass Fehler in Arbeitsergebnisse der Softwareentwicklung eingeführt werden.

Verbundenheitsannahme zwischen Entwicklungsprozessen und Fehlern

Hat ein Prozess einen Einfluss auf die Entstehung und damit die Anzahl von Fehlern in den Arbeitsergebnissen des Prozesses? Die Beantwortung dieser Frage ist für ein Verfahren entscheidend, das Anforderungsfehler und Folgefehler vermeiden oder zumindest früh entdecken will, indem Mängel im Prozess identifiziert und mit dieser Kenntnis Verbesserungen am Prozess vorgenommen werden sollen. Zunächst ist es nahe liegend, dass Fehler nicht von selbst in Arbeitsergebnisse eingehen. Vielmehr werden diese Arbeitsergebnisse durch Projektbeteiligte erstellt und folglich auch Fehler von diesen Beteiligten eingeführt. Mehrere Studien zeigen einen Zusammenhang zwischen der Art des Entwicklungsprozesses und Fehlern in der entwickelten Software auf. [298]

---

[295] Vgl. hierzu und zum folgenden Satz Grochla /Organisationstheorie/ 74.

[296] Vgl. Grochla /Organisationstheorie/ 75 f.

[297] Vgl. hierzu Kapitel 1.3.3.

[298] Vgl. hierzu auch die empirischen Befunde in Kapitel 3.2, die bereits einen statistischen Zusammenhang zwischen Prozessen der Anforderungsanalyse und Fehlern im Besonderen beschreiben.

Slaughter, Harter und Krishnan stellen beispielsweise fest, dass die Anzahl der Fehler pro 1000 Zeilen Code nach jeder Verbesserungsinitiative eines Prozesses sinkt.[299] In zwei weiteren Untersuchungen im gleichen Unternehmen kann diese Beobachtung bestätigt werden. Die Autoren zeigen auf, dass jeweils eine höhere Prozessreife im Sinne der Reifestufen nach CMM mit einer geringeren Fehlerrate einhergeht.[300] Krishnan und Kellner untersuchen 45 Projekte in einem Unternehmen dahingehend, in welchem Umfang jeweils ihre Gestaltung mit ausgewählten Gestaltungsempfehlungen des CMM übereinstimmen.[301] Die Autoren stellen fest, dass die Anzahl der ausgelieferten Fehler pro 1000 Codezeilen in einem Projekt umso geringer ist, je größer die Übereinstimmung zwischen den untersuchten Gestaltungsempfehlungen und der jeweiligen Projektgestaltung ist.
Neufelder untersucht, wie einzelne Entwicklungspraktiken in Softwareentwicklungen mit der Fehlerrate korrelieren und bestimmt damit insbesondere die zehn Praktiken, die am stärksten mit einer geringen Fehlerrate korrelieren.[302]

Asymmetrieannahmen zwischen Entwicklungsprozessen und Fehlern

Auch wenn die oben aufgeführten empirischen Studien einen Zusammenhang zwischen einem Prozess und den Fehlern in den Arbeitsergebnissen feststellen, beschreiben sie zunächst nur einen *statistischen* Zusammenhang. Dieser ist zwar notwendig, aber nicht hinreichend für einen *kausalen* Zusammenhang. Entscheidend ist jedoch, ob ein kausaler Zusammenhang zwischen Prozessen und Fehlern besteht, da ein statistischer Zusammenhang zwischen zwei Größen auch dann vorliegen kann, wenn er inhaltlich nicht gerechtfertig ist.[303] Um einen kausalen Zusammenhang zu begründen, müssen sachlogische Überlegungen herangezogen werden und die Richtung der Wirkung erklärt werden. Wie kann also ein kausaler Zusammenhang zwischen Prozessen und Fehlern begründet werden?
Originäre Fehler entstehen aufgrund von Irrtümern der Projektmitarbeiter.[304] James Reason beschäftigt sich aus der psychologischen Perspektive eingehend mit dem Thema des menschlichen Irrtums in seiner Monografie „Human Error“. Im Sinne seiner Arbeit wird hier ein Irrtum (error) definiert als alle Ereignisse, in denen eine geplante Abfolge von geistigen oder

---

[299] Vgl. zu diesem Absatz Slaughter, Harter, Krishnan /Software quality/ 69-73

[300] Vgl. Harter, Krishnan, Slaughter /Process maturity/ 454, 461-464 und Harter, Slaughter /Quality improvement/ 786, 791-797. In beiden Studien wird der Kehrwert der Fehlerrate als Produktqualität bezeichnet, also die Anzahl Codezeilen pro Fehler. Ergebnis beider Studien ist, dass bei einer höheren Prozessreife auch eine höhere Produktqualität und damit eine geringere Fehlerrate festzustellen ist.

[301] Vgl. Krishnan, Kellner /Process Consistency/ 807-814.

[302] Vgl. Neufelder /Impact/ 148-153.

[303] Vgl. hierzu und zum nächsten Satz Fahrmeir u.a. /Statistik/ 148-150.

[304] Vgl. hierzu Kapitel 4.1.

körperlichen Aktivitäten nicht zum beabsichtigten Ergebnis führt, sofern dies nicht fremdem Einwirken zugeschrieben werden kann.[305] Reason weist daraufhin, dass es nur dann zweckmäßig ist von einem Irrtum eines Individuums zu sprechen, wenn zuvor die Absicht zur Handlung vorlag. Eine Absicht liegt vor, wenn dem Individuum das erwartete Ergebnis und die Mittel bewusst sind, mit denen das Ergebnis erreicht werden soll. Somit umfasst der Begriff des Irrtums weder unfreiwillige noch spontane Handlungen.

Reason unterscheidet zwei Arten von Irrtümern. Entweder führt ein Individuum seine Handlungen nicht so aus, wie von ihm eigentlich geplant, oder es führt sie wie geplant aus, muss jedoch feststellen, dass sie nicht angemessen sind, um das beabsichtigte Ergebnis zu erreichen.[306]

Ein Prozess beschreibt, wie eine Aufgabe durchgeführt wird bzw. werden soll. Es definiert folglich einen Teil des möglichen Spielraums an Handlungen. Insofern kann ein Prozess Irrtümer begünstigen, aber auch verringern, je nachdem, ob er für die gegebenen Rahmenbedingungen zweckmäßige Gestaltungsmaßnahmen umfasst oder nicht.

Werden durch eine Gestaltungsmaßnahme eines Prozesses Irrtümer begünstigt, können in Arbeitsergebnissen originäre Fehler entstehen. Werden diese originären Fehler nicht rechtzeitig entdeckt, können sie Folgefehler verursachen. Folglich kann auch sachlogisch begründet werden, dass ein Prozess beeinflusst, ob Fehler in Arbeitsergebnissen entstehen

Konstrukt eines Prozessmangels

Auf der Grundlage des kausalen Zusammenhangs zwischen Entwicklungsprozessen und Fehlern wird das Konstrukt eines Prozessmangels begründet. Ein *Mangel in einem Prozess*, kurz *Prozessmangel*, ist eine Gestaltungsmaßnahme eines Prozesses, die menschliche Irrtümer begünstigt, so dass Fehler in Arbeitsergebnisse der Softwareentwicklung eingeführt werden. Werden diese originären Fehler nicht entdeckt, können zudem im weiteren Verlauf Folgefehler entstehen. Diese Zusammenhänge werden in der Abbildung 4-3 für Prozessmängel einer Anforderungsanalyse als Ursache-Wirkungskette konkretisiert.

[305] Vgl. hierzu und im Folgenden Reason /Human error/ 5-9.

[306] Vgl. Reason /Human error/ 7 f.

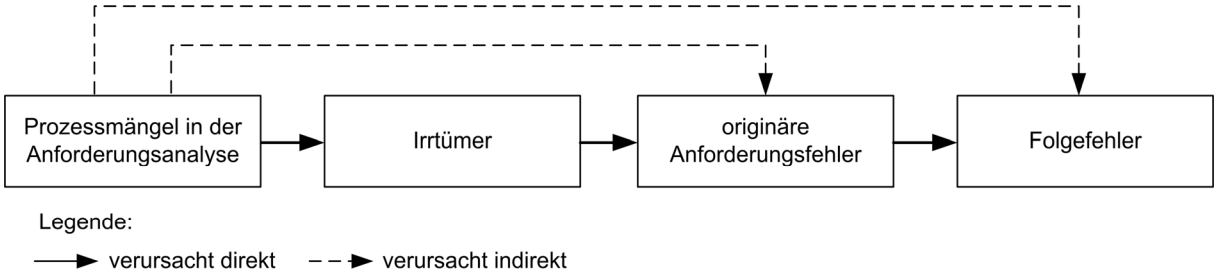


Abbildung 4-3: Ursache-Wirkungskette für Prozessmängel in der Anforderungsanalyse

In der Abbildung 4-3 ist zu erkennen, dass Prozessmängel nicht direkt Anforderungsfehler und ihre Folgefehler verursachen, sondern indirekt. Trotzdem sind es die Prozessmängel, die als Ursache im Mittelpunkt der Betrachtung stehen müssen, da sie der Ausgangspunkt für die entstandenen Fehler sind.[307] Erst durch ihre Beseitigung, d. h. die Verbesserung des Prozesses der Anforderungsanalyse, können Fehler nachhaltig vermieden werden. Sofern die Prozessmängel der Anforderungsanalyse nicht beseitigt werden können, besteht die Möglichkeit, den Prozess der Qualitätssicherung gezielt zu verbessern, um entsprechende Anforderungsfehler und ihre Folgefehler früher entdecken zu können.

Da der Begriff des Prozessmangels auf dem des Prozesses aufbaut, kann sich auch ein Prozessmangel auf jeden Gestaltungsbereich beziehen, der die Durchführung der Anforderungsanalyse als Teilaufgabe beschreibt.[308] Um Missverständnisse zu vermeiden, wird zudem der Begriff des Mangels in dieser Arbeit ausschließlich mit Bezug auf einen Prozess verwendet und nicht auf die Arbeitsergebnisse eines Prozesses.

[307] Prozessmängel sind ein Beispiel dafür, was in der angloamerikanischen Literatur mit root cause als die eigentliche Grundursache bezeichnet wird. Vgl. hierzu und im Folgenden Rooney, Heuvel /Root cause analysis/ 46.

[308] Vgl. Kapitel 2.2.2.

4.3 Prozessmängel in der Anforderungsanalyse und Anforderungsfehler

In Kapitel 4.2 wurde die kausale Aussage begründet, dass Fehler durch Softwareentwicklungsprozesse verursacht werden. Hierauf aufbauend werden in diesem Kapitel aus empirischen Untersuchungen Aussagen extrahiert, die Hinweise darüber geben, welche Maßnahmen eines Prozesses der Anforderungsanalyse Prozessmängel im Sinne dieser Arbeit sein können. Wenn es möglich ist, werden typische Gemeinsamkeiten der Anforderungsfehler eines Prozessmangels beschrieben.

Hierzu wird eine systematische Literaturanalyse durchgeführt. Diese Forschungsmethode wird in Kapitel 4.3.1 eingeführt. In Kapitel 4.3.2 wird dargelegt, wie diese Forschungsmethode eingesetzt wird. Das Ergebnis der Literaturrecherche sind empirische Literaturquellen, die bestimmte Kriterien erfüllen. Die Literaturquellen werden in Kapitel 4.3.3 dargestellt und bewertet. Aus den Literaturquellen werden in Kapitel 4.3.4 empirische Befunde zu Zusammenhängen zwischen Prozessmängeln in der Anforderungsanalyse und verursachten Fehlern zusammengetragen.

4.3.1 Systematische Literaturanalyse als Forschungsmethode

Eine systematische Literaturanalyse im Sinne eines „literature reviews“ nach Webster und Watson[309] untersucht mit einem definierten Ziel einen festgelegten Ausschnitt der Literatur, um eine Synthese relevanter Aussagen zu erreichen. Der nordamerikanische Fachverband für Psychologie, die American Psychological Association, definiert das Ergebnis einer systematischen Literaturanalyse und damit die Forschungsmethode wie folgt:

> *„Review articles, including meta-analyses, are critical evaluations of material that has already been published. By organizing, integrating and evaluating previously published material, the author of a review article considers the progress of current research toward clarifying a problem. In a sense, a review article is a tutorial in that the author*
>
> - *defines and clarifies the problem;*
> - *summarizes previous investigations in order to inform the reader of the state of current research;*
> - *identifies relations, contradictions, gaps, and inconsistencies in the literature; and*

[309] Vgl. Webster, Watson /Literature review/ xiii-xxiii.

- *suggests the next step or steps in solving the problem.“*[310]

Allgemeine Empfehlungen zur systematischen Literaturanalyse finden sich insbesondere in den Arbeiten von Webster und Watson[311], Bem[312] und Eisenberg[313].

4.3.2 Vorgehensweise bei der systematischen Literaturanalyse

4.3.2.1 Ziele der Literaturanalyse in dieser Arbeit

Zielfragen

Die Literaturanalyse soll folgende Fragen beantworten:

1. *Welche Prozessmängel der Anforderungsanalyse sind gemäß Hinweisen in empirischen Untersuchungen bekannt?*
2. *Welche Gemeinsamkeiten weisen die durch Prozessmängel verursachten Fehler jeweils auf?*

Um diese Fragen zu beantworten, wird auf zwei Größen zurückgegriffen, die in den vorangegangen Kapitel schrittweise herausgearbeitet worden sind:

- Ein *Fehler* ist ein unzureichendes Merkmal oder ein erwartetes, jedoch fehlendes Merkmal eines Arbeitsergebnisses der Softwareentwicklung, sofern es eine Änderung in diesem Ergebnis notwendig macht.
- Ein *Prozessmangel der Anforderungsanalyse* ist eine Gestaltungsmaßnahme in einem entsprechenden Prozess, die menschliche Irrtümer begünstigt, so dass Anforderungsfehler entstehen.

Der Zusammenhang zwischen diesen beiden Größen wird genutzt, um die untersuchte Literatur nach inhaltlichen Mustern zu organisieren. Hierzu werden Prozessmängel inhaltlich nach Gestaltungsbereichen gruppiert und bestimmte Merkmale von Fehlern identifiziert und voneinander abgegrenzt. Das Ergebnis der Literaturanalyse erweitert den gedanklicher Bezugsrahmen dieser Arbeit um eine Synthese relevanter empirischer Befunde in der Literatur. Dieser Bezugsrahmen hat explorativen Charakter,[314] da er versucht spezifische Zusammenhänge zwischen Maßnahmen in der Anforderungsanalyse und Fehlern erstmalig zu erkennen.

---

[310] American Psychological Association /Publication manual/ 7.
[311] Vgl. Webster, Watson /Literature review/ xiii-xxiii.
[312] Vgl. Bem /Review article/ 172-177.
[313] Vgl. Eisenberg /Literature review/ 17-34.
[314] Vgl. hierzu die Ausführungen in 1.3.4.2.

Gemäß Eisenberg handelt es sich in diesem Fall um eine Anwendung der systematischen Literaturanalyse, um neues Wissen zu generieren.[315]

Beitrag des entwickelten Bezugsrahmen

Der Bezugsrahmen benennt in der empirischen Literatur identifizierte Prozessmängel der Anforderungsanalyse. Ein Zusammenhang zwischen einem Prozess und Fehlern reicht nicht aus, um durch Fehler auf Mängel im Prozess zu schließen. Ein Prozessmangel muss nicht nur Fehler verursachen, sondern diese Fehler müssen Gemeinsamkeiten aufweisen. Erst so eröffnet sich die Möglichkeit, nach Mustern in einer Fehlermenge zu suchen. Erst diese Gemeinsamkeiten ermöglichen eine gezielte Suche nach Maßnahmen des betroffenen Prozesses, die als potenzielle Mängel in Fragen kommen können. Der entwickelte Bezugsrahmen zeigt, soweit möglich, welche Gemeinsamkeiten die Fehler besitzen, die durch einen bestimmten Prozessmangel verursacht worden sind.

4.3.2.2 Auswahl der Literaturquellen und Ablauf der Literaturrecherche

Auswahlkriterien

Es werden ausschließlich Literaturquellen in der Literaturanalyse berücksichtigt, die die folgenden drei Kriterien erfüllen:

1. Die Literaturquelle ist empirisch, d. h. gemachte Aussagen beruhen auf Beobachtungen der Realität.
2. Die empirischen Aussagen beziehen sich auf die Entwicklung oder Wartung von Software. Sofern sich Aussagen auf Systemsoftware oder systemnahe Software beziehen,[316] werden sie nur dann berücksichtigt, sofern sie auch auf Anwendungssoftware übertragbar sind.
3. Die empirischen Aussagen geben explizit Hinweise auf Maßnahmen im Prozess der Anforderungsanalyse, die Fehler verursachen.[317] Diese Aussagen können auch ein

---

[315] Eisenberg unterscheidet fünf Typen von Literaturanalysen. Die in diesem Kapitel verfolgte Literaturanalyse entspricht ihrem ersten Typ, den sie wie folgt beschreibt: „In a common type of review, an author uses existing empirical data to answer old or new questions. [...] A review of this type [...] can be based on the author's qualitative review of the literature. In this type of review, the major contribution generally is the *generation of new knowledge* [...]." Vgl. Eisenberg /Literature review/ 19 f.

[316] Vgl. hierzu Kapitel 2.2.1.

[317] „Explizit" heißt in diesem Kontext insbesondere, dass beobachtete Unzulänglichkeiten der Anforderungsanalyse aus empirischen Untersuchungen nicht berücksichtigt werden, sofern offen bleibt, ob sie wirklich Fehler verursachen oder ihre Auswirkungen beispielsweise sich auf die Erhöhung des erforderlichen Aufwands beschränken.

„Nebenprodukt“ einer empirischen Untersuchung sein, die eigentlich andere Ziele verfolgt.

Die genannten Kriterien machen deutlich, dass eine automatische Suche in Literaturdatenbanken anhand ausgewählter Schlüsselbegriffe nicht zweckmäßig ist. Stattdessen wurde die Zusammenfassung von jeder Literaturquelle untersucht. Sofern anhand dieser Informationen nicht geklärt werden konnte, ob eines der Kriterien verletzt wird, wurden ausgewählte Bereiche des Quelleninhalts untersucht.

Auswahl der wissenschaftlichen Zeitschriften

Im ersten Schritt wurden Beiträge aus 16 wissenschaftlichen Zeitschriften (vgl. Tabelle 4-1) in die Literaturanalyse einbezogen. Es wurden alle Ausgaben einer Zeitschrift berücksichtigt, die im Zeitraum von Anfang Januar 1990 bis einschließlich Ende Dezember 2005 veröffentlicht wurden.

Nr.	Titel der Zeitschrift	Abkürzung	Rang	Untersuchte Ausgaben	Anmerkungen
Z1	Communications of the AIS	CAIS	B	Nr. 1, Jg. 1, 1999 bis Nr. 41, Jg. 15, 2005	1
Z2	Communications of the ACM	CACM	A	Nr. 1, Jg. 33, 1990 bis Nr. 12, Jg. 48, 2005	-
Z3	Empirical Software Engineering	ESE	-	Nr. 1, Jg. 1, 1996 bis Nr. 4, Jg. 10, 2005	1
Z4	European Journal of Information Systems	EJIS	B	Nr. 1, Jg. 6, 1997 bis Nr. 4, Jg. 14, 2005	2
Z5	IBM Systems Journal	IBMSJ	C	Nr. 1, Jg. 29, 1990 bis Nr. 4, Jg. 44, 2005	-
Z6	IEEE Software	IEEESw	C	Nr. 1, Jg. 7, 1990 bis Nr. 6, Jg. 22, 2005	-
Z7	IEEE Transactions on Software Engineering	IEEETSE	B	Nr. 1, Jg. 16, 1990 bis Nr. 12, Jg. 31, 2005	-
Z8	Information Systems Journal	ISJ	C	Nr. 1, Jg. 6, 1990 bis Nr. 4, Jg. 15, 2005	-
Z9	Information Systems Research	ISR	A	Nr. 1, Jg. 1, 1990 bis Nr. 4, Jg. 16, 2005	-
Z10	Journal of Management Information Systems	JMIS	A	Nr. 4, Jg. 6, 1990 bis Nr. 2, Jg. 22, 2005	-
Z11	Journal of the Association for Information Systems	JAIS	C	Nr.1, Jg. 1, 2000 bis Nr. 9, Jg. 6, 2005	1
Z12	Management Science	MS	A	Nr. 1, Jg. 36, 1990 bis Nr. 12, Jg. 51, 2005	-
Z13	MIS Quarterly	MISQ	A	Nr. 1, Jg. 14, 1990 bis Nr. 4, Jg. 29, 2005	-
Z14	Requirements Engineering	RE	-	Nr. 1, Jg. 1, 1996 bis Nr. 4, Jg. 10, 2005	1
Z15	Software Quality Journal	SQJ	-	Nr. 1, Jg. 6, 1997 bis Nr. 4, Jg. 13, 2005	2
Z16	Wirtschafts-informatik	WIRT	C	Nr. 1, Jg. 37, 1995 bis Nr. 6, Jg. 47, 2005	2

Tabelle 4-1: Auswahl der wissenschaftlichen Zeitschriften

Die Tabelle 4-1 führt neben einer fortlaufenden Nummer den Titel der wissenschaftlichen Zeitschrift, die Abkürzung des Titels gemäß einer Abkürzungsliste auf ISWorld.org[318], den Rang der Zeitschrift in Anlehnung an eine Rangliste auf ISWorld.org[319], die berücksichtigten Ausgaben sowie Anmerkungen zur Zeitschrift.

In Tabelle 4-1 haben die Zahlen in der Spalte „Anmerkungen“ folgende Bedeutung:

- 1: Die Zeitschrift wurde erstmalig nach 1990 veröffentlicht. Sie wurde berücksichtigt, obwohl ihre Ausgaben, nicht den Zeitraum von 1990 bis 2005 abdecken.

[318] Vgl. http://www.isworld.org/csaunders/jcode.htm, Stand 25.07.2006.

[319] Vgl. http://www.isworld.org/csaunders/rankings.htm, Stand 25.07.2006.

- 2: Ältere Ausgaben der Zeitschrift standen online nicht zur Verfügung. Die Zeitschrift wurde ab der ersten Ausgabe berücksichtigt, auf die online zugegriffen werden konnte.

Die Zeitschriftenrangliste auf ISWorld.org beruht auf neun empirischen Untersuchungen, in denen wissenschaftliche Zeitschriften, die Informationssysteme zum Gegenstand haben, bewertet wurden. Nachfolgend werden entsprechende Zeitschriften kurz als IS-Zeitschrift bezeichnet. Die Ergebnisse dieser Untersuchungen werden in einer Rangliste zusammengetragen, in der die Zeitschriften nach absteigendem Rang sortiert sind. In der Tabelle 4-1 haben die Buchstaben in der Spalte „Rang“ in diesem Zusammenhang folgende Bedeutung:

- A: Die Zeitschrift wird in der Rangliste unter den ersten 10 Zeitschriften aufgeführt. Sie kann als eine IS-Zeitschrift angesehen werden, in der stets wissenschaftlich hochwertige Beiträge veröffentlicht werden.
- B: Die Zeitschrift wird in der Rangliste zwischen dem 11. und dem 20. Rang aufgeführt. Sie kann als eine IS-Zeitschrift angesehen werden, in der regelmäßig wissenschaftlich hochwertige Beiträge veröffentlicht werden.
- C: Die Zeitschrift wird in der Rangliste ab dem 21. Rang aufgeführt. Sie kann als eine IS-Zeitschrift angesehen werden, in der auch wissenschaftlich hochwertige Beiträge veröffentlicht werden.
- -: Die Zeitschrift wird in der Rangliste nicht aufgeführt. Dies kann bedeuten, dass sie selten wissenschaftlich hochwertige Beiträge beinhaltet. Es können aber auch andere Gründe vorliegen. So kann es beispielsweise sein, dass es eine junge Zeitschrift ist, die deshalb noch nicht berücksichtigt worden ist. Ebenso kann es sein, dass eine Zeitschrift einen sehr engen thematischen Schwerpunkt verfolgt. Entsprechende IS-Zeitschriften werden in empirischen Untersuchungen, die Zeitschriften bewerten, in der Regel nicht aufgenommen und finden sich deshalb auch in der ISWorld.org-Rangliste nicht wieder. Es kann also nicht ausgeschlossen werden, dass auch Zeitschriften, die nicht in der Rangliste aufgeführt sind, ggf. unter wissenschaftlichen Gesichtspunkten hochwertige Beiträge umfassen.

Webster und Watson empfehlen eine Literaturanalyse mit wissenschaftlich renommierten IS-Zeitschriften zu beginnen, also beispielsweise Zeitschriften des Rangs A, sich aber nicht auf diese zu beschränken.[320] Vielmehr sollen möglichst viele veröffentlichte Beiträge berücksichtigt werden, die für das verfolgte Ziel einer Literaturanalyse relevant sind. In der vorlie-

[320] Vgl. Webster, Watson /Literature Review/ xv f.

genden Literaturanalyse haben von 16 Zeitschriften 5 (31%) den Rang A, 3 (19%) den Rang B, 5 (31%) den Rang C und 3 (19%) werden nicht in der Rangliste aufgeführt.

Die Tabelle 4-2 nennt zusätzliche Informationen zu den 16 Zeitschriften: die ISSN[321], Webadressen zur der Zeitschrift in Form einer URL und die herausgebende Körperschaft. Auffällig ist, dass 12 von 16 Zeitschriften von Körperschaften herausgegeben werden, die ihren Sitz in den USA haben. Auch wenn eine bessere geographische Verteilung wünschenswert gewesen wäre,[322] war es nicht möglich, da de facto führende IS-Zeitschriften aus den USA kommen. Mit der Zeitschrift Wirtschaftsinformatik wurde die einzige deutsche IS-Zeitschrift berücksichtigt.

[321] Die International Standard Serial Number, kurz ISSN, ist eine aus acht Ziffern bestehende Nummer, die Zeitschriften und Schriftenreihen eindeutig identifiziert. Vgl. http://www.issn.org/en/node/64, Stand 04.09.2006.

[322] Vgl. Webster, Watson /Literature Review/ xv f.

Nr.	Zeit-schrift	ISSN[323]	URL (Stand 01.09.06)	herausgebende Körperschaft
Z1	CAIS	1529-3181	http://cais.isworld.org/	Association for Information Systems Atlanta (USA, Georgia)
Z2	CACM	0001-0782	http://www.acm.org/pubs/cacm/	Association for Computing Machinery New York (USA, New York)
Z3	ESE	1573-7616	http://www.springer.com/journal/10664/	Springer US New York (USA, New York)
Z4	EJIS	1476-9344	http://www.palgrave-journals.com/ejis/	Palgrave Macmillan Journals Basingstoke (England,Hampshire)
Z5	IBM SJ	0018-8670	http://researchweb.watson.ibm.com/journal/sj/	IBM Technical Journals Yorktown Heights (USA, New York)
Z6	IEEESw	0740-7459	http://www.computer.org/software/	IEEE Computer Society Los Alamitos (USA, California)
Z7	IEEETSE	0098-5589	http://www.computer.org/tse/	IEEE Computer Society Los Alamitos (USA, California)
Z8	ISJ	1365-2575	http://www.blackwellpublishing.com/journals/isj/ http://disc.brunel.ac.uk/isj/	Blackwell Publishing Ltd Oxford (England)
Z9	ISR	1526-5536	http://isr.pubs.informs.org/	Institute for Operations Research and the Management Sciences Hanover (USA, Maryland)
Z10	JMIS	0742-1222	http://jmis.bentley.edu/	M.E. Sharpe, Inc. Armonk (USA, New York)
Z11	JAIS	1536-9323	http://jais.isworld.org/	Association for Information Systems Atlanta (USA, Georgia)
Z12	MS	1526-5501	http://mansci.pubs.informs.org/	Institute for Operations Research and the Management Sciences Hanover (USA, Maryland)
Z13	MISQ	0276-7783	http://www.misq.org/	MIS Quarterly Carlson School of Management University of Minnesota Minneapolis (USA, Minnesota)
Z14	RE	1432-010X	http://www.springerlink.com/link.asp?id=102830 http://rej.co.umist.ac.uk/	Springer Verlag London Ltd. Godalming (England, Surrey)
Z15	SQJ	1573-1367	http://www.springerlink.com/link.asp?id=100222	Springer Boston/Norwell Norwell (USA, Massachusetts)
Z16	WIRT	0937-6429	http://www.wirtschaftsinformatik.de	Vieweg Verlag Wiesbaden (Deutschland)

Tabelle 4-2: Zusätzliche Information zu den Zeitschriften

Auswahl der Konferenzbände

Um mit der Literaturanalyse möglichst ein großes Spektrum von Literatur abzudecken, wurden zusätzlich die Konferenzbände von 8 etablierten Konferenzen (vgl. Tabelle 4-3) in die Li-

[323] Sofern eine Zeitschrift für die gedruckte Version und die Online-Version zwei unterschiedliche ISSN hat, wird nur die ISSN der Online-Version genannt.

teraturanalyse einbezogen.[324] Analog zu den Zeitschriften wurden alle Konferenzbände berücksichtigt, die im Zeitraum von Anfang Januar 1990 bis einschließlich Ende Dezember 2005 veröffentlicht wurden. Die Konferenzen finden in der Regel jährlich statt.

Nr.	Titel der Konferenz	Abkürzung	Untersuchte Konferenzbände	herausgebende Körperschaft	Anmerkungen
K1	International Symposium on Software Metrics	METRICS	1993 bis 2005	IEEE	1
K2	International Symposium on Software Reliability Engineering	ISSRE	1991 bis 2005	IEEE	2
K3	International Symposium on Empirical Software Engineering	ISESE	2002 bis 2005	IEEE	1
K4	IEEE International Conference on Software Maintenance	ICSM	1990 bis 2005	IEEE	-
K5	International Conference on Software Engineering	ICSE	1990 bis 2005	IEEE, ACM	-
K6	IEEE International Symposium on Requirements Engineering	ISRE[325]	1993 bis 2001	IEEE	1, 3
K7	IEEE International Conference on Requirements Engineering	ICRE	1994 bis 2005	IEEE	1, 3
K8	International Conference on Information Systems	ICIS	1990 bis 2005	Association for Information Systems	-

Tabelle 4-3: Auswahl der Konferenzbände

Die Tabelle 4-3 führt neben einer fortlaufenden Nummer, den Titel der Konferenz, die gebräuchliche Abkürzung für den Konferenztitel, die berücksichtigten Konferenzbände, die herausgebende Körperschaft sowie Anmerkungen zur Konferenz auf.

In der Tabelle 4-3 haben die Zahlen in der Spalte „Anmerkungen“ folgende Bedeutung:

- 1: Die Konferenz hat erstmalig nach 1990 stattgefunden. Die Konferenzbände dieser Konferenz decken also nicht den Zeitraum von 1990 bis 2005 ab.
- 2: Das Konferenzband zur ISSRE 1990 stand online nicht zur Verfügung. Die Konferenz wurde deshalb erst ab 1991 berücksichtigt.
- 3: ISRE und ICRE fanden bis 2001 jeweils abwechselnd statt (ISRE: 1993, 1995, 1997, 1999, 2001 und ICRE: 1994, 1996, 1998, 2000). Ab 2002 fusionierten beide Konferenzen zu ICRE, welche bis 2005 jährlich stattfand.[326]

[324] Die renommierten Konferenzen Hawaii International Conference on System Sciences (HICSS) und die European Conference on Information Systems (ECIS) wurden nicht berücksichtigt, da zum Zeitpunkt der Literaturanalyse auf die Konferenzbände online nicht zugegriffen werden konnte.

[325] Die geläufige Abkürzung der IEEE International Symposium on Requirements Engineering lautet RE. Diese Abkürzung wird jedoch auch für den Ausdruck „Requirements Enginering“ allgemein und die Zeitschrift Requirements Engineering verwendet. Um Verwechslungen zu vermeiden, wird deshalb die Konferenz in dieser Arbeit mit ISRE abgekürzt.

Die in der Literaturanalyse berücksichtigten Konferenzbände wurden mit Ausnahme der ICIS alle von der IEEE Computer Society herausgegeben, wobei dies für die ICSE-Bände in Kooperation mit der ACM erfolgte.

Ablauf der Literaturrecherche

Der Ablauf der Suche nach relevanten Literaturquellen orientiert sich an den Empfehlungen von Webster und Watson:[327]

1. *Ausgangssuche (AS)*: Im ersten Schritt wurden im definierten Literaturspektrum mit den 16 Zeitschriften und 8 Konferenzen nach Literaturquellen gesucht, die die Auswahlkriterien erfüllen. Die Quellen, die den Kriterien genügen, sind die Ausgangstreffer.
2. *Rückwärtssuche (RS)*: Die Literaturverzeichnisse der Ausgangstreffer wurden nach weiteren Literaturquellen durchsucht, die die Auswahlkriterien erfüllen.
3. *Vorwärtssuche (VS)*: Sowohl über Web of Science[328] als auch Citeseer[329] wurde ermittelt, welche Literaturquellen die Ausgangstreffer zitieren. Jede so gefundene Literaturquelle wurde dahingehen überprüft, ob sie die Auswahlkriterien erfüllt.

In allen drei Suchschritten wurde die Zusammenfassung jeder Literaturquelle untersucht. Sofern anhand dieser Informationen nicht geklärt werden konnte, ob eines der Kriterien verletzt wird, wurden ausgewählte Abschnitte des Beitrags gelesen.

In der Rückwärts- und Vorwärtssuche können auch Literaturquellen gefunden werden, die sich in Zeitschriften oder Konferenzbänden befinden, die nicht in der Ausgangssuche berücksichtigt worden sind. Ebenso müssen entsprechende Quellen nicht im Zeitraum von 1990 bis 2005 liegen. Entscheidend sind für Treffer in der Rückwärts- und Vorwärtssuche ausschließlich die genannten Auswahlkriterien.

Sofern zu einem Beitrag in einem Konferenzband ein nahezu inhaltlich identischer Beitrag in einer wissenschaftlichen Zeitschriften erscheint, wird nur der Beitrag in der Zeitschrift berücksichtigt.

Die Literaturrecherche wurde im Januar und Februar 2006 durchgeführt.

---

[326] Ab 2002 wird für die Konferenz die Abkürzung RE verwendet. In dieser Arbeit wird die Abkürzung ICRE beibehalten. Vgl. hierzu auch die Fußnote 325.

[327] Vgl. Webster, Watson /Literature review/ xvi.

[328] Web of Science ist unter der URL http://scientific.thomson.com/products/wos/ zu finden (Stand 26.07.2006). Es ist ein kostenpflichtiges Produkt der Thomson Scientific.

[329] Die CiteSeer Scientific Literature Digital Library ist unter der URL http://citeseer.ist.psu.edu/ zu finden (Stand 26.07.2006). Die Eigendarstellung lautet: „CiteSeer is a scientific literature digital library and search engine that focuses primarily on the literature in computer and information science." Citeseer wurde ursprünglich durch das NEC Research Institute entwickelt. Die Nutzung ist kostenfrei.

4.3.2.3 Auswertung der Literaturquellen

Wie wurden die in der Literaturrecherche gefundenen empirischen Quellen ausgewertet? Ein konstituierendes Merkmal einer systematischen Literaturanalyse ist, dass für den festgelegten Ausschnitt der Literatur eine Synthese relevanter Aussagen erreicht wird. Im ersten Schritt wurden hierzu die identifizierten Literaturquellen nach Abschluss der Literaturrecherche mit dem Ziel gelesen, Beobachtungen zu Ursache-Wirkungszusammenhängen zu extrahieren. Um eine Synthese dieser Beobachtungen vorzubereiten und den explorativen Charakter der vorliegenden Literaturanalyse zu stützen, wurden zwei Mindmaps erstellt. Beide Mindmaps geben einen Überblick über die extrahierten Beobachtungen. Die erste Mindmap ist nach den Ursachen für Anforderungsfehler, die zweite nach den Wirkungen, also den Anforderungsfehlern, strukturiert. Immer wenn eine neue Beobachtung identifiziert worden ist, wurden beide Mindmaps erweitert.[330] Bei jeder Erweiterung wurden die Ursachen im ersten Mindmap inhaltlich zu Gruppen zusammengefasst. Im zweiten Mindmap erfolgt dies analog für Anforderungsfehler. Beide Mindmaps wurden im nächsten Schritt überarbeitet. Sie sind beide im Anhang B aufgeführt.

4.3.3 Ergebnisse der Literaturrecherche

4.3.3.1 Überblick über die identifizierten empirischen Quellen

In der Ausgangssuche wurden 17 Literaturquellen identifiziert, die die Auswahlkriterien erfüllen. Die Rückwärtssuche erweitert diese Menge um weitere 3 Literaturquellen, die Vorwärtssuche um eine Literaturquelle. Folglich liegen insgesamt 21 Treffer vor (vgl. Tabelle 4-4).[331]

---

[330] Vgl. zu dieser Vorgehensweise Webster, Watson /Literature review/ xvi f.

[331] Die Beiträge Damian u. a. /Case study/ (Q7), Damian u. a. /Requirements engineering/ (Q8) und Damian, Zowghi /Requirements engineering/ (Q9) wurden in den wissenschaftlichen Zeitschriften „Empirical Software Engieering“ oder „Requirements Engineering“ veröffentlicht. Zu diesen Beiträgen existieren die inhaltlich nahezu identischen Konferenzbeiträge Damian u. a. /Process improvement/, Damian u. a. /Downstream development/, Damian, Zowghi /Impact/. Die Konferenzbeiträge werden deshalb nicht berücksichtigt. Vgl. zu dieser Vorgehensweise Kapitel 4.3.2.2.

Nr.	Kurzzitat	gefunden während	Veröffent-lichungs-jahr	gefunden in
Q1	Adelson, Soloway /Domain experience/	RS	1985	IEEETSE
Q2	Al-Rawas, Easterbrook /Communications problems/	RS	1996	PASE[332]
Q3	Bhandari u. a. /Improvement/	AS	1994	IBMSJ
Q4	Briand u.a. /Change analysis process/	AS	1994	ICSM
Q5	Crowston, Kammerer /Coordination/	AS	1998	IBMSJ
Q6	Curtis, Krasner, Iscoe /Field study/	RS	1988	CACM
Q7	Damian u. a. /Case study/	AS	2004	ESE
Q8	Damian u. a. /Requirements engineering/	AS	2005	ESE
Q9	Damian, Zowghi / Requirements engineering/	AS	2003	RE
Q10	Daneva /ERP/	AS	2004	IEEESw
Q11	Ebert, De Man /Requirements uncertainty/	AS	2005	ICSE
Q12	Günther, Rombach, Ruhe /Qualitätsverbesserung/	AS	1996	WIRT
Q13	Hofmann, Lehner /Requirements engineering/	AS	2001	IEEESw
Q14	Kauppinen u. a. /Cultural change/	AS	2002	ICRE
Q15	Lauesen, Vinter /Requirement defects/	AS	2001	RE
Q16	Lubars, Potts, Richter /Requirements modeling/	AS	1993	ISRE
Q17	Morris, Masera, Wilikens /Requirements Engineering/	AS	1998	ICRE
Q18	Nakajo, Kume /Case history analysis/	AS	1991	IEEETSE
Q19	Sutcliffe, Economou, Markis /Requirements errors/	AS	1999	RE
Q20	Takahashi u. a. /Methodologies/	VS	1995	JSS[333]
Q21	Walz, Elam, Curtis /Softtware design team/	AS	1993	CACM

Tabelle 4-4: In der Literaturrecherche gefundene Quellen

Die Tabelle 4-4 führt neben einer identifizierenden Nummer,

- ein Kurzzitat des Beitrags,
- die Angabe, in welchen Schritt der Literaturrecherche der Beitrag gefunden wurde (AS = Ausgangssuche, RS = Rückwärtssuche, VS = Vorwärtssuche),
- das Jahr der Veröffentlichung und
- die Abkürzung der Zeitschrift oder des Konferenzbandes, in der oder dem der Beitrag veröffentlicht worden ist.[334]

[332] Im Rahmen der Rückwärtssuche wurde die Literaturquelle Al-Rawas, Easterbrook /Communication problems/ im Konferenzband zur Konferenz „First Westminster Conference on Professional Awareness in Software Engineering (PASE), 1-2 February 1996, London, England" gefunden. Der Konferenzband gehört nicht zu den in der Ausgangssuche der systematischen Literaturanalyse berücksichtigten Konferenzbänden.

[333] Im Rahmen der Vorwärtssuche wurde die Literaturquelle Takahashi u. a. /Methodologies/ in der Zeitschrift Journal of Systems and Software (JSS) gefunden. Die Zeitschrift gehört nicht zu den in der Ausgangssuche berücksichtigten Zeitschriften.

[334] Vgl. Tabelle 4-1 und Tabelle 4-3.

Die Tabelle 4-5 zeigt auf, in welchem Jahr die 21 Literaturquellen jeweils veröffentlicht worden sind. Auffällige Häufigkeiten in bestimmten Jahren sind nicht festzustellen. Vielmehr verteilen sich die Literaturquellen auf den Zeitraum von 1985 bis 2005.

Veröffentlichungsjahr	Absolute Häufigkeit Literaturquellen	Relative Häufigkeit Literaturquellen
1985	1	4,76%
1986	0	0,00%
1987	0	0,00%
1988	1	4,76%
1989	0	0,00%
1990	0	0,00%
1991	1	4,76%
1992	0	0,00%
1993	2	9,52%
1994	2	9,52%
1995	1	4,76%
1996	2	9,52%
1997	0	0,00%
1998	2	9,52%
1999	1	4,76%
2000	0	0,00%
2001	2	9,52%
2002	1	4,76%
2003	1	4,76%
2004	2	9,52%
2005	2	9,52%
	Summe: 21	Summe: 100%

Tabelle 4-5 : Veröffentlichungsjahre der Literaturquellen

13 Beiträge (62%) wurden im Zeitraum von 1985 bis 1999 veröffentlicht. Die restlichen 8 Beiträge (38%) fallen auf den kürzeren Zeitraum von 2000 bis 2005. Folglich ist ein Trend zu erkennen, dass in aktuelleren Beiträgen ab 2000 mehr Ursache-Wirkungszusammenhänge zwischen Prozessmängeln der Anforderungsanalyse und Anforderungsfehlern behandelt werden. Trotzdem ist jedoch festzuhalten, dass diese Ursache-Wirkungszusammenhänge noch verhältnismäßig wenig empirisch untersucht worden sind. Dies überrascht angesichts der Tatsache, dass die Anforderungsanalyse seit Mitte der siebziger Jahre als eine eigenständige Teilaufgabe der Softwareentwicklung wahrgenommen wird, ganze Zeitschriften (z. B. RE) und Konferenzen (z. B. ICRE) sich allein diesem Thema widmen und es zahlreiche Veröffent-

lichungen gibt. Dieser Umstand wird noch deutlicher, wenn berücksichtigt wird, dass von den gefundenen 21 empirischen Quellen nur 2 explizit die Ursachen von Fehlern untersuchen.[335]

Unter den 21 Treffern sind 15 Beiträge aus Zeitschriften (vgl. Tabelle 4-6) und 6 aus Konferenzen (vgl. Tabelle 4-7). Die Beiträge aus wissenschaftlichen Zeitschriften kommen unter anderem aus jeweils einer Zeitschrift mit Rang A und B sowie aus 4 Zeitschriften mit Rang C. Insgesamt 6 Beiträge kommen aus Zeitschriften oder Konferenzen, die sich schwerpunktmäßig mit der Anforderungsanalyse beschäftigen: RE (3), ICRE (2) und ISRE (1).

Zeitschrift	Rang[336]	Absolute Häufigkeit Literaturquellen	Relative Häufigkeit Literaturquellen
RE	-	3	20,00%
CACM	A	2	13,33%
ESE	-	2	13,33%
IBMSJ	C	2	13,33%
IEEESw	C	2	13,33%
IEEETSE	B	2	13,33%
JSS	C	1	6,67%
WIRT	C	1	6,67%
CAIS	B	0	0%
EJIS	B	0	0%
ISJ	C	0	0%
ISR	A	0	0%
JMIS	A	0	0%
JAIS	C	0	0%
MS	A	0	0%
MISQ	A	0	0%
SQJ	-	0	0%
		Summe: 15	Summe: 100%

Tabelle 4-6: Verteilung der Literaturquellen auf Zeitschriften

[335] Hierbei handelt es sich um die Beiträge Nakajo, Kume /Case history analysis/ (Q18) und Sutcliffe, Economou, Markis /Requirements errors/ (Q19).

[336] Vgl. Tabelle 4-1.

Konferenz	Absolute Häufigkeit Literaturquellen	Relative Häufigkeit Literaturquellen
ICRE	2	33,33%
ICSE	1	16,67%
ISRE	1	16,67%
ICSM	1	16,67%
PASE	1	16,67%
ICIS	0	0%
ISESE	0	0%
ISSRE	0	0%
METRICS	0	0%
	Summe: 6	Summe: 100%

Tabelle 4-7: Verteilung der Literaturquellen auf Konferenzen

4.3.3.2 Darstellung der empirischen Quellen

In der systematischen Literaturanalyse werden 21 empirische Quellen berücksichtigt. Um dem Leser die Möglichkeit zu geben, sich einen schnellen Überblick über diese einzelnen Quellen verschaffen zu können, werden sie im Anhang A.1 anhand einheitlicher Beschreibungskriterien zusammengefasst. Die Beschreibungskriterien werden nachfolgend eingeführt.

Untersuchungseinheit und Ziele der Untersuchung

Sofern ein Beitrag implizit oder explizit die *Ziele der empirischen Untersuchung* beinhaltet, werden diese in Form von möglichst präzisen Fragen wiedergegeben. Diese Form der Darstellung muss dabei nicht der in dem jeweiligen Beitrag entsprechen. Sind die genauen Ziele der Untersuchung aus einem Beitrag nicht ersichtlich, wird darauf entsprechend hingewiesen. [337] Je präziser die Ziele der Untersuchung in einem Beitrag offen gelegt werden, desto präziser wird auch die *Untersuchungseinheit* abgegrenzt, über die neue Kenntnisse gewonnen werden sollen.

Vorgehensweise der Untersuchung

Die grundlegende Vorgehensweise der Untersuchung wird beschrieben, falls dies für das Verständnis der Ergebnisse der Untersuchung erforderlich ist. Sofern der Beitrag *explizit* aufführt,

[337] Von der Darstellung der Untersuchungsziele in Form von Fragen wird abgewichen, wenn aus der jeweiligen Quelle nicht ersichtlich wird, welche Fragen im Einzelnen die Untersuchung beantworten soll. Dies ist insbesondere bei deskriptiven Untersuchungen der Fall. Welche Merkmale deskriptive Untersuchungen aufweisen, wird in Kapitel 4.3.3.3.1 beschrieben.

wie die Daten der Untersuchung erhoben und ausgewertet worden sind, werden entsprechende Informationen kurz aufgeführt.

Subjekte bzw. Objekte der Untersuchung

Unter diesen Punkt wird kurz beschrieben,

- welche *Subjekte* im Rahmen der Untersuchung z. B. befragt oder beobachtet worden sind oder
- welche *Objekte* (z. B. welche Software) untersucht worden sind.

Alle in dem jeweiligen Beitrag für relevant erachteten Merkmale der Subjekte (z. B. Erfahrung, Aufgabenbereich) oder der Objekte (z. B. Umfang) werden aufgeführt. Die Subjekte oder Objekte der Untersuchung sind nicht zu verwechseln mit der Untersuchungseinheit. Wird eine Untersuchung wiederholt, bleibt die Untersuchungseinheit bestehen, während Subjekte und Objekte der Untersuchung ersetzt werden können.

Wesentliche Ergebnisse der Untersuchung

Die wesentlichen Ergebnisse der Untersuchung werden kurz zusammengefasst.[338]

4.3.3.3 Bewertung der empirischen Quellen

4.3.3.3.1 Darstellung der Bewertungskriterien

Empirische Untersuchungen im Bereich der Softwareentwicklung haben häufig Unzulänglichkeiten.[339] Es ist realitätsfern, eine perfekte empirische Untersuchung zu erwarten.[340] Bei einer empirischen Untersuchung erfolgt eine Konfrontation mit der Realität, weshalb vorab nicht alle Eventualitäten berücksichtigt werden können. Jedoch ist es entscheidend zu klären, ob und inwiefern etwaige Unzulänglichkeiten einer empirischen Untersuchung dazu führen, dass bestimmte Ergebnisse der Untersuchung hinsichtlich ihrer Aussagefähigkeit eingeschränkt oder gänzlich in Frage gestellt werden müssen.

Insbesondere aus Sicht einer systematischen Literaturanalyse ist es unzureichend, ausschließlich auf Unzulänglichkeiten der berücksichtigten Literaturquellen hinzuweisen.[341] Stattdessen sind neben den Schwächen einer Literaturquelle auch ihr Beitrag für die Forschung und Praxis zu würdigen.

---

[338] Die Darstellung der Ergebnisse der Untersuchung beschränkt sich dabei nicht auf die, die für die systematische Literaturanalyse dieser Arbeit relevant sind.

[339] Vgl. Fenton, Pfleeger, Glass /Science and Substance/ 87-93 und Perry, Porter, Votta /Empirical studies/ 349.

[340] Vgl. Tichy /Empirical work/ 309.

[341] Vgl. Webster, Watson /Literature review/ xviii, Eisenberg /Literature review/ 25 f. und Bem /Review article/ 175.

Nachdem in Kapitel 4.3.3.2 die in der Literaturrecherche berücksichtigten empirischen Quellen dargestellt worden sind, werden sie im vorliegenden Kapitel anhand ausgewählter Kriterien bewertet. Es besteht hierbei nicht der Anspruch, die Literaturquellen inhaltlich und forschungsmethodisch eingehend zu bewerten. Vielmehr werden die Literaturquellen dahingehend bewertet, inwieweit sie überhaupt grundsätzliche Voraussetzungen erfüllen, um inhaltlich und forschungsmethodisch bewertet werden zu können. Dies schließt zum Beispiel Fragen dahingehend ein, ob offen gelegt wird, welche Methoden in der empirischen Untersuchung eingesetzt worden sind, um Daten zu erheben und auszuwerten. Die Bewertungskriterien sind an Empfehlungen angelehnt, die Kitchenham u. a. für empirische Untersuchungen vorschlagen.[342]

Die Bewertungskriterien können in drei Kategorien gruppiert werden:

- Kriterien zum Gegenstand der Untersuchung,
- Kriterien zum Kontext der Untersuchung und
- Kriterien zur Durchführung der Untersuchung.

Kriterien zum Gegenstand der Untersuchung

Zum Gegenstand einer empirischen Untersuchung werden vier Bewertungskriterien berücksichtigt. Sie erleichtern die Interpretation der Ergebnisse einer empirischen Untersuchung. Im Einzelnen lauten sie:

- *Explizite Formulierung der Untersuchungsziele*

  Werden die Ziele der empirischen Untersuchung explizit formuliert? Sofern diese Frage für eine Quelle bejaht werden kann, erhält sie für dieses Kriterium die Ausprägung „ja“, ansonsten „nein“.
- *Typ der Untersuchung gemäß Eigendarstellung der empirischen Quelle*

  Für jede Quelle wird festgehalten, um welchen Typ einer empirischen Untersuchung es sich gemäß der Eigendarstellung der Autoren handelt. Mögliche Ausprägungen sind z. B. Fallstudie oder Experiment. Sofern in der Quelle der Typ der Untersuchung nicht genannt wird, steht in der Tabelle 4-8 oder Tabelle 4-9 die Ausprägung „keine Angabe“.

[342] Vgl.Kitchenham u. a. /Guidelines/ 723-732.

- *Typ der Untersuchung in Anlehnung an die Typologie von Zelkowitz und Wallace*
  Für jede Quelle wird der Typ der Untersuchung angelehnt an der Typologie von Zelkowitz und Wallace[343] ermittelt und damit der Eigendarstellung der Autoren gegenüber gestellt.[344] Der Typ einer Untersuchung liefert Hinweise darüber, in welchem Maße die Erkenntnisse der Untersuchung als stichhaltig betrachtet werden können.[345] Die Typologie von Untersuchungen wird nach dem letzten Kriterium beschrieben.
- *Zweck der Untersuchung*
  Empirische Untersuchungen können drei verschiedene Zwecke verfolgen.[346] *Explorative Untersuchungen* versuchen Zusammenhänge in wenig verstandenen Situationen erstmalig zu erkennen. Sie eignen sich insbesondere dazu, um neue Hypothesen zu formulieren, die in zukünftigen Untersuchungen geprüft werden können. *Deskriptive Untersuchungen* versuchen auf der Grundlage eines bestehenden umfangreichen Wissens in einem Bereich, ein Objekt des Interesses (z. B. Personen, Ereignisse oder Zustände) möglichst präzise und ausgiebig zu beschreiben. *Explanatorische Untersuchungen* haben das Ziel, ausgewählte Ausschnitte der Realität zu erklären. Dies geschieht häufig, indem Hypothesen zu Ursache-Wirkungszusammenhängen geprüft werden.

Kriterien zum Kontext der Untersuchung

Drei Bewertungskriterien beziehen sich auf den Kontext der Untersuchung.[347] Sofern die Frage eines Kriteriums für eine Quelle bejaht werden kann, erhält die Quelle für dieses Kriterium die Ausprägung „ja“, ansonsten „nein“.

- *Rahmenbedingungen der Untersuchungen*
  Werden die Rahmenbedingungen der empirischen Untersuchung genannt, um die Ergebnisse der Untersuchung besser zu verstehen zu können? Mögliche Rahmenbedingungen sind z. B. Erfahrungen der befragten Personen, Art der entwickelten Software etc.

---

[343] Vgl. Zelkowitz, Wallace /Models/ 25-29.

[344] Zelkowitz und Wallace stellen in ihrer umfassenden Literaturanalyse fest, dass Autoren empirischer Untersuchungen dazu neigen, den Typ ihrer Untersuchung falsch zu klassifizieren. Vgl. Zelkowitz, Wallace /Models/ 30.

[345] Vgl. Zelkowitz, Wallace /Models/ 23, 31 und Kitchenham u. a. /Guidelines/ 731.

[346] Vgl. zu diesem Absatz Robson /Research/ 59 f.

[347] Vgl.zu Folgendem Kitchenham u. a. /Guidelines/ 723-725.

- *Hypothesen*
  Werden bei einer explanatorischen Untersuchung die Hypothesen, die geprüft werden, theoretisch begründet?
- *Vergleich mit verwandten Forschungsergebnissen*
  Werden die Ergebnisse der Untersuchung mit verwandten Forschungsergebnissen verglichen?

Kriterien zur Durchführung der Untersuchung

Sechs Bewertungskriterien beziehen sich auf die Durchführung der Untersuchung. Sofern nichts anderes angegeben, gibt es zu einem Kriterium die Ausprägungen „ja“ oder „nein“, je nachdem ob die Frage eines Kriteriums für eine Quelle bejaht werden kann oder nicht.

- *Auswahl der Untersuchungssubjekte bzw. –objekte*[348]
  Wird in der Quelle beschrieben, wie die in der Untersuchung berücksichtigten Subjekte oder Objekte ausgewählt worden sind?
- *Definition der Untersuchungseinheit*[349]
  Wird die Untersuchungseinheit definiert?[350]
- *Datenerhebung*
  Wird in der Quelle beschrieben, wie die Daten der Untersuchung erhoben worden sind?
- *Art der erhobenen Daten*[351]
  Werden quantitative, qualitative oder sowohl quantitative als auch qualitative Daten erhoben und ausgewertet? Entsprechend sind die Ausprägungen dieses Kriteriums jeweils „quantitativ“, „qualitativ“ oder „quantitativ/qualitativ“.
- *Datenauswertung*
  Wird in der Quelle beschrieben, wie die Daten der Untersuchung ausgewertet worden sind?
- *Einschränkungen der Untersuchung*[352]
  Werden Einschränkungen der Untersuchung, die Auswirkungen auf die Interpretation der Ergebnisse haben können, in der Quelle offen gelegt?

---

[348] Vgl. Kitchenham u. a. /Guidelines/ 725.
[349] Vgl. Kitchenham u. a. /Guidelines/ 726.
[350] Vgl. hierzu auch Kapitel 4.3.3.2.
[351] Vgl. Freimut u. a. /Empirical Studies/ 15-17.
[352] Vgl. Kitchenham u. a. /Guidelines/ 732.

Typologie von empirischen Untersuchungen

Zelkowitz und Wallace schlagen eine Typologie von Untersuchungen vor und evaluieren diese Typologie anhand einer umfassenden Literaturanalyse, bei der 612 wissenschaftliche Artikel analysiert werden.[353] Die nachfolgende Typologie ist an dem Vorschlag von Zelkowitz und Wallace angelehnt. Es werden ausschließlich die Typen vorgestellt, die für die Bewertung der empirischen Literaturquellen in Kapitel 4.3.3.3.2 und 5.2.2.3 relevant sind.

Die Typen lauten im Einzelnen:

- *Fallstudie*: Eine Fallstudie untersucht ein aktuelles Phänomen innerhalb seines realen Kontextes.[354] Sie eignet sich insbesondere dann, wenn die Grenzen zwischen dem Phänomen und seinem Kontext nicht direkt einsichtig sind. Ein Forscher kann den Ablauf der Untersuchung beeinflussen.
- *Laborexperiment*: Bei einem Laborexperiment werden bestimmte Variablen in einer künstlich geschaffenen Wirklichkeit untersucht.[355] Unabhängige Variablen werden aktiv durch die beteiligten Forscher verändert, um ihre Wirkung auf die abhängigen Variablen zu bestimmen.
- *Feldexperiment*: Bei einem Feldexperiment werden Variablen in ihrer natürlichen Umgebung untersucht.[356] Unabhängige Variablen werden wie beim Laborexperiment durch die beteiligten Forscher verändert, um ihre Wirkung auf die abhängigen Variablen zu bestimmen.
- *Beobachtungsstudie*: Bei diesem Typ einer Untersuchung werden Objekte oder Subjekte in ihrer natürlichen Umgebung untersucht. Dabei bedient man sich der Beobachtung als Erhebungstechnik.
- *Befragungsstudie*: Eine Befragungsstudie ist eine Untersuchung, bei der durch Fragebogen oder Interviews die Meinungen einer Menge von Personen aus einer Grundgesamtheit erhoben werden. [357]
- *Gruppendiskussion*: Bei einer Gruppendiskussion werden die Daten durch die Interaktionen der Gruppenmitglieder gewonnen, wobei das Diskussionsthema durch das Interesse des Forschers bestimmt wird.[358]

---

[353] Vgl. zu diesem Absatz Zelkowitz, Wallace /Models/ 25-30.

[354] Vgl. hierzu und zum folgenden Satz Yin /Case study research/ 13 f. und Yin /Case study crisis/ 58.

[355] Vgl. zu diesem Absatz Heinrich /Wirtschaftsinformatik/ 99-101. Dieser Typ entspricht im Wesentlichen dem Typ „synthetic environment experiments“. Vgl. hierzu Zelkowitz, Wallace /Models/ 27 f.

[356] Vgl. zu diesem Absatz Heinrich / Wirtschaftsinformatik / 99-101. Dieser Typ entspricht im Wesentlichen dem Typ „replicated experiment.“ Vgl. hierzu Zelkowitz, Wallace /Models/ 27.

[357] Vgl. Freimut u. a. /Empirical Studies/ 55, 59-62.

[358] Vgl. Lamnek /Gruppendiskussion/ 27.

- *Ex-post-Analyse*: Der Forscher beschränkt sich auf die nachträgliche Auswertung von vorhandenen quantitativen oder qualitativen Daten von abgeschlossenen Softwareentwicklungsprojekten.[359] D. h., dass im Unterschied zu den anderen Untersuchungstypen bei der Ex-post-Analyse *keine* neuen Daten im Rahmen der Untersuchung erhoben werden. Es bleibt zusätzlich festzuhalten, dass die vorhandenen Daten ursprünglich nicht zum Zweck der empirischen Untersuchung erhoben worden sind, sondern Ergebnisse von Softwareentwicklungsprojekten darstellen. Entsprechende Ergebnisse sind z. B. Softwarecode, Projektabschlussberichte („lessons learned"), beschriebene Softwarefehler, Dokumente zur Anforderungsanalyse, zum Entwurf oder zum Test.

4.3.3.3.2 Anwendung der Bewertungskriterien

Die in Kapitel 4.3.3.3.1 definierten Bewertungskriterien wurden auf die in der Literaturanalyse berücksichtigten Quellen angewandt. Die Tabelle 4-8 und Tabelle 4-9 geben das Ergebnis wieder. In beiden Tabellen steht „j" für „ja" und „n" für „nein".

	Gegenstand der Untersuchung				**Kontext der Untersuchung**			**Durchführung der Untersuchung**					
Nr. der empirischen Quelle	**explizite Formulierung der Untersuchungsziele**	**Typ der Untersuchung nach Eigendarstellung**	**Typ der Untersuchung nach Typologie dieser Arbeit**	**Zweck der Untersuchung**	**Rahmenbedingungen**	**Hypothesen**	**Vergleich mit verwandten Forschungsergebnissen**	**Auswahl der Untersuchungssubjekte bzw. –objekte**	**Definition der Untersuchungseinheit**	**Datenerhebung**	**Art der erhobenen Daten**	**Datenauswertung**	**Einschränkungen der Untersuchung**
Q1	j	Experiment	Labor-experiment	explorativ	j	-	n	n	j	j	qualitativ	n	n
Q2	j	Feldstudie	Befragung	explorativ	j	-	j	j	j	j	quantitativ, qualitativ	n	n
Q3	j	keine Angabe	Ex-post-Analyse	deskriptiv	j	-	n	n	j	j	quantitativ, qualitativ	j	n
Q4	j	Fallstudie	Fallstudie	deskriptiv	j	-	n	n	j	n	qualitativ	n	j
Q5	j	keine Angabe	Fallstudie	explanato-risch	j	n	n	j	j	j	qualitativ	j	j
Q6	n	Feldstudie	Befragung	explorativ	j	-	j	j	j	j	qualitativ	j	j
Q7	j	Fallstudie	Fallstudie	deskriptiv	j	-	n	j	j	j	quantitativ, qualitativ	n	j
Q8	j	Fallstudie	Fallstudie	deskriptiv	j	-	n	j	j	j	quantitativ, qualitativ	n	j

Tabelle 4-8: Bewertung der empirischen Literaturquellen

[359] Vgl. hierzu die Typen „legacy data" und „lessons learned" in Zelkowitz, Wallace /Models/ 26 f.

	Gegenstand der Untersuchung				Kontext der Untersuchung			Durchführung der Untersuchung					
Nr. der empirischen Quelle	**explizite Formulierung der Untersuchungsziele**	**Typ der Untersuchung nach Eigendarstellung**	**Typ der Untersuchung nach Typologie dieser Arbeit**	**Zweck der Untersuchung**	**Rahmenbedingungen**	**Hypothesen**	**Vergleich mit verwandten Forschungsergebnissen**	**Auswahl der Untersuchungssubjekte bzw. –objekte**	**Definition der Untersuchungseinheit**	**Datenerhebung**	**Art der erhobenen Daten**	**Datenauswertung**	**Einschränkungen der Untersuchung**
Q9	j	Fallstudie	Fallstudie	explorativ	j	-	j	j	j	j	qualitativ	j	j
Q10	j	gelernte Lektionen	Ex-post-Analyse	deskriptiv	j	-	n	j	j	n	qualitativ	n	n
Q11	j	Feldstudie	Befragung, Ex-post-Analyse[360]	explorativ, explanatorisch	n	j	j	j	j	j	quantitativ, qualitativ	j	n
Q12	n	keine Angabe	Ex-post-Analyse	deskriptiv	j	-	n	n	n	n	quantitativ, qualitativ	n	n
Q13	j	Feldstudie	Befragung	explanatorisch	j	n	j	j	j	j	quantitativ, qualitativ	n	n
Q14	j	gelernte Lektionen	Fallstudie	explorativ	j	-	j	n	n	j	qualitativ	n	n
Q15	n	Experiment	Fallstudie	deskriptiv	j	-	n	n	n	j	quantitativ, qualitativ	j	n
Q16	n	Feldstudie	Befragung	explorativ	j	-	j	n	n	j	qualitativ	n	n
Q17	j	Arbeitskreis	Gruppendiskussion	explorativ	n	-	n	j	j	j	qualitativ	n	n
Q18	n	keine Angabe	Fallstudie	deskriptiv	n	-	n	n	n	j	quantitativ, qualitativ	j	j
Q19	j	Fallstudie	Fallstudie	explorativ	j	-	j	n	n	j	qualitativ	n	j
Q20	j	Experiment	Laborexperiment	explanatorisch	j	n	n	j	j	j	quantitativ	n	n
Q21	j	keine Angabe	Beobachtung	explorativ	j	-	n	n	j	j	quantitativ, qualitativ	j	j

Tabelle 4-9: Bewertung der empirischen Literaturquellen (Fortsetzung)

Die Literaturquellen werden in der Tabelle 4-8 und Tabelle 4-9 jeweils anhand einer eindeutigen Nummer identifiziert, welche in der Tabelle 4-4 bereits aufgeführt wurde. Um ein schnelles Auffinden der Quelle zu ermöglichen, werden die eindeutigen Nummern und die dazugehörigen Kurzzitate der empirischen Literaturquellen in der Tabelle 4-10 wiederholt.

[360] Der Beitrag Ebert, De Man /Requirements uncertainty/ (Q11) umfasst zwei Untersuchungen, die unterschiedlichen Typen zugeordnet werden können.

Nr.	Kurzzitat
Q1	Adelson, Soloway /Domain experience/
Q2	Al-Rawas, Easterbrook /Communications problems/
Q3	Bhandari u. a. /Improvement/
Q4	Briand u.a. /Change analysis process/
Q5	Crowston, Kammerer /Coordination/
Q6	Curtis, Krasner, Iscoe /Field study/
Q7	Damian u. a. /Case study/
Q8	Damian u. a. /Requirements engineering/
Q9	Damian, Zowghi / Requirements engineering/
Q10	Daneva /ERP/
Q11	Ebert, De Man /Requirements uncertainty/

Nr.	Kurzzitat
Q12	Günther, Rombach, Ruhe /Qualitätsverbesserung/
Q13	Hofmann, Lehner /Requirements engineering/
Q14	Kauppinen u. a. /Cultural change/
Q15	Lauesen, Vinter /Requirement defects/
Q16	Lubars, Potts, Richter /Requirements modeling/
Q17	Morris, Masera, Wilikens /Requirements Engineering/
Q18	Nakajo, Kume /Case history analysis/
Q19	Sutcliffe, Economou, Markis /Requirements errors/
Q20	Takahashi u. a. /Methodologies/
Q21	Walz, Elam, Curtis /Softtware design team/

Tabelle 4-10: Eindeutige Nummern und Kurzzitate der Literaturquellen

Die Bewertung der 21 empirischen Literaturquellen führt zu folgenden Ergebnissen:

- Der überwiegende Anteil der Literaturquellen verfolgt einen explorativen (ca. 45%) oder deskriptiven (ca. 36%) Zweck.[361] Folglich versuchen diese Untersuchungen neue Zusammenhänge zu erkennen oder durch die präzise Beschreibung einer bekannten Untersuchungseinheit neue Erkenntnisse zu gewinnen. Nur ca. 18% der Untersuchungen versuchen ausgewählte Ausschnitte der Realität zu erklären.
- Als Untersuchungstyp wird die Fallstudie bevorzugt (ca. 41%). An zweiter Stelle kommt die Befragungsstudie (ca. 23%) und an dritter die Ex-post-Analyse (ca. 18%).
- Ein großer Anteil der Untersuchungen erheben nur qualitative (ca. 48%) oder sowohl qualitative als auch quantitative Daten (ca. 48%).

4.3.4 Ergebnisse der Auswertung der empirischen Quellen

4.3.4.1 Überblick über die Ergebnisse

In der Literaturanalyse wurden 21 empirische Literaturquellen ausgewertet.[362] Diese Quellen tragen zu insgesamt 92 relevanten Beobachtungen bei.[363] Eine *Beobachtung* ist dabei ein Ur-

[361] Die Quelle Ebert, De Man /Requirements uncertainty/ (Q11) beschreibt zwei Untersuchungen, die einen unterschiedlichen Zweck verfolgen und unterschiedlichen Typs sind. Bei der Auswertung des Typs und des Zwecks der Untersuchungen wird diese Quelle deshalb zweimal berücksichtigt.

[362] In Kapitel 4.3.2.3 wurde bereits erläutert, wie die Quellen ausgewertet worden sind.

[363] Die Quellen Damian u. a. /Case study/ (Q7) und Damian u. a. /Requirements engineering/ (Q8) sind zwei aufeinander aufbauende Untersuchungen und beschreiben beide unter anderem eine identische Beobachtung. Diese Beobachtung wurde nur einmal berücksichtigt und Damian u. a. /Requirements engineering/ (Q8) zugeordnet.

sache-Wirkungszusammenhang mit folgender Struktur: eine Ursache, der man im Prozess der Anforderungsanalyse entgegenwirken kann, hat einen bestimmten Typ von Anforderungsfehlern oder generell Anforderungsfehler zur Folge. Sofern eine Ursache verschiedene Typen von Anforderungsfehlern bewirkt, wird für jeden Typ eine einzelne Beobachtung erfasst.
Die Tabelle 4-11 gibt einen Überblick darüber, wie sich die 92 Beobachtungen auf die 21 empirischen Quellen verteilen.

Nr.	Kurztitel	Absolute Häufigkeit Beobachtungen	Relative Häufigkeit Beobachtungen
Q1	Adelson, Soloway /Domain experience/	3	3,26%
Q2	Al-Rawas, Easterbrook /Communications problems/	5	5,43%
Q3	Bhandari u. a. /Improvement/	3	3,26%
Q4	Briand u.a. /Change analysis process/	7	7,61%
Q5	Crowston, Kammerer /Coordination/	9	9,78%
Q6	Curtis, Krasner, Iscoe /Field study/	11	11,96%
Q7	Damian u. a. /Case study/	2	2,17%
Q8	Damian u. a. /Requirements engineering/	1	1,09%
Q9	Damian, Zowghi / Requirements engineering/	7	7,61%
Q10	Daneva /ERP/	2	2,17%
Q11	Ebert, De Man /Requirements uncertainty/	5	5,43%
Q12	Günther, Rombach, Ruhe /Qualitätsverbesserung/	4	4,35%
Q13	Hofmann, Lehner /Requirements engineering/	1	1,09%
Q14	Kauppinen u. a. /Cultural change/	7	7,61%
Q15	Lauesen, Vinter /Requirement defects/	4	4,35%
Q16	Lubars, Potts, Richter /Requirements modeling/	4	4,35%
Q17	Morris, Masera, Wilikens /Requirements Engineering/	2	2,17%
Q18	Nakajo, Kume /Case history analysis/	2	2,17%
Q19	Sutcliffe, Economou, Markis /Requirements errors/	8	8,70%
Q20	Takahashi u. a. /Methodologies/	1	1,09%
Q21	Walz, Elam, Curtis /Softtware design team/	4	4,35%
		Summe: 92	Summe: 100%

Tabelle 4-11 : Verteilung der Beobachtungen auf die empirischen Literaturquellen.

Die drei Untersuchungen, aus denen die meisten Beobachtungen extrahiert werden können, sind mit 11 Beobachtungen die Feldstudie von Curtis, Krasner und Iscoe (Q6), gefolgt von der Fallstudie von Crowston und Kammerer (Q5) mit 9 Beobachtungen. Die dritte Untersuchung ist die Fallstudie von Sutcliffe, Economou und Markis, die sich als einzige Untersuchung unter den 21 explizit damit beschäftigt, die Ursachen von Anforderungsfehlern aufzuzeigen.

Die Tabelle 4-12 zeigt wie viele Quellen jeweils zu wie vielen Beobachtungen beisteuern. Die meisten Quellen enthalten 1 bis 5 Beobachtungen. Durchschnittlich trägt eine Quelle zu 4,38 Beobachtungen bei.

Anzahl Beobachtungen	Anzahl Quellen mit der jeweiligen Anzahl von Beobachtungen
1	3
2	4
3	2
4	4
5	2
6	0
7	3
8	1
9	1
10	0
11	1
	Summe: 21

Tabelle 4-12: Anzahl der Quellen mit einer bestimmten Anzahl von Beobachtungen

Im Kapitel 4.3.4.2 wird eine Typologie von Anforderungsfehlern dargestellt, die ein Ergebnis der explorativen Literaturanalyse ist. Daran schließt in Kapitel 4.3.4.3 eine Liste von Ursachen an, die gemäß des ausgewerteten Literaturausschnitts maßgeblich zu Anforderungsfehlern beiträgt.

4.3.4.2 Typologie von Anforderungsfehlern

Darstellung der Typologie

Ein *Anforderungsfehler* ist ein unzureichendes Merkmal oder ein erwartetes, jedoch fehlendes Merkmal eines Arbeitsergebnisses der Anforderungsanalyse, sofern es eine Änderung in diesem Ergebnis notwendig macht.[364] Aus der Auswertung von 92 Beobachtungen zu Ursache-Wirkungszusammenhängen kann eine Typologie von Anforderungsfehlern hergeleitet werden, die nachfolgend dargestellt wird.[365]

Verschiedene Typen eines Anforderungsfehlers können anhand eines hierarchischen Baumdiagramms wie in der Abbildung 4-4 unterschieden werden.

[364] Vgl. hierzu auch Kapitel 4.1.

[365] Vgl. zur Vorgehensweise Kapitel 4.3.2.3

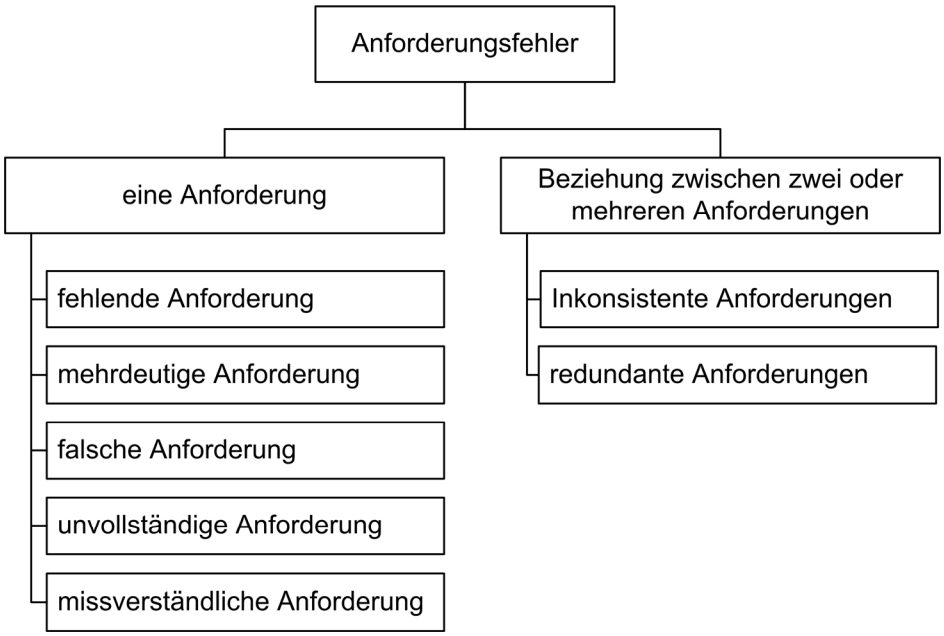


Abbildung 4-4: Typologie von Anforderungsfehler

Wenn sich ein Anforderungsfehler auf eine Anforderung bezieht, können insgesamt fünf Fälle voneinander abgegrenzt werden:

- Für ein bestehendes Kundenbedürfnis wurde eine Anforderung gänzlich übersehen (*fehlende Anforderung*).
- Die Beschreibung der Anforderung kann auf mehrere Arten gedeutet werden (*mehrdeutige Anforderung*).
- Die Anforderung erfüllt kein Kundenbedürfnis oder erfüllt sie nicht korrekt (*falsche Anforderung*).[366]
- Die Anforderung ist unvollständig beschrieben (*unvollständige Anforderung*).
- Die Anforderung ist zwar vollständig beschrieben, jedoch fehlen Kontextinformationen zu dieser Anforderung oder Verknüpfungen zu vorhandenen Kontextinformationen, um die Anforderung zu verstehen (*missverständliche Anforderung*). Kontextinformationen sind z. B.
  - Kundenbedürfnisse, die eine Anforderung begründen,
  - für die Anforderung relevante Kundengruppen oder
  - andere abhängige Anforderungen.

Ein Anforderungsfehler kann sich auch auf eine Beziehung zwischen zwei oder mehreren Anforderungen beziehen. Zwei Fälle können dabei unterschieden werden:

- Eine Anforderung steht im Widerspruch zu einer oder mehreren anderen Anforderungen (*inkonsistente Anforderungen*).

[366] Vgl. hierzu Kapitel 2.2.1.

- Eine Anforderung ist redundant zu mindestens einer anderen Anforderung (*redundante Anforderungen*).

Die Tabelle 4-13 führt auf, welchem Typ von Anforderungsfehler die Wirkungen in den 92 Beobachtungen zugeordnet werden können.

Typ des Anforderungsfehlers	Absolute Häufigkeit Beobachtungen	Relative Häufigkeit Beobachtungen
Anforderungsfehler allgemein	39	42,39%
missverständliche Anforderung	15	16,30%
inkonsistente Anforderungen	11	11,96%
fehlende Anforderung	10	10,87%
falsche Anforderung	6	6,52%
unvollständige Anforderung	5	5,43%
mehrdeutige Anforderung	4	4,35%
redundante Anforderungen	2	2,17%
	Summe: 92	Summe: 100%

Tabelle 4-13: Verteilung der Beobachtungen nach Typen von Anforderungsfehlern

In 39 Beobachtungen (ca. 42%) werden als Wirkung Anforderungsfehler allgemein beschrieben ohne einen Typ zu konkretisieren.[367] Folglich ist der Typ des Anforderungsfehlers für 53 Beobachtungen ermittelbar. Darunter sind missverständliche, inkonsistente und fehlende Anforderungen am häufigsten vertreten.

Die aus der systematischen Literaturanalyse entwickelte Typologie von Anforderungsfehlern konzentriert sich auf inhaltliche Unzulänglichkeiten von Anforderungen. Formale Unzulänglichkeiten von Anforderungen werden in dem ausgewerteten Literaturausschnitt nicht berücksichtigt, weshalb sie auch nicht in die Typologie einfließen. Dies wäre ohnehin nur möglich, wenn bestimmte formale Regeln zugrunde gelegt würden. Dass formale Aspekte nicht berücksichtigt werden, macht die Typologie folglich auch unabhängig von Methoden oder Modellierungssprachen, anhand derer Anforderungen beschrieben werden können.

[367] Unter diesen 39 Beobachtungen mit einem allgemeinen Anforderungsfehler sind 16 Beobachtungen enthalten, bei denen die Wirkung als „instabile Anforderung“ beschrieben wird. Diese 16 Beobachtungen stammen aus drei Literaturquellen: 7 aus Curtis, Krasner, Iscoe /Field study/ (Q6), 5 aus Ebert, De Man /Requirements uncertainty/ (Q11) und 4 aus Günther, Rombach, Ruhe /Qualitätsverbesserung/ (Q12). Die Autoren dieser drei Literaturquellen verstehen unter instabilen Anforderungen implizit Anforderungsfehler im Sinne der vorliegenden Arbeit, weshalb sie auch entsprechend ausgewertet worden sind.

Vergleich der Typologie mit verwandten Forschungsergebnissen in der Literatur

In der Literatur werden mehrere Typologien von Anforderungsfehlern vorgeschlagen.[368] Die vorliegende Typologie weist zahlreiche Gemeinsamkeiten mit einem Vorschlag von Hayes auf, der für Softwareentwicklungen bei der NASA entwickelt wurde.[369] Die Unterschiede beschränken sich im Wesentlichen darauf, dass ihre Typologie in bestimmten Bereichen umfassender ist und bei manchen Typen von Anforderungsfehlern zusätzlich zwischen Untertypen differenziert. Die Tabelle 4-14 vergleicht die Typen von Anforderungsfehlern der beiden Typologien und benennt relevante Unterschiede. Die englischen Bezeichnungen aus der Hayes-Typologie wurden unverändert übernommen.

Typen nach Hayes-Typologie[370]	Entsprechender Typ nach vorliegender Typologie	Unterschiede
incompleteness	unvollständige Anforderung	Keine relevanten Unterschiede.
omitted/missing	fehlende Anforderung	Hayes-Typologie unterscheidet für diesen Typ zwei Untertypen. Einer der Untertypen entspricht dem Typ „fehlende Anforderung“.
incorrect	falsche Anforderung	Der Hayes-Typ „incorrect“ beschreibt eine Anforderung, die ein Kundenbedürfnis nicht korrekt erfüllt. Entsprechende Fehler werden in der vorliegenden Typologie durch den Typ „falsche Anforderung“ abgedeckt.
ambigous	mehrdeutige Anforderung	Keine relevanten Unterschiede.
infeasible	falsche Anforderung	Der Hayes-Typ „infeasible“ umfasst unter anderem Anforderungen, die durch kein Kundenbedürfnis begründet sind. Entsprechende Fehler werden in der vorliegenden Typologie durch den Typ „falsche Anforderung“ abgedeckt.
inconsistent	inkonsistente Anforderungen	Hayes-Typologie unterscheidet für diesen Typ zwei Untertypen. Einer der Untertypen entspricht dem Typ „inkonsistente Anforderungen“.
over-specification	falsche Anforderung	Der Hayes-Typ „over-specification“ beschreibt Anforderungen, die mehr fordern als durch das entsprechende Kundenbedürfnis begründet werden kann. Entsprechende Fehler werden in der vorliegenden Typologie durch den Typ „falsche Anforderung“ abgedeckt.
not traceable	missverständliche Anforderung	Hayes-Typ „not traceable“ hat eine engere Auffassung als der Typ „missverständliche Anforderung“, da Verknüpfungen zwischen Anforderungen nicht berücksichtigt werden.
-	redundante Anforderungen	Für den Typ „redundante Anforderungen“ gibt es keinen entsprechenden Typ in der Hayes-Typologie.

Tabelle 4-14: Vergleich der entwickelten Typologie mit der Hayes-Typologie

[368] Vgl. z. B. Hayes /Requirement fault/ 51-56, Sakthivel /Survey/ 68-70 oder Roman /Requirements engineering/ 16 f. Häufig werden Typologien von Anforderungsfehlern theoretisch hergeleitet. Ein Ansatz ist z. B. empfohlene Merkmale von Anforderungen oder Anforderungsdokumenten zu negieren und als Fehlertypen zu übernehmen. Entsprechende Merkmale werden z. B. aufgeführt in IEEE /Software requirements/ 4-8.

[369] Vgl. Hayes /Requirement fault/ 56.

[370] Vgl. Hayes /Requirement fault/ 56.

Hayes klassifiziert nachträglich 486 von geschätzten 8500 Anforderungsfehlern gemäß ihrer Typologie. Die Fehler stammen alle von Softwareentwicklungen für die internationale Raumstation ISS.[371] Eine Erfahrung dieser Untersuchung ist, dass mit ihrer Typologie Anforderungsfehler jeweils weitgehend eindeutig einem Typ zugeordnet werden können.

4.3.4.3 Ursachen für Anforderungsfehler

4.3.4.3.1 Überblick über Ursachenkategorien und Synthese der Ergebnisse

Neben der Typologie von Anforderungsfehlern wurden die Ursachen für Anforderungsfehler als ein weiteres Ergebnis der systematischen Literaturanalyse herausgearbeitet.[372] Insgesamt 20 Ursachen können nach der Auswertung der 92 Beobachtungen unterschieden werden. Diese Ursachen können zum einen direkt Prozessmängel der Anforderungsanalyse im Sinne dieser Arbeit sein. [373] Zum anderen können es Rahmenbedingungen der Softwareentwicklung sein, die Anforderungsfehler verursachen. Diese Rahmenbedingungen können nicht geändert werden. Vielmehr muss die Gestaltung eines Prozesses der Anforderungsanalyse diese Rahmenbedingungen angemessen berücksichtigen, um ihre negativen Wirkungen zu reduzieren.[374] Werden relevante Rahmenbedingungen in einem Prozess der Anforderungsanalyse nicht berücksichtigt, stellt dies ebenfalls einen Prozessmangel der Anforderungsanalyse dar. Insofern beschreiben Beobachtungen, in denen Rahmenbedingungen als Ursachen für Anforderungsfehler genannt werden, indirekt Prozessmängel der Anforderungsanalyse.

Die identifizierten Ursachen werden nachfolgend in 7 Kategorien eingeteilt und beschrieben. Die Tabelle 4-15 gibt einen Überblick über die einzelnen Ursachen und ihre Kategorien sowie die Verteilung der 92 Beobachtungen auf die 20 Ursachen und 7 Ursachenkategorien.

Um die 92 Beobachtungen vollständig darzustellen und die Teilergebnisse der systematischen Literaturanalyse zusammenzuführen, beschreiben die Tabelle 4-16 und Tabelle 4-17 in den Zeilen jeweils die Ursachen und in den Spalten die Typen von Anforderungsfehlern.[375] Sofern eine Beobachtung zu einem bestimmten Ursache-Wirkungszusammenhang vorliegt, steht in

---

[371] Vgl. zu diesem Absatz Hayes /Requirement fault/ 57 f.

[372] Vgl. zur Vorgehensweise Kapitel 4.3.2.3.

[373] Das Konstrukt eines Prozessmangels wurde in Kapitel 4.2 erörtert.

[374] Vgl. hierzu auch die Abbildung 1-3 zu Beziehungen zwischen den Elementen praxeologischer Aussagen.

[375] Die Tabelle 4-16 und Tabelle 4-17 stellen eine Konzeptmatrix im Sinne von Webster und Watson dar. Vgl. Webster xvi f.

der entsprechenden Tabellenzelle die Nummer der empirischen Literaturquelle[376] und die relevanten Seitenzahlen zu der Beobachtung in dieser Quelle.

[376] Die Zuordnung der eindeutigen Nummern zu den Literaturquellen findet sich in der Tabelle 4-4 und Tabelle 4-10.

Ursachen für Anforderungsfehler	Absolute Häufigkeit Beobachtungen		Relative Häufigkeit Beobachtungen
Dokumentation von Anforderungen		*26*	*28,26%*
Form der Dokumente	3		
Inhalt der Dokumente	7		
Software-Werkzeuge und ihre Nutzung	6		
horizontale Verfolgbarkeit	3		
vertikale Verfolgbarkeit	7		
Benutzerbeteiligung		*17*	*18,48%*
keine, zu geringe oder zu späte Benutzerbeteiligung	7		
Beteiligung ungeeigneter Personen	6		
sonstige Ausgestaltung der Benutzerbeteiligung	4		
Wissen über Anwendungsgebiet		*15*	*16,30%*
Wissensmonopole bei wenigen Entwicklern	3		
fehlendes Wissen über das Anwendungsgebiet	12		
Erhebung von Anforderungen		*12*	*13,04%*
unklare Ziele der Software	4		
unzureichende Auseinandersetzung mit Kundenbedürfnissen	5		
offene Punkte werden nicht umgehend geklärt	3		
Inhärente Komplexität der Anforderungsanalyse		*10*	*10,87%*
Vielzahl und Heterogenität der Kunden	5		
Vielzahl der Anforderungen	4		
Vielzahl der Änderungen in den Kundenbedürfnissen	1		
Projektmanagement		*9*	*9,78%*
Mangelnde Definition oder Einhaltung eines Prozesses	6		
Unzureichende Berücksichtigung von Projektabhängigkeiten	2		
politische Gründe	1		
Interne Kommunikation		*3*	*3,26%*
Kommunikation im Entwicklungsteam	3		
		Summe: 92	Summe: 100%

Tabelle 4-15: Verteilung der Beobachtungen nach Ursachen

	Anforderungsfehler allgemein	**missverständliche Anforderung**	**inkonsistente Anforderungen**	**fehlende Anforderung**	**falsche Anforderung**	**unvollständige Anforderung**	**mehrdeutige Anforderung**	**redundante Anforderungen**
Dokumentation von Anforderungen								
Form der Dokumente	Q2, 51 Q14, 47	Q2, 51						
Inhalt der Dokumente	Q14, 45	Q18, 835	Q9, 154 Q9, 154				Q8, 260	Q9, 154 Q9, 154
Software-Werkzeuge und ihre Nutzung		Q7, 51	Q5, 236	Q5, 235 Q5, 235	Q5, 235 Q5, 235			
horizontale Verfolgbarkeit	Q14, 47	Q20, 71	Q2, 55					
vertikale Verfolgbarkeit	Q14, 45 Q16, 6 Q19, 145	Q7, 51 Q18, 835		Q21, 72	Q10, 32			
Benutzerbeteiligung								
keine, zu geringe oder zu späte Benutzerbeteiligung	Q14, 47 Q14, 47	Q13, 63 Q16, 5 Q19, 144 Q19, 144		Q19,144				
Beteiligung ungeeigneter Personen	Q2, 6-8 Q11, 556 Q19, 20 Q19, 20	Q9, 154 Q9, 156						
sonstige Ausgestaltung der Benutzerbeteiligung	Q6, 1276 Q6, 1276			Q21, 70		Q4, 47		
Wissen über Anwendungsgebiet								
Wissensmonopole bei wenigen Entwicklern	Q12, 170 Q14, 45				Q5, 234			
fehlendes Wissen über das Anwendungsgebiet	Q4, 47 Q4, 48 Q6, 1271 Q12, 170 Q19, 145	Q15, 41	Q1, 1356	Q1, 1357 Q19, 144		Q5, 234	Q1, 1359 Q5, 234	

Tabelle 4-16: Konzeptmatrix: Anforderungsfehler und ihre Ursachen

	Anforderungsfehler allgemein	missverständliche Anforderung	inkonsistente Anforderungen	fehlende Anforderung	falsche Anforderung	unvollständige Anforderung	mehrdeutige Anforderung	redundante Anforderungen
Erhebung von Anforderungen								
unklare Ziele der Software	Q6, 1277 Q11, 556 Q11, 556 Q12, 170							
unzureichende Auseinandersetzung mit Kundenbedürfnissen	Q15, 41 Q16, 6			Q15, 41 Q15, 41	Q10, 31			
offene Punkte werden nicht umgehend geklärt				Q21, 72		Q4, 47 f.	Q4, 47 f.	
Inhärente Komplexität der Anforderungs-analyse								
Vielzahl und Heterogenität der Kunden	Q3, 191 f. Q6, 1276		Q17, 131 Q21, 64 f.		Q16, 5			
Vielzahl der Anforderungen	Q17, 131 Q6, 1276		Q2, 50 Q6, 1277					
Vielzahl der Änderungen in den Kundenbedürfnissen	Q6, 1276							
Projektmanagement								
mangelnde Definition oder Einhaltung eines Prozesses	Q3, 203 Q3, 191 f. Q11, 556	Q9, 155				Q4, 47 Q4, 47		
unzureichende Berücksichtigung abhängiger Projekte	Q11, 556 Q12, 170							
politische Gründe	Q6, 1276							
Interne Kommunikation								
Kommunkation zwischen Entwicklern		Q6 1279 f.	Q5, 236 Q6, 1278					

Tabelle 4-17: Konzeptmatrix: Anforderungsfehler und ihre Ursachen (Fortsetzung)

Ursache-Wirkungsdiagramm für Anforderungsfehler

In 53 von 92 Beobachtungen wird der Typ des Anforderungsfehlers genannt.[377] In der Abbildung 4-5 wird für diese Beobachtungen ein Ursache-Wirkungsdiagramm nach Ishikawa[378] dargestellt.

[377] Vgl. hierzu Kapitel 4.3.4.2.

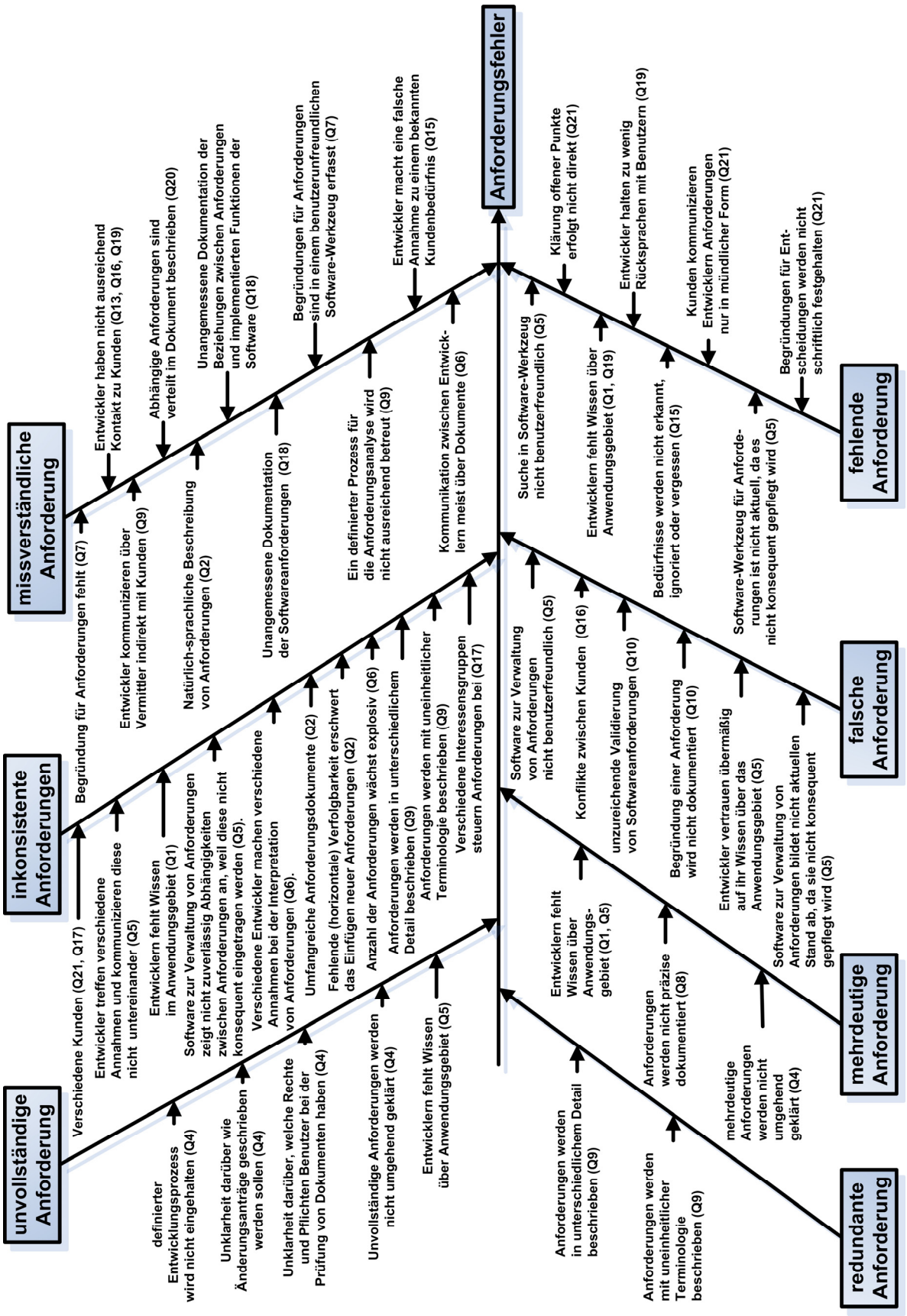


Abbildung 4-5: Ursache-Wirkungsdiagramm für Anforderungsfehler

[378] Vgl. Ishikawa /Quality Control/ 18-29.

4.3.4.3.2 Darstellung der einzelnen Ursachen

Vorgehensweise bei der Darstellung

Nach dem Überblick über die Ursachenkategorien im vorhergehenden Kapitel 4.3.4.3.1, werden die Ursachen im Folgenden einzeln beleuchtet. Die Darstellung ist nach den Ursachenkategorien gegliedert, welche nach der Anzahl der Beobachtungen absteigend sortiert sind. Jede Ursachenkategorie wird zunächst definiert. Anschließend werden die Beobachtungen zu dieser Ursachenkategorie in tabellarischer Form aufgelistet. Entsprechende Tabellen haben jeweils eine einheitliche Struktur:

- Eine eindeutige Nummer für die Beobachtung (von 1 bis 92).
- Die Beschreibung der Ursache (weitgehend im Wortlaut der Originalquelle).
- Die Beschreibung der Wirkung, d. h. den Typ des Anforderungsfehlers, sofern er angegeben ist. Ist der genaue Typ des Anforderungsfehlers nicht bekannt, wird als Wirkung nur „Anforderungsfehler“ angegeben.
- Der Fundort der Beobachtung, d. h. die Nummer der empirischen Literaturquelle[379] und die relevanten Seitenzahlen zu der Beobachtung in dieser Quelle.

Innerhalb der Ursachenkategorien erfolgt abgesehen von einer Ausnahme nochmals eine Unterteilung in Teilkategorien, für die jeweils Tabellen mit Beobachtungen aufgeführt werden.

In bestimmten Fällen besteht die Möglichkeit, einzelne Ursachen verschiedenen Ursachenkategorien zuzuordnen. Folglich wird für die Menge der Ursachenkategorien nicht der Anspruch erhoben, dass sie überlappungsfrei sind. Aus pragmatischen Gründen wird jede einzelne Ursache erstens nur einer Ursachenkategorie zugeordnet und zweitens orientiert sich die Zuordnung daran, welchen Aspekt der Ursache die Autoren der jeweiligen Literaturquelle stärker betonen.

Die Ursache eines Anforderungsfehlers ist direkt ein Prozessmangel der Anforderungsanalyse oder weist indirekt auf einen Prozessmangel hin.[380] Er legt aber nicht eindeutig den Gestaltungsbereich fest, indem durch eine Gestaltungsmaßnahme dieser Prozessmangel behoben werden kann. Fehlt einem Entwicklungsteam Wissen über das Anwendungsgebiet der Software und verursacht dieser Umstand Anforderungsfehler, kann dieser Prozessmangel beispielsweise auf verschiedene Arten behoben werden:

---

[379] Die Zuordnung der eindeutigen Nummern zu den Literaturquellen findet sich in der Tabelle 4-4 und Tabelle 4-10.

[380] Vgl. hierzu Kapitel 4.3.4.3.1.

- Ein Entwickler mit Wissen über das Anwendungsgebiet wird Mitglied des Teams (Maßnahme der Projektbesetzung).
- Im Rahmen der Benutzerbeteiligung werden insbesondere jene Benutzer eingebunden, die Wissen über das Anwendungsgebiet verfügen (Maßnahme der Benutzerbeteiligung).
- Die Entwickler setzen spezifische Methoden zur Erhebung von Anforderungen ein, die es ihnen während des Projekts ermöglichen, schnell erforderliches Wissen im Anwendungsgebiet der Software aufzubauen (Maßnahme zur Erhebung von Anforderungen).

Welche Maßnahme zweckmäßig ist, hängt von einem Projekt und seinen Rahmenbedingungen ab. Diese Überlegungen betonen nochmals, dass die Gestaltung eines Prozesses der Anforderungsanalyse und Prozessmängel der Anforderungsanalyse im hohen Maße kontextabhängig, also situativ, sind.

Ursachenkategorie: Dokumentation von Anforderungen

Die Dokumentation von Anforderungen ist eine fachlich-technische Teilaufgabe der Anforderungsanalyse.[381] Von den 92 identifizierten Beobachtungen entfallen ca. 28% auf Probleme in dieser Teilaufgabe. Die Ursachen beziehen sich im Einzelnen auf die Form und den Inhalt von Anforderungsdokumenten, eingesetzte Software-Werkzeuge und die Verfolgbarkeit von Anforderungen.

Form der Anforderungsdokumente

Bei der Gestaltung der Form von Anforderungsdokumenten geht es um die Frage, mit welchen Darstellungsmitteln Softwareanforderungen beschrieben werden sollen. Drei Beobachtungen beziehen sich auf diesen Gestaltungsbereich.

Nr.	Ursache	Wirkung	Fundort
1	Natürlich-sprachliche Anforderungsdokumente erschweren die Erkennung einzelner Softwareanforderungen aus dem Text heraus.	Anforderungsfehler	Q14, S. 47
2	Anforderungen sind natürlich-sprachlich beschrieben.	missverständliche Anforderungen	Q2, S. 51
3	Kunden fällt es schwer, Beschreibungen von Anforderungen zu verstehen, wenn ausschließlich Darstellungsmittel eingesetzt werden, die eine grafische Notation nutzen.	Anforderungsfehler	Q2, S. 51

Tabelle 4-18: Form der Anforderungsdokumente als Ursache für Anforderungsfehler

[381] Vgl. Kapitel 2.3.2.

Inhalt der Anforderungsdokumente

Sieben Beobachtungen beziehen sich auf inhaltliche Aspekte von Anforderungsdokumenten.

Nr.	Ursache	Wirkung	Fundort
4	Softwareanforderungen werden unangemessen dokumentiert.	missverständliche Anforderung	Q18, S. 835
5	Anforderungen werden nicht präzise dokumentiert, sondern oberflächlich in Form von grundlegenden Funktionen der Software kommuniziert.	mehrdeutige Anforderung	Q8, S. 260
6	Anforderungen werden in unterschiedlichem Detail beschrieben.	inkonsistente Anforderungen	Q9, S. 154
7	Anforderungen werden in unterschiedlichem Detail beschrieben.	redundante Anforderungen	Q9, S. 154
8	Anforderungen werden mit einer uneinheitlichen Terminologie beschrieben.	inkonsistente Anforderungen	Q9, S. 154
9	Anforderungen werden mit einer uneinheitlichen Terminologie beschrieben.	redundante Anforderungen	Q9, S. 154
10	Anforderungen werden aus einer technischen Perspektive dokumentiert.	Anforderungsfehler	Q14, S. 45

Tabelle 4-19: Inhalt der Anforderungsdokumente als Ursache für Anforderungsfehler

Software-Werkzeuge und ihre Nutzung

Zur Dokumentation und Verwaltung von Softwareanforderungen können besondere Software-Werkzeuge eingesetzt werden, die wesentlich mehr leisten als eine Textverarbeitungssoftware. Anforderungsfehler können auch entstehen aufgrund Unzulänglichkeiten entsprechender Software-Werkzeuge oder Unzulänglichkeiten bei ihrer Nutzung.

Nr.	Ursache	Wirkung	Fundort
11	Datenbank mit Anforderungen ist schwierig zu durchsuchen.	fehlende Anforderung	Q5, S. 235
12	Datenbank mit Anforderungen ist schwierig zu durchsuchen.	falsche Anforderung	Q5, S. 235
13	Datenbank mit Anforderungen ist nicht aktuell.	fehlende Anforderung	Q5, S. 235
14	Datenbank mit Anforderungen ist nicht aktuell.	falsche Anforderung	Q5, S. 235
15	Datenbank mit Anforderungen zeigt Abhängigkeiten zwischen Anforderungen nicht zuverlässig an, weil diese nicht immer eingetragen werden.	inkonsistente Anforderungen	Q5, S. 236
16	Anforderungen werden nicht vollständig verstanden, da ihre Begründung in einer Datenbank mit Kundenbedürfnissen "vergraben" sind	missverständliche Anforderungen	Q7, S. 51

Tabelle 4-20: Software-Werkzeuge und ihre Nutzung als Ursache für Anforderungsfehler

Verfolgbarkeit von Anforderungen

Eine Anforderung ist verfolgbar, sofern sie mit abhängigen, eindeutig identifizierbaren Einzelergebnissen explizit verknüpft ist.[382] Es können drei grundlegende Typen der Verfolgbarkeit unterschieden werden:[383]

[382] In Anlehnung an IEEE /Glossary/ 82.

- Die horizontale Verfolgbarkeit bezieht sich auf Verknüpfungen zwischen Einzelergebnissen der gleichen Teilaufgabe. D. h., eine Anforderung ist mit abhängigen Anforderungen und Entwicklungseinschränkungen[384] verknüpft.
- Die vertikale Vorverfolgbarkeit bezieht sich auf Verknüpfungen zwischen einer Anforderung und abhängigen Einzelergebnissen vorgelagerter Teilaufgaben. Diese Einzelergebnisse begründen den Ursprung der Anforderung. Hierzu zählen z. B. abhängige Kundenbedürfnisse und Zuordnungen zu Kundengruppen.
- Die vertikale Nachverfolgbarkeit bezieht sich auf Verknüpfungen zwischen einer Anforderung und abhängigen Einzelergebnissen nachgelagerter Teilaufgaben. Diese Einzelergebnisse werden durch die Anforderung begründet. Hierzu zählen z. B. Entwurfsentscheidungen, Implementierungsentscheidungen und Testfälle.

Die Abbildung 4-6 veranschaulicht die Typen der Verfolgbarkeit.

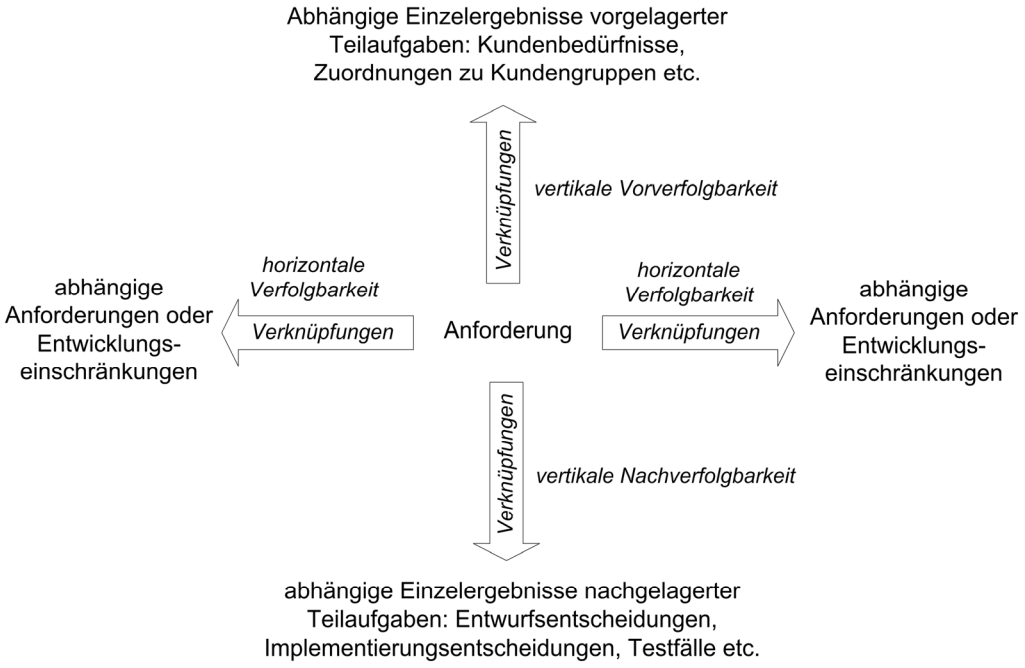


Abbildung 4-6: Verfolgbarkeit einer Anforderung

Auf die Verfolgbarkeit von Anforderungen beziehen sich 10 Beobachtungen, davon 3 auf die horizontale (vgl. Tabelle 4-21) und 7 auf die vertikale Verfolgbarkeit (vgl. Tabelle 4-22).

[383] Vgl. zu Folgendem Boldyreff u. a. /Traceability/ 101 und Gotel, Finkelstein /Requirements traceability/ 96 f.
[384] Vgl. Kapitel 2.2.1.

Nr.	Ursache	Wirkung	Fundort
17	Beziehungen zwischen Anforderungen wurden nicht dokumentiert.	Anforderungsfehler	Q14, S. 47
18	Fehlende Verfolgbarkeit erschwert das Einfügen neuer Anforderungen	inkonsistente Anforderungen	Q2, S. 55
19	Abhängige Anforderungen, die mit der Strukturierten Analyse beschrieben worden sind, sind verteilt im Dokument beschrieben.	missverständliche Anforderung	Q20, S. 71

Tabelle 4-21: Horizontale Verfolgbarkeit als Ursache für Anforderungsfehler

Von den 7 Benobachtungen zur vertikalen Verfolgbarkeit hat ausschließlich die Beobachtung 20 die Nachverfolgbarkeit zum Gegenstand. Die Beobachtungen 21 bis 26 beziehen sich auf die Vorverfolgbarkeit von Anforderungen.

Nr.	Ursache	Wirkung	Fundort
20	Unangemessene Dokumentation der Beziehungen zwischen Softwareanforderungen und implementierten Funktionen der Software.	missverständliche Anforderung	Q18, S. 835
21	Begründung einer Entscheidung wird vergessen oder nicht festgehalten.	fehlende Anforderung	Q21, S. 72
22	Annahmen über Entscheidungen werden nicht festgehalten.	Anforderungsfehler	Q16, S. 6
23	Beziehungen zwischen geschäftlichen Informationen und Anforderungen fehlen.	Anforderungsfehler	Q14, S. 45
24	Häufig fehlt die Begründung der Anforderungen.	missverständliche Anforderung	Q7, S. 51
25	Begründung einer Anforderung wird nicht dokumentiert.	falsche Anforderung	Q10, S. 32
26	Sitzungen werden nicht dokumentiert, so dass die Verfolgbarkeit des Entscheidungsprozesses nicht möglich ist.	Anforderungsfehler	Q19, S. 145

Tabelle 4-22: Vertikale Verfolgbarkeit als Ursache für Anforderungsfehler

Ursachenkategorie: Benutzerbeteiligung

Benutzer einer Software sind jene Personen, die unmittelbar mit der Software arbeiten.[385] Der Begriff der Benutzerbeteiligung wird in der Literatur sehr uneinheitlich definiert.[386] In dieser Arbeit wird unter der Benutzerbeteiligung allgemein verstanden, dass Benutzer oder Vertreter von Benutzern an einem Softwareentwicklungs- oder Wartungsprojekt mitwirken. Anhand bestimmter Fragen können verschiedene Formen der Benutzerbeteiligung unterschieden und präzisiert werden:[387]

- Wer wird beteiligt?

  Sofern Benutzer beteiligt werden, liegt eine *direkte Benutzerbeteiligung* vor. Bei einer kleinen Anzahl von Benutzern können dies alle Benutzer sein. Ebenso können ausge-

[385] Vgl. Kapitel 2.2.1.

[386] Vgl. Aslim /Benutzerbeteiligung/ 4-7.

[387] In Anlehnung an Baitsch u. a. /Büroarbeit/ 88.

wählte Benutzer bestimmte Benutzergruppen vertreten. Werden hingegen Personen beteiligt, die Benutzer vertreten sollen und selbst keine Benutzer sind, ist eine *indirekte Benutzerbeteiligung* gegeben.

- Wie werden Benutzer bzw. Vertreter von Benutzern beteiligt?

  Benutzer bzw. Vertreter von Benutzern können auf drei grundlegende Arten an Softwareentwicklungs- oder Wartungsprojekten beteiligt werden.[388] Beim *wechselseitigen Informationsaustausch* tauschen Benutzer und Entwickler lediglich Informationen aus, die relevant für das Projekt sind.[389] Dieser Informationsaustausch kann durch Maßnahmen wie z. B. gemeinsamen Sitzungen, Kommunikation über E-Mails oder über Telefon umgesetzt werden. Bei einer *Gestaltungsbeteiligung* sind Benutzer oder ihre Vertreter Mitglied des Entwicklungsteams und erstellen gemeinsam mit den Entwicklern Zwischenergebnisse der Softwareentwicklung.[390] Während beim wechselseitigen Informationsaustausch Benutzer oder ihre Vertreter die Rolle eines externen Informationslieferanten aus Sicht des Entwicklungsteams innehaben, gehören sie bei der Gestaltungsbeteiligung mit zum Entwicklungsteam und tragen aktiv zu Ergebnissen bei. Bei einer *Entscheidungsbeteiligung* nehmen Benutzer oder ihre Vertreter Ergebnisse des Entwicklungsteams ab.[391]

  Die Gestaltungs- und Entscheidungsbeteiligung beinhalten beide den wechselseitigen Informationsaustausch. Zur Vereinfachung wird im Folgenden von einer *schwach ausgeprägten Benutzerbeteiligung* gesprochen, sofern ausschließlich ein wechselseitiger Informationsaustausch vorliegt. Bei einer *mittel ausgeprägten Benutzerbeteiligung* ist entweder eine Gestaltungs- oder eine Entscheidungsbeteiligung gegeben. Sofern sowohl eine Gestaltungs- als auch eine Entscheidungsbeteiligung vorliegen, charakterisiert dies eine *stark ausgeprägte Benutzerbeteiligung*. Von der schwach zu stark ausgeprägten Benutzerbeteiligung nimmt die Einflussmöglichkeit der Benutzer oder ihrer Vertreter auf die Softwareentwicklung zu.

---

[388] Vgl. Peschke /Systementwicklung/ 87.

[389] Die Ausprägung des wechselseitigen Informationsaustausch umfasst beispielsweise das Konzept „involvement by advice“ nach Ives und Olson (vgl. Ives, Olson /User involvement/ 590) und die „non-instrumental voice“ nach Hunton und Beeler (vgl. Hunton, Beeler /User participation/ 365).

[390] Die Ausprägung der Gestaltungsbeteiligung umfasst beispielsweise das Konzept „involvement by doing“ nach Ives und Olson (vgl. Ives, Olson /User involvement/ 590).

[391] Die Ausprägung der Entscheidungsbeteiligung umfasst beispielsweise das Konzept „involvement by weak control“ (vgl. Ives, Olson /User involvement/ 590) und die „instrumental voice“ nach Hunton und Beeler (vgl. Hunton, Beeler /User participation/ 365).

- Bei welchen Teilaufgaben der Softwareentwicklung werden Benutzer bzw. Vertreter von Benutzern beteiligt?

  Benutzer oder ihre Vertreter können bei verschiedenen Teilaufgaben der Softwareentwicklung mitwirken. Eine *frühe Benutzerbeteiligung* liegt vor, wenn sie neben anderen Teilaufgaben insbesondere bei der Anforderungsanalyse mitwirken. In Abgrenzung hierzu ist eine eine *späte Benutzerbeteiligung* gegeben, wenn sie bei der Anforderungsanalyse nicht mitwirken, dafür aber an anderen Teilaufgaben wie z. B. der Qualitätssicherung oder der Endabnahme beteiligt sind.

Eine Form der Benutzerbeteiligung kann demnach zum Beispiel folgende Ausprägungen besitzen: direkte Benutzerbeteiligung, wechselseitiger Informationsaustausch (schwache Benutzerbeteiligung), Mitwirkung an Anforderungsanalyse und Qualitätssicherung und damit eine frühe Benutzerbeteiligung. Während eines Softwareentwicklungsprojekts können mehrere Formen der Benutzerbeteiligung auftreten.

Von den 92 identifizierten Beobachtungen führen ca. 18% Ursachen auf, die mit der Benutzerbeteiligung zusammenhängen. Die nachfolgenden drei Tabellen (Tabelle 4-23, Tabelle 4-24 und Tabelle 4-25) listen diese Ursachen auf. Die Beobachtungen wurden weitgehend im Wortlaut der jeweiligen Quellen übernommen. In den empirischen Quellen wird auf eine genaue Definition der Benutzerbeteiligung verzichtet. Ebenso finden sich meist nur Anhaltspunkte darüber, welche Form einer Benutzerbeteiligung gemeint ist.

Keine, zu geringe oder zu späte Benutzerbeteiligung

Tabelle 4-23 enthält Beobachtungen zu einer fehlenden, geringen oder späten Benutzerbeteiligung. Gemäß 6 Beobachtungen (Beobachtungen 27 bis 32) führt eine zu geringe Intensität des wechselseitigen Informationsaustauschs oder eine fehlende mittel oder stark ausgeprägte Benutzerbeteiligung zu Anforderungsfehlern. Eine späte Benutzerbeteiligung (Beobachtung 33) kann ebenso Anforderungsfehler verursachen.

Nr.	Ursache	Wirkung	Fundort
27	Es finden nur wenige Rücksprachen mit Benutzern statt.	fehlende Anforderung	Q19, S. 144
28	Entwickler haben nicht ausreichend Kontakt mit Kunden.	missverständliche Anforderung	Q16, S. 5
29	Kunden werden nicht ausreichend eingebunden.	missverständliche Anforderung	Q13, S. 63
30	Entwickler haben keinen Kontakt zu Benutzern.	missverständliche Anforderung	Q19, S. 144
31	Entwicklungsteam hat keinen Kontakt zu Benutzern	Anforderungsfehler	Q14, S. 47
32	Kunden und Entwickler kommunizieren selten.	missverständliche Anforderung	Q19, S. 144
33	Entwicklungsteam hat zu spät Kontakt zu Benutzern	Anforderungsfehler	Q14, S. 47

Tabelle 4-23: Keine, zu geringe oder zu späte Benutzerbeteiligung als Ursache für Anforderungsfehler

Beteiligung ungeeigneter Personen

Werden ungeeignete Personen beteiligt, kann dies ebenfalls nach 6 Beobachtungen Anforderungsfehler verursachen (vgl. Tabelle 4-24). Zum einen können Anforderungsfehler entstehen, weil Benutzer durch Personen vertreten werden und diese Vertreter selbst keine Benutzer sind. Diese indirekte Benutzerbeteiligung wird als Ursache in den Beobachtungen 34 bis 36 genannt. Zum anderen kann aber auch eine direkte Benutzerbeteiligung Anforderungsfehler verursachen, wenn Benutzergruppen durch Benutzer vertreten werden, die für diese Aufgabe nicht ausreichende Erfahrungen und oder nicht das erforderliche Wissen im Anwendungsgebiet aufweisen (Beobachtungen 37 bis 39).

Nr.	Ursache	Wirkung	Fundort
34	Aufgrund von Vermittlern gibt es eine Kommunikationslücke zwischen Kunden und Entwicklern.	missverständliche Anforderung	Q9, S. 156
35	Indirekte Kommunikation mit Kunden bei der Erhebung und Untersuchung der bestehenden Software.	missverständliche Anforderung	Q9, S. 154
36	Wahren Benutzer werden nicht beteiligt	Anforderungsfehler	Q19, S. 20
37	Schlüsselpersonen werden nicht einbezogen.	Anforderungsfehler	Q11, S. 556
38	Erfahrene Benutzer haben keine Zeit.	Anforderungsfehler	Q19, S. 20
39	Kunden wählen nicht Mitarbeiter mit dem meisten Wissen im Anwendungsgebiet aus, um mit den Softwareentwicklern zu arbeiten	Anforderungsfehler	Q2, S. 6-8

Tabelle 4-24: Beteiligung ungeeigneter Personen als Ursache für Anforderungsfehler

Sonstige Ausgestaltung der Benutzerbeteiligung

Die Benutzerbeteiligung birgt als eine Gestaltungsmaßnahme gewisse Risiken, die bei ihrer konkreten Ausgestaltung in einem Projekt berücksichtigt werden müssen, da sie ansonsten ebenfalls Anforderungsfehler verursachen können (vgl. Tabelle 4-25). Häufig verstehen Kunden die Komplexität von Entwicklungsprozessen nicht. Insbesondere können sie nicht die Auswirkungen von Anforderungsänderungen abschätzen. Je größer die Einflussmöglichkeiten von Benutzern im Rahmen der Benutzerbeteiligung ausfallen, desto eher kann dies die Benutzer dazu verleiten, zum einen häufig Änderungen von Anforderungen zu verlangen und zum anderen entsprechende Änderungsanträge ohne eine formale Prüfung durchzusetzen (Beobachtungen 40 und 41). Wie notwendig es ist, die Form der Benutzerbeteiligung zu definieren und den Betroffenen zu kommunizieren, zeigen die Beobachtungen 42 und 43. Sind die genauen Aufgaben, Rechten und Pflichten der Benutzer oder ihrer Vertreter unklar, kann dies dazu führen, dass Anforderungen den Entwicklern ausschließlich in mündlicher Form vermittelt werden oder die Prüfung von Dokumenten ineffektiv bleibt.

Nr.	Ursache	Wirkung	Fundort
40	Kunden verstehen selten die Komplexität des Entwicklungsprozesses und verlangen daher häufig Änderungen in den Anforderungen.	Anforderungsfehler	Q6, S. 1276
41	Wenn Kunden direkten Zugriff auf das Entwicklungsteam haben, verlangen sie oft Änderungen in den Anforderungen ohne eine formale Prüfung des Änderungsantrags	Anforderungsfehler	Q6, S. 1276
42	Anforderungen werden vom Kunden an Entwickler nur in mündlicher Form übergeben.	fehlende Anforderung	Q21, S. 70
43	Es ist nicht klar definiert, welche Rechten und Pflichten Benutzer bei der Prüfung von Dokumenten haben.	unvollständige Anforderung	Q4, S. 47

Tabelle 4-25: Sonstige Ausgestaltung der Benutzerbeteiligung als Ursache für Anforderungsfehler

Ursachenkategorie: Wissen über das Anwendungsgebiet

Unter dem Begriff Wissen werden begründete Überzeugungen verstanden, die die Fähigkeit einer Person zum effektiven Handeln verbessern.[392] Für die Softwareentwicklung sind zwei Typen von Wissen besonders relevant:[393]

- *technisches Wissen*, das genutzt wird, um eine Software zu entwickeln (z. B. Wissen über Entwicklungsmethoden, Programmiersprachen, Entwicklungsumgebungen etc.).
- *Wissen über das Anwendungsgebiet der Software* (z. B. Geschäftsprozesse der Kunden der Software, Geschäftsregeln, Bedürfnisse der Kunden etc.).

Von den 92 identifizierten Beobachtungen führen ca. 16% Ursachen auf, die mit dem Wissen über das Anwendungsgebiet zusammenhängen. Die Tabelle 4-26 beschreibt Probleme mit Wissensmonopolen durch wenige Personen. Die geringe Verbreitung von Wissen über das Anwendungsgebiet im Entwicklungsteam ist Gegenstand der Tabelle 4-27.

Wissensmonopole bei wenigen Entwickler

Sofern nur wenige Personen im Entwicklungsteam Wissen über das Anwendungsgebiet einer Software verfügen und das relevante Wissen nicht dokumentieren, birgt dieser Umstand gewisse Risiken (vgl. Tabelle 4-26). Wird diesen Risiken nicht begegnet, können Anforderungsfehler entstehen. Wenn beispielsweise entsprechende Personen ausfallen, fehlt mit den Personen das erforderliche Wissen im Team (Beobachtungen 44 und 45). Ebenso besteht die Gefahr, dass entsprechende Entwickler übermäßig auf ihr Wissen über das Anwendungsgebiet vertrauen und deshalb Anforderungen aufnehmen, die von den Kunden gar nicht gewünscht werden (Beobachtung 46).

[392] Vgl. Nonaka /Knowledge/ 15 f. und Alavi, Leidner /Knowledge management/ 109. Für unterschiedliche Definition des Begriffs siehe Alavi, Leidner /Knowledge management/ 108-110.

[393] Vgl. Tiwana /Knowledge integration/ 900.

Nr.	Ursache	Wirkung	Fundort
44	Es gibt „Kopfmonopole“ an kritischen Stellen.	Anforderungsfehler	Q12, S. 170
45	Informationen über Kunden und Benutzer sind implizites Wissen in den Köpfen erfahrener Personen.	Anforderungsfehler	Q14, S. 45
46	Entwickler vertrauen übermäßig auf ihr eigenes Wissen über das Anwendungsgebiet.	falsche Anforderung	Q5, S. 234

Tabelle 4-26: Wissensmonopole bei wenigen Entwickler als Ursache für Anforderungsfehler

Fehlendes Wissen über das Anwendungsgebiet

Häufig ist in Entwicklungsteams Wissen über das Anwendungsgebiet gering verbreitet, was verschiedene Typen von Anforderungsfehlern zur Folge haben kann (vgl. Tabelle 4-27).

Nr.	Ursache	Wirkung	Fundort
47	Mangelnde Erfahrungen im Anwendungsgebiet.	fehlende Anforderung	Q1, S. 1357
48	Mangelnde Erfahrungen im Anwendungsgebiet.	mehrdeutige Anforderung	Q1, S. 1359
49	Mangelnde Erfahrungen im Anwendungsgebiet.	inkonsistente Anforderungen	Q1, S. 1356
50	Entwickler verstehen die Bedürfnisse der Benutzer nicht.	Anforderungsfehler	Q4, S. 48
51	Mangelndes Wissen über das Anwendungsgebiet.	Anforderungsfehler	Q4, S. 47
52	Wissen über spezielles Anwendungsgebiet erforderlich.	unvollständige Anforderung	Q5, S. 234
53	Wissen über spezielles Anwendungsgebiet erforderlich.	mehrdeutige Anforderung	Q5, S. 234
54	Entwickler haben kein ausreichendes Wissen über das Anwendungsgebiet, um die Absicht des Kunden aus einer Anforderung zu interpretieren.	Anforderungsfehler	Q6, S. 1271
55	Fehlende fachliche Erfahrungen der Projektgruppen-Mitglieder in dem mit dem Projekt verbundenen Geschäftsprozessen.	Anforderungsfehler	Q12, S. 170
56	Entwickler kennt ein Kundenbedürfnis, macht jedoch eine falsche Annahme darüber.	missverständliche Anforderung	Q15, S. 41
57	Bei den Entwicklern fehlt Wissen über das Anwendungsgebiet.	Anforderungsfehler	Q19, S. 145
58	Mangel an Experten des Anwendungsgebiets im Entwicklungsteam.	fehlende Anforderung	Q19, S. 144

Tabelle 4-27: Fehlendes Wissen über das Anwendungsgebiet als Ursache für Anforderungsfehler

Ursachenkategorie: Erhebung von Anforderungen

Um Softwareanforderungen zu erheben, müssen *Kundenbedürfnisse und Entwicklungseinschränkungen* verschiedener Kundengruppen erkannt und verstanden werden.[394] Aus den Kundenbedürfnissen werden dann Softwareanforderungen abgeleitet. Die Erhebung von Anforderungen setzt voraus, dass eine *Vision über die Software und ihre Ziele* vorliegt. Sofern sich *offene Punkte bei erhobenen Anforderungen* ergeben, sind diese möglichst zeitnah zu klären.

[394] Vgl. Kapitel 2.3.2.

Von den 92 identifizierten Beobachtungen führen ca. 13% Ursachen auf, die mit der Erhebung von Anforderungen zusammenhängen. Die nachfolgenden drei Tabellen führen folgende Punkte im Einzelnen als Ursachen für Anforderungsfehler auf:

- unklare Ziele der Software (Tabelle 4-28),
- unzureichende Auseinandersetzung mit Kundenbedürfnissen (Tabelle 4-29) und
- die späte Klärung offener Punkte bei erhobenen Anforderungen (Tabelle 4-30).

Unklare Ziele der Software

Die Erhebung von Anforderungen wird erschwert, sofern eine Vision über den grundlegenden Nutzen der Software fehlt, also die Ziele der Software unklar sind. Die vier Beobachtungen 59 bis 62 beschreiben entsprechende Probleme.

Nr.	Ursache	Wirkung	Fundort
59	Business Case wurde nicht gründlich bewertet.	Anforderungsfehler	Q11, S. 556
60	Eine klare Vision des Produkts ist erforderlich, um sie in klare Produktanforderungen zu übersetzen.	Anforderungsfehler	Q6, S. 1277
61	Vage Produktvision und –strategie	Anforderungsfehler	Q11, S. 556
62	Fehlende vollständige, eindeutige und durchführbare Zielvorgaben	Anforderungsfehler	Q12, S. 170

Tabelle 4-28: Unklare Ziele als Ursache für Anforderungsfehler

Unzureichende Auseinandersetzung mit Kundenbedürfnissen

Wenn sich das Entwicklungsteam unzureichend mit den Kundenbedürfnissen auseinandersetzt, treten sehr wahrscheinlich Anforderungsfehler auf. Eine unzureichende Auseinandersetzung kann sich darin äußern, dass

- bestehende Bedürfnisse nicht entdeckt werden (Beobachtung 63),
- entdeckte Bedürfnisse ignoriert oder vergessen werden (Beobachtung 64),
- nicht ausreichend untersucht wird, ob erhobene Softwareanforderungen die Kundenbedürfnisse erfüllen (Beobachtung 65) und
- inkonsistente Kundenaussagen nicht aufgelöst werden können (Beobachtung 66).

Die Kunden können ebenfalls zu Anforderungsfehlern beisteuern, wenn sie selbst Anforderungen fordern, die durch keine wahren Bedürfnisse begründet werden können (Beobachtung 67).

Nr.	Ursache	Wirkung	Fundort
63	Bestehende Bedürfnisse werden nicht erkannt	fehlende Anforderung	Q15, S. 41
64	Entdeckte Bedürfnisse werden ignoriert oder vergessen	fehlende Anforderung	Q15, S. 41
65	Softwareanforderungen werden unzureichend validiert.	falsche Anforderung	Q10, S. 31
66	Entwicklern gelingt es nicht, inkonsistente Kundenaussagen aufzulösen.	Anforderungsfehler	Q15, S. 41
67	Kunden wünschen Anforderungen, die unzweckmäßig sind oder ihre wahren Bedürfnisse nicht erfüllen	Anforderungsfehler	Q16, S. 6

Tabelle 4-29: Unzureichende Auseinandersetzung mit Kundenbedürfnissen als Ursache für Anforderungsfehler

Offene Punkte werden nicht umgehend geklärt

Gemäß den Beobachtungen 68 bis 70 entstehen auch dann Anforderungsfehler, wenn die Klärung offener Punkte verzögert wird.

Nr.	Ursache	Wirkung	Fundort
68	Entwickler und Benutzer sehen nicht die Notwendigkeit, unvollständige Anforderungen umgehend zu klären.	unvollständige Anforderung	Q4, S. 47 f.
69	Entwickler und Benutzer sehen nicht die Notwendigkeit, mehrdeutige Anforderungen umgehend zu klären.	mehrdeutige Anforderung	Q4, S. 47 f.
70	Offene Punkte werden nicht direkt geklärt	fehlende Anforderung	Q21, S. 72

Tabelle 4-30: Späte Klärung offener Punkte als Ursache für Anforderungsfehler

Ursachenkategorie: inhärente Komplexität der Anforderungsanalyse

Die Anforderungsanalyse hat eine inhärente Komplexität. Diese Komplexität kann sich aus der *Vielzahl und Heterogenität der Kunden*, der *Vielzahl der Anforderungen* und der *Vielzahl der Änderungen in den Kundenbedürfnissen oder Anforderungen* ergeben. Die inhärente Komplexität ist weitgehend eine Rahmenbedingung, die kaum beeinflusst werden kann. Trotzdem muss diese Rahmenbedingung bei der Gestaltung eines Prozesses der Anforderungsanalyse angemessen berücksichtigt werden, um ihre negativen Wirkungen zu reduzieren.[395] Werden relevante Rahmenbedingungen in einem Prozess der Anforderungsanalyse nicht berücksichtigt, stellt dies ebenfalls einen Prozessmangel der Anforderungsanalyse im Sinne dieser Ar-beit dar.

Die inhärente Komplexität der Anforderungsanalyse tritt bei 11% der Beobachtungen als Ursache für Anforderungsfehler auf. Die nachfolgenden drei Tabellen führen die einzelnen Aspekte der inhärenten Komplexität im Einzelnen als Ursachen für Anforderungsfehler auf:

- Vielzahl und Heterogenität der Kunden (Tabelle 4-31),
- Vielzahl der Anforderungen (Tabelle 4-32) und

[395] Vgl. hierzu auch die Abbildung 1-3 zu Beziehungen zwischen den Elementen praxeologischer Aussagen.

- Vielzahl der Änderungen in den Kundenbedürfnissen oder Anforderungen (Tabelle 4-33).

Vielzahl und Heterogenität der Kunden

Die Beobachtungen 71 bis 75 haben zum Inhalt, dass verschiedene Kunden bzw. Kundengruppen unterschiedliche und zum Teil sich widersprechende Bedürfnisse haben können.

Nr.	Ursache	Wirkung	Fundort
71	Verschiedene Interessensgruppen steuern Anforderungen bei.	inkonsistente Anforderungen	Q17, S. 131
72	Konflikte zwischen verschiedenen Kunden.	falsche Anforderung	Q16, S. 5
73	Es gibt sehr verschiedene Kunden.	Anforderungsfehler	Q3, S. 191 f.
74	Verschiedene Kunden haben verschiedene Bedürfnisse.	Anforderungsfehler	Q6, S. 1276
75	Es gibt verschiedene Kunden.	inkonsistente Anforderungen	Q21, S. 64 f.

Tabelle 4-31: Vielzahl und Heterogenität der Kunden als Ursache für Anforderungsfehler

Vielzahl der Anforderungen

Die Vielzahl der Anforderungen erschwert zum einen ihre Validierung (Beobachtung 76) und zum anderen die Sicherstellung der Konsistenz (Beobachtung 77). Dieses Problem kann dadurch verschärft werden, dass die Anzahl der Anforderungen während der gesamten Projektlaufzeit steigt, da beispielsweise die Kunden selbst erst in der direkten Auseinandersetzung mit Zwischenergebnissen der Softwareentwicklung ihre eigenen Anforderungen besser verstehen (Beobachtung 78 und 79).

Nr.	Ursache	Wirkung	Fundort
76	Die Anzahl der Anforderungen erschwert ihre Validierung	Anforderungsfehler	Q17, S. 131
77	Die Anforderungsdokumente sind umfangreich.	inkonsistente Anforderungen	Q2, S. 50
78	Die Anzahl der Anforderungen wächst explosiv.	inkonsistente Anforderungen	Q6, S. 1277
79	Kunden wünschen nachträglich zusätzliche Anforderungen, sobald sie im Projektverlauf besser verstehen, was die geplante Software leisten soll.	Anforderungsfehler	Q6, S. 1276

Tabelle 4-32: Vielzahl der Anforderungen als Ursache für Anforderungsfehler

Vielzahl der Änderungen in den Kundenbedürfnissen

Änderungen in den Kundenbedürfnissen können ebenfalls Anforderungsfehler verursachen.

Nr.	Ursache	Wirkung	Fundort
80	Die Bedürfnisse eines Kunden ändern sich im Verlauf der Zeit.	Anforderungsfehler	Q6, S. 1276

Tabelle 4-33: Vielzahl der Änderungen als Ursache für Anforderungsfehler

Ursachenkategorie: Projektmanagement

Das Software-Projektmanagement ist eine Aufgabe mit dem Ziel, ein Softwareentwicklungsprojekt zu planen, zu steuern und zu überwachen.[396]

Von den 92 identifizierten Beobachtungen führen ca. 10% Ursachen auf, die mit dem Software-Projektmanagement zusammenhängen. Die einzelnen Ursachen sind nach folgenden Kategorien gruppiert:

- Mangelnde Definition oder Einhaltung eines Prozesses der Anforderungsanalyse (Tabelle 4-34),
- unzureichende Berücksichtigung abhängiger Projekte (Tabelle 4-35) und
- politische Gründe (Tabelle 4-36).

Mangelnde Definition oder Einhaltung eines Prozesses der Anforderungsanalyse

Das Software-Projektmanagement schließt mit ein, dass ein Prozess der Anforderungsanalyse definiert und dessen Einhaltung überwacht wird. Erfolgt dies nicht, können Anforderungsfehler entstehen (Beobachtungen 81 bis 86).

Nr.	Ursache	Wirkung	Fundort
81	Ein definierter Prozess der Anforderungsanalyse fehlt.	Anforderungsfehler	Q3, S. 203
82	Ein Anforderungsdokument existiert nicht und einzelne Teamleiter pflegen eigene Listen mit Anforderungen.	Anforderungsfehler	Q3, S. 191 f.
83	Es unklar, wie ein Änderungsantrag für Anforderungen geschrieben werden soll, der sowohl für Entwickler als auch Benutzer geeignet ist.	unvollständige Anforderung	Q4, S. 47
84	Anforderungen werden nicht ausreichend beschrieben und analysiert.	Anforderungsfehler	Q11, S. 556
85	Eine mangelnde Betreuung des definierten Prozesses der Anforderungsanalyse führt dazu, dass einzelne Teams ihn nicht einhalten.	missverständliche Anforderung	Q9, S. 155
86	Ein definierter Entwicklungsprozess wird nicht eingehalten.	unvollständige Anforderung	Q4, S. 47

Tabelle 4-34: Mangelnde Definition oder Einhaltung eines Prozesses als Ursache für Anforderungsfehler

Unzureichende Berücksichtigung abhängiger Projekte

In Unternehmen laufen häufig mehrere Projekte parallel. Mögliche Abhängigkeiten zwischen diesen Projekten sind frühzeitig zu beachten, da andernfalls Anforderungsfehler verursacht werden können (Beobachtungen 87 und 88).

Nr.	Ursache	Wirkung	Fundort
87	Es existieren unbekannte Projektabhängigkeiten.	Anforderungsfehler	Q11, S. 556
88	Parallele Projekte beeinflussen sich gegenseitig.	Anforderungsfehler	Q12, S. 170

Tabelle 4-35: Unzureichende Berücksichtigung abhängiger Projekte als Ursache für Anforderungsfehler

[396] In Anlehnung an Schwarze /Projektmanagement/14 f. und Kerzner /Project management/ 4 f.

Politische Gründe

Sofern die Entwicklung einer Software ausgeschrieben wird und die Vergabe des Projekts von Ergebnissen der Anforderungsanalyse abhängt, besteht die Gefahr, dass ein Projektteam berechnend ausschließlich den Gewinn der Vergabe anstrebt und die Anforderungsanalyse nicht mit der erforderlichen Sorgfalt durchführt (Beobachtung 89).

Nr.	Ursache	Wirkung	Fundort
89	Das anfängliche Projektteam ist mehr daran interessiert, eine Ausschreibung zu gewinnen als präzise die erforderlichen Kosten und Ressourcen zu schätzen	Anforderungsfehler	Q6, S. 1276

Tabelle 4-36: Politische Gründe als Ursache für Anforderungsfehler

Interne Kommunikation

Die Kommunikation innerhalb des Entwicklungsteams kann ebenfalls dazu beitragen, dass Fehler entstehen.

Nr.	Ursache	Wirkung	Fundort
90	Entwickler treffen verschiedene Annahmen und kommunizieren diese nicht untereinander.	inkonsistente Anforderungen	Q5, S. 236
91	Verschiedene Entwickler machen verschiedene Annahmen bei der Interpretation von Anforderungen.	inkonsistente Anforderungen	Q6, S. 1278
92	Entwickler kommunizieren im Wesentlichen über Dokumente.	missverständliche Anforderung	Q6, S. 1279 f.

Tabelle 4-37: Kommunikation im Entwicklungsteam als Ursache für Anforderungsfehler

4.3.4.4 Diskussion der Ergebnisse

Die Tabelle 4-38 gibt einen Überblick über die Häufigkeiten der einzelnen Ursachenkategorien (in den Zeilen) und die Häufigkeiten der Typen von Anforderungsfehlern (in den Spalten). Missverständliche, inkonsistente und fehlende Anforderungen machen 67% der Anforderungsfehler aus, deren Typ bekannt ist (53 von 92, ca. 58%). Gemäß der systematischen Literaturanalyse tragen die Ursachenkategorien Dokumentation von Anforderungen, Benutzerbeteiligung, Wissen über das Anwendungsgebiet und Erhebung von Anforderungen zu ca. 73% der Fehler bei.

Nachfolgend werden zentrale Ergebnisse dieser vier Ursachenkategorien herausgestellt und mit relevanten empirischen Befunden in der Literatur verglichen.[397]

[397] Wobei sich entsprechende Erörterungen auf die empirische Literatur beziehen, die in der Literaturanalyse nicht berücksichtigt wurde.

	Anforderungsfehler allgemein	missverständliche Anforderung	inkonsistente Anforderungen	fehlende Anforderung	falsche Anforderung	unvollständige Anforderung	mehrdeutige Anforderung	redundante Anforderungen	Summe	prozentualer Anteil
Dokumentation von Anforderungen	7	6	4	3	3		1	2	26	28,26%
Benutzerbeteiligung	8	6		2		1			17	18,48%
Wissen über Anwendungsgebiet	7	1	1	2	1	1	2		15	16,30%
Erhebung von Anforderungen	6			3	1	1	1		12	13,04%
Inhärente Komplexität der Anforderungsanalyse	5		4		1				10	10,87%
Projektmanagement	6	1				2			9	9,78%
Interne Kommunikation		1	2						3	3,26%
Summe	39	15	11	10	6	5	4	2		
prozentualer Anteil	42,39 %	16,30 %	11,96 %	10,87 %	6,52 %	5,43 %	4,35 %	2,17 %		
prozentualer Anteil	42,39 %	57,61%								

Tabelle 4-38: Häufigkeiten von Ursachenkategorien und Typen von Anforderungsfehlern

Dokumentation von Anforderungen

Die identifizierten Ursachen für Anforderungsfehler innerhalb der Dokumentation von Anforderungen beziehen sich im Einzelnen auf die Form und den Inhalt von Anforderungsdokumenten, auf die eingesetzten Software-Werkzeuge und die Verfolgbarkeit von Anforderungen. Prozessmängel in diesem Bereich verursachen insbesondere missverständliche, inkonsistente, fehlende und falsche Anforderungen.

Form der Anforderungsdokumente

Bei der Gestaltung der Form von Anforderungsdokumenten geht es um die Frage, mit welchen Darstellungsmitteln Softwareanforderungen beschrieben werden sollen. Innerhalb dieses Gestaltungsbereichs ist insbesondere die Frage zu beantworten, ob Anforderungen natürlich-

sprachlich oder semi-formal mit Methoden wie der UML beschrieben werden sollen. Einerseits kann eine natürlich-sprachliche Beschreibung gemäß der Literaturanalyse zu Anforderungsfehlern führen (Beobachtungen 1 und 2). Dies legt nahe, semi-formale Beschreibungsmethoden einzusetzen. Chatzoglou zeigt in einer Untersuchung, dass in Projekten mit Einsatz von Methoden der Aufwand, die Kosten und der Zeitbedarf für die gesamte Entwicklung geringer ausfallen als in Projekten ohne Einsatz von Methoden.[398]

Andererseits können jedoch Kunden Schwierigkeiten haben, semi-formale Beschreibungen von Anforderungen zu verstehen (Beobachtung 3). Dieser Sachverhalt wird ebenfalls in weiteren empirischen Untersuchungen festgestellt.[399] Dem Spannungsfeld zwischen natürlich-sprachlicher und semi-formaler Beschreibung begegnen Entwicklungsteams,

- indem sie semi-formale Beschreibungen von Anforderungen zusätzlich und redundant natürlich-sprachlich für Kunden erläutern,[400]
- indem sie informale Beschreibungen von Anforderungen (überwiegend natürlich-sprachlich) für Kunden und davon getrennt für die Arbeit innerhalb des Entwicklungsteams zusätzlich semi-formale Beschreibungen für Anforderungen erstellen[401] oder
- indem sie sich auf die natürlich-sprachliche Beschreibung beschränken.[402]

Inhalt der Anforderungsdokumente

Neben der Form von Anforderungsdokumenten gibt es mehrere Beobachtungen zum Inhalt. Werden Anforderungen aus technischer Perspektive beschrieben, kann dies zu Anforderungsfehlern führen (Beobachtung 10). Die gleiche Wirkung haben Unklarheiten, in welchem Detailgrad die Anforderungen beschrieben werden sollen (Beobachtungen 5 bis 7). Die Beobachtungen 5 bis 7 und 10 stehen im Einklang mit Ergebnissen einer Untersuchung von El Emam und Madhavji.[403]

Software-Werkzeuge und ihre Nutzung

Wird ein Software-Werkzeug eingesetzt, um Anforderungen zu dokumentieren oder zu verwalten, kann sein Einsatz zu Anforderungsfehlern führen, sofern es Unzulänglichkeiten wie

---

[398] Vgl. Chatzoglou /Methodologies/ 269.

[399] Vgl. El Emam, Madhavji /Requirements Engineering/ 76 und Dawson, Swatman /Requirements engineering/ 269 f.

[400] Vgl. Al-Rawas, Easterbrook /Communication problems/ 51.

[401] Vgl. Dawson, Swatman /Requirements engineering/ 269 f.

[402] Vgl. Chatzoglou /Methodologies/ 259, Nikula, Sajaniemi, Kälviäinen /Requirements Engineering/ 7 f. und Neill, Laplante /Requirements Engineering/ 42-44.

[403] Vgl. El Emam, Madhavji /Requirements Engineering/ 73 f.

eine geringe Benutzbarkeit aufweist (Beobachtungen 11, 12 und 16) oder die Nutzung nicht geplant, gesteuert und überwacht wird (Beobachtungen 13, 14 und 15).
Dies sind einige der Gründe, warum Entwicklungsteams entsprechende Werkzeuge eher als hinderlich wahrnehmen.[404] Empirische Befunde zeigen, dass der Einsatz von entsprechenden Software-Werkzeugen wie z. B. CASE-Werkzeugen nur dann effektiv ist, wenn zumindest ein Prozess der Anforderungsanalyse definiert ist, die Entwickler in dem Werkzeug geschult werden und die Unterstützung der Nutzung organisatorisch institutionalisiert wird.[405]

Verfolgbarkeit von Anforderungen
Insgesamt 10 Beobachtungen beziehen sich auf die Verfolgbarkeit von Anforderungen. Mit 6 Beobachtungen ist die vertikale Vorverfolgbarkeit am häufigsten vertreten. Sie bezieht sich auf Verknüpfungen zwischen einer Anforderung und abhängigen Einzelergebnissen vorgelagerter Teilaufgaben wie z. B. Kundenbedürfnissen. Gotel und Finkelstein kommen in einer Untersuchung zur Verfolgbarkeit von Anforderungen ebenfalls zum Ergebnis, dass Praktiker eine unzureichende vertikale Vorverfolgbarkeit von Anforderungen am häufigsten als Problem nennen.[406]

Benutzerbeteiligung
Die identifizierten Ursachen in der Benutzerbeteiligung sind vor allem eine unzureichende oder zu späte Benutzerbeteiligung (Beobachtungen 27 bis 33) oder eine Beteiligung von ungeeigneten Personen (Beobachtungen 34 bis 39). Prozessmängel in der Benutzerbeteiligung verursachen insbesondere missverständliche Anforderungen.
Empirische Befunde einer Untersuchung von Weltz und Ortmann zeigen ebenfalls auf, dass die Weichen für die Benutzerbeteiligung häufig falsch gestellt werden.[407] Gemäß ihrer Untersuchung spielen sowohl Entwickler als auch Benutzer in der Formulierung von Anforderungen häufig nur eine marginale Rolle. Nach ihrer Ansicht wird ein Benutzervertreter eher deswegen bestimmt, um die tatsächlich unzulängliche Einbindung der Benutzer zu verdecken. Vergleichbare Beobachtungen machen auch El Emam und Madhavji.[408] Sie bemerken, dass gerade jene Benutzer, deren Beteiligung am sinnvollsten für ein Projekt ist, selten für entsprechende Aufgaben zur Verfügung stehen.

---

[404] Vgl. Juristo, Moreno, Silva /Requirements Engineering/ 73, Hofmann, Lehner /Requirements Engineering/ 62
[405] Vgl. El Emam, Madhavji /Requirements Engineering/ 78.
[406] Vgl. Gotel, Finkelstein /Requirements traceability/ 99.
[407] Vgl. hierzu und im Folgenden Weltz, Ortmann /Softwareprojekt/ 72-80.
[408] Vgl. hierzu und zum nächsten Satz El Emam, Madhavji /Requirements Engineering/ 75.

Die Beobachtungen 40 bis 43 beschreiben Schwierigkeiten bei der interdisziplinären Zusammenarbeit zwischen Entwicklern und Benutzern sowie mangelnde Konsequenz bei der Umsetzung der Benutzerbeteiligung. In einer 1994 veröffentlichten Untersuchung zur Benutzerbeteiligung[409] fasst Heinbokel zusammen, dass die „Benutzerbeteiligung in der gegenwärtig praktizierten Form weder zu ersichtlich besserer Software führt noch für den Entwicklungsprozess selbst förderlich ist, sondern im Gegenteil zunächst ein Hemmschuh für das Projekt ist“[410]. Heinbokel kommt zu dieser Schlussfolgerung, weil Projekte mit einer Benutzerbeteiligung beim Gesamterfolg, bei der Termin- und Kosteneinhaltung sowie bei der Teameffektivität schlechter abschneiden als Projekte ohne Benutzerbeteiligung.[411]

Wissen über das Anwendungsgebiet

Insgesamt 12 Beobachtungen weisen darauf hin, dass die Entstehung von Anforderungsfehlern begünstigt wird, wenn dem Entwicklungsteam Wissen über das Anwendungsgebiet fehlt (Beobachtungen 47 bis 58). Prozessmängel in diesem Bereich verursachen gemäß der Literaturanalyse abgesehen von redundanten Anforderungen alle Typen von Anforderungsfehlern.

Tiwana kommt auf der Grundlage einer Untersuchung von 232 Entwicklungsprojekten in 232 verschiedenen Organisationen zum Ergebnis, dass die Integration von technischem Wissen und Wissen über das Anwendungsgebiet der Software unter anderem die Fehlerhäufigkeit in den einzelnen Teilaufgaben der Softwareentwicklung reduziert.[412]

Erhebung von Anforderungen

Die Erhebung von Anforderungen verursacht gemäß der Literaturanalyse 13% der Anforderungsfehler. Insbesondere treten eine unklare Ziele der Software (Beobachtungen 59 bis 62) und die unzureichende Auseinandersetzung mit Kundenbedürfnissen (Beobachtungen 63 bis 67) als Ursachen auf. Prozessmängel in der Erhebung von Anforderungen verursachen insbesondere fehlende Anforderungen.

Eine Fallstudie von Cusumano und Selby bei Microsoft verdeutlicht, wie wichtig eine klare Vision und klare Ziele für die Entwicklung einer Software sind.[413] Microsoft nutzt eine Visionsaussage und eine vorläufige Spezifikation als Projektleitfaden. Statt von Anfang an eine

---

[409] Nach Heinbokel liegt eine Benutzerbeteiligung vor, wenn Benutzer oder Benutzervertreter Mitglied des Projektteams sind (vgl. Heinbokel /Benutzerbeteiligung/ 108 f.). Diese Definition schließt folglich den wechselseitigen Informationsaustausch als eine Form der Benutzerbeteiligung aus (vgl. hierzu Kapitel 4.3.4.3.2).

[410] Heinbokel /Benutzerbeteiligung/ 119.

[411] Vgl. Heinbokel /Benutzerbeteiligung/ 116-119.

[412] Vgl. Tiwana /Knowledge integration/ 904 f.

[413] Vgl. zu diesem Absatz Cusumano, Selby /Microsoft-Methode/ 210-217.

detaillierte Spezifikation zu schreiben, beginnt Microsoft die Projekte mit einer ehrgeizigen Visionsaussage und einer vorläufigen Spezifikation. Die Visionsaussage wird von den Produktplanern der Marketinggruppe und den Programm-Managern verfasst und listet eine Reihe von Zielen auf. Sie enthält keine detaillierten Anforderungen, sondern wird vielmehr als ein Ausgangspunkt genutzt, um systematisch Anforderungen herzuleiten.

Die Beobachtungen 63 bis 67 beschreiben eine unzureichende Auseinandersetzung mit Kundenbedürfnissen als Ursachen für Anforderungsfehler. Dieser Umstand rührt unter anderem daher, dass Entwicklungsteams Methoden zur systematischen Erhebung von Anforderungen[414] wie z. B. Quality Function Deployment (QFD)[415] nicht kennen oder nicht einsetzen.[416] Vielmehr existiert häufig eine unberechtigte Erwartung, dass Methoden zur Beschreibung von Anforderungen zugleich auch zu ihrer Erhebung dienen.[417] Mehrere Untersuchungen zeigen, dass erfolgreiche Teams konsequent Methoden zur Erhebung von Anforderungen wie z. B. QFD einsetzen.[418]

[414] Christel, Kang /Requirements elicitation/ gibt einen ausführlichen Überblick über die Erhebung von Anforderungen, insbesondere Herausforderungen und verfügbare Methoden.

[415] Die Anwendung von QFD für die Softwareentwicklung wird ausführlich behandelt in Herzwurm, Schockert, Mellis /QFD/.

[416] Vgl. Haag, Raja, Schkade /Quality Function Deployment/ 45.

[417] Vgl. Lubars, Potts, Richter /Requirements modeling/ 7.

[418] Vgl. z. B. Haag, Raja, Schkade /Quality Function Deployment/ 45-47 und Hofmann, Lehner /Requirements engineering/ 64

4.4 Erklärungsmodell

Warum entstehen Anforderungsfehler? Dies ist die Ausgangsfrage des Kapitels 4. Nachdem Fehler und ihre Merkmale dargestellt (Kapitel 4.1), das Konstrukt eines Prozessmangels eingeführt (in Kapitel 4.2) und Beobachtungen zu Prozessmängeln der Anforderungsanalyse aus der empirischen Literatur erhoben worden sind (Kapitel 4.3), werden diese Erkenntnisse in dem vorliegenden Kapitel zu einem Erklärungsmodell integriert. Dieses Erklärungsmodell ist ein gedanklicher Bezugsrahmen[419] und nutzt Gesetzesmäßigkeiten (nomologische Aussagen), um zu erklären, warum Anforderungsfehler entstehen. Aufgrund der explorativen Ausrichtung der vorliegenden Arbeit haben diese Gesetzesmäßigkeiten hypothetischen Charakter, da ihre Prüfung noch aussteht.[420]

Das Kapitel 4.4.1 erörtert zunächst Merkmale von Gesetzmäßigkeiten in der Wirtschaftsinformatik. Daran schließt die Darstellung des Erklärungsmodells in Kapitel 4.4.2 an.

4.4.1 Merkmale von Gesetzmäßigkeiten in der Wirtschaftsinformatik

In dieser Arbeit wird ein Verfahren hergeleitet, das darauf abzielt, Gestaltungsempfehlungen für eine bestimmte Klasse von praktischen Problemen zu geben. Hierzu muss die Wirkung alternativer Gestaltungsempfehlungen aus nomologischen Aussagen (Gesetzmäßigkeiten) inhaltlich prognostiziert werden.[421] Nomologische Aussagen geben also Hinweise, wie durch eine zielgerichtete Gestaltung angestrebte Zustände erreicht werden können.

Eine Gesetzmäßigkeit hat oder kann in eine Wenn-Dann-Form überführt werden, die besagt unter welchen Bedingungen (Wenn-Teil) regelmäßig bestimmte Ereignisse (Dann-Teil) eintreten.[422] Bedingungen können sowohl Rahmenbedingungen sein, die nicht beeinflusst werden können, als auch Faktoren, auf die gestalterisch Einfluss genommen werden kann.

Gesetzmäßigkeiten in der Wirtschaftsinformatik unterscheiden sich von Gesetzmäßigkeiten in den Naturwissenschaften. Während naturwissenschaftliche Gesetze immer und überall gelten, sind derartige räumlich und zeitlich invariante Beziehungen für die Wirtschaftsinformatik nicht zu erwarten.[423] So können beispielsweise kulturelle und historische Gegebenheiten eine Beziehung zwischen Größen beeinflussen, die für die Wirtschaftsinformatik relevant sind. Es

---

[419] Vgl. hierzu Kapitel 1.3.4.1.

[420] Vgl. Gadenne /Rationalismus/ 5, 14.

[421] Vgl. hierzu und zum folgenden Satz Kapitel 1.3.3 und die in der Fußnote 74 aufgeführten Literaturquellen.

[422] Vgl. Kubicek /Organisationsforschung/ 24 f.

[423] In Anlehnung an Kubicek /Organisationsforschung/ 48. Kubicek erörtert, wie Gesetzmäßigkeiten in der Organisationstheorie zu interpretieren sind. Bestimmte Überlegungen, auf die im Folgenden Bezug genommen wird, sind auf die Wirtschaftsinformatik übertragbar.

ist daher zweckmäßiger von *Quasi-Gesetzen* auszugehen, die sich „stets nur auf einen raumzeitlich abgegrenzten Gegenstandsbereich beziehen können [...] und nur für diesen Bereich Gültigkeit besitzen“[424].

Aufgrund der Komplexität des Untersuchungsgegenstands der Wirtschaftsinformatik kommt erschwerend hinzu, dass in einer nomologischen Aussage nicht alle für die Kausalität relevanten Bedingungen abgedeckt werden können.[425] Infolge hat man *unvollständige* bzw. *partielle* Gesetzmäßigkeiten. Dies ist keine Besonderheit der Wirtschaftsinformatik, sondern ein Umstand, mit der die Sozial- und Wirtschaftswissenschaften im Allgemeinen konfrontiert sind. Gadenne schlägt daher vor, Gesetzmäßigkeiten in der Wirtschaftsinformatik stets als *certeris paribus* zu interpretieren. Dies besagt, dass sich die Gesetzmäßigkeiten auf einen Normalzustand beziehen, bei dem alle relevanten Bedingungen mit Ausnahme der berücksichtigen konstant gehalten werden.

4.4.2 Hypothetische Gesetzmäßigkeiten

Das Erklärungsmodell wird in Form von hypothetischen Gesetzmäßigkeiten beschrieben. Es besteht aus zwei Teilen:

- Einem theoretischen Ansatz, der allgemein anhand hypothetischer Gesetzmäßigkeiten erklärt, warum Anforderungsfehler entstehen.
- Einer Menge von Erklärungsmustern für bestimmte Typen von Anforderungsfehlern. Diese Erklärungsmuster sind einzelne hypothetische Gesetzmäßigkeiten, die erklären, warum bestimmte Typen von Anforderungsfehlern entstehen. Diese Menge von Erklärungsmustern ist an die Ergebnisse der Literaturanalyse aus Kapitel 4.3.4 angelehnt und erhebt nicht den Anspruch auf Vollständigkeit.

Theoretischer Ansatz des Erklärungsmodells

Bisher wurde von dem vereinfachten Modell ausgegangen, dass Prozessmängel der Anforderungsanalyse Anforderungsfehler verursachen und diese für Folgefehler in Entwurf und Implementierung verantwortlich sind (vgl. Abbildung 4-7).

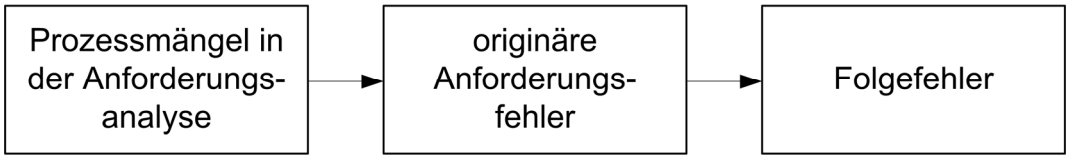


Abbildung 4-7: Erklärungsmodell für Anforderungsfehler und ihre Folgefehler

[424] Kubicek /Organisationsforschung/ 48.

[425] Vgl. zu diesem Absatz Gadenne /Rationalismus/ 15-17 und Kubicek /Organisationsforschung/ 41 f.

Dieses vereinfachte Erklärungsmodell wird auf der Grundlage der erarbeiteten Erkenntnisse in Kapitel 4 präzisiert und in der Abbildung 4-8 dargestellt.

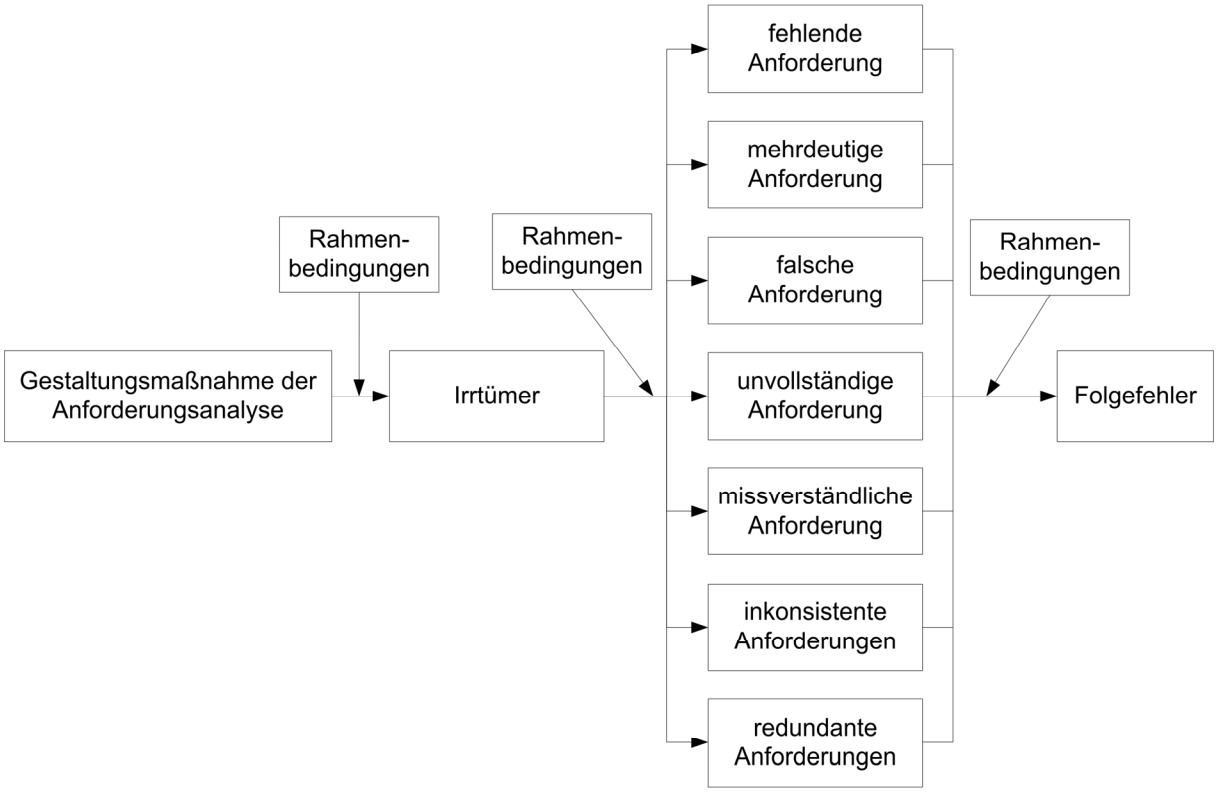


Abbildung 4-8: Theoretischer Ansatz des Erklärungsmodells für Anforderungsfehler und ihre Folgefehler

Ausgehend von dem Verständnis, dass ein Prozess ein Bündel von Gestaltungsmaßnahmen ist, ist ein Prozessmangel der Anforderungsanalyse eine Maßnahme, die *menschliche Irrtümer* begünstigt.[426] Ein Irrtum liegt vor, wenn eine geplante Abfolge von geistigen oder körperlichen Aktivitäten ohne Fremdeinwirkung nicht zum beabsichtigten Ergebnis führt. Voraussetzung für einen Irrtum ist die Absicht zur Handlung. Eine Absicht ist gegeben, wenn dem Individuum das erwartete Ergebnis und die Mittel bewusst sind, mit denen das Ergebnis erreicht werden soll.

Unterliegt ein Projektmitarbeiter einem Irrtum während er Teilaufgaben der Anforderungsanalyse ausführt, kann sein Irrtum dazuführen, dass Anforderungsfehler entstehen. Ein *Fehler* ist ein unzureichendes Merkmal oder ein erwartetes, jedoch fehlendes Merkmal eines Arbeitsergebnisses der Softwareentwicklung, sofern es eine Änderung in diesem Ergebnis notwendig macht. Fehler, die infolge eines Irrtums neu eingeführt werden, sind *originäre Fehler*.

[426] Vgl. zum Folgenden Kapitel 4.2.

Irrtümer in der Anforderungsanalyse verursachen systematisch bestimmte Typen von Anforderungsfehlern. Wenn sich ein Anforderungsfehler auf eine Anforderung bezieht, können insgesamt fünf Fälle voneinander abgegrenzt werden:

- Für ein bestehendes Kundenbedürfnis wurde eine Anforderung gänzlich übersehen (*fehlende Anforderung*).
- Die Beschreibung der Anforderung kann auf mehrere Arten gedeutet werden (*mehrdeutige Anforderung*).
- Die Anforderung erfüllt kein Kundenbedürfnis oder erfüllt sie nicht korrekt (*falsche Anforderung*).[427]
- Die Anforderung ist unvollständig beschrieben (*unvollständige Anforderung*).
- Die Anforderung ist zwar vollständig beschrieben, jedoch fehlen Kontextinformationen zu dieser Anforderung oder Verknüpfungen zu vorhandenen Kontextinformationen, um die Anforderung zu verstehen (*missverständliche Anforderung*). Kontextinformationen sind z. B.
  - Kundenbedürfnisse, die eine Anforderung begründen,
  - für die Anforderung relevante Kundengruppen oder
  - andere abhängige Anforderungen.

Ein Anforderungsfehler kann sich auch auf eine Beziehung zwischen zwei oder mehreren Anforderungen beziehen. Zwei Fälle können dabei unterschieden werden:

- Eine Anforderung steht im Widerspruch zu einer oder mehreren anderen Anforderungen (*inkonsistente Anforderungen*).
- Eine Anforderung ist redundant zu mindestens einer anderen Anforderung (*redundante Anforderungen*).

Die 7 Typen von Anforderungsfehlern können Folgefehler in Entwurf und Implementierung verursachen, wenn sie nicht frühzeitig entdeckt werden. Diese *Folgefehler* haben in Abgrenzung zu originären Fehlern ihre Ursache in einem Fehler einer vorgelagerten Entscheidung.
Die Wirkung einzelner Gestaltungsmaßnahmen ist abhängig von den Rahmenbedingungen, unter denen sie durchgeführt werden.[428] Folglich sind auch Prozessmängel situativ.[429]

Erklärungsmuster für einzelne Typen von Anforderungsfehlern

Nachfolgend werden Erklärungsmuster für einzelne Typen von Anforderungsfehlern formuliert. Diese Erklärungsmuster basieren auf den Ergebnissen der systematischen Literaturana-

[427] Vgl. hierzu Kapitel 2.2.1.
[428] Vgl. hierzu Abbildung 1-3.
[429] Vgl. hierzu die Fußnote 32 und die dort aufgeführten Literaturquellen.

lyse aus Kapitel 4.3.4. Ein Erklärungsmuster beschreibt jeweils, welche Maßnahme der Anforderungsanalyse einen bestimmten Typ von Anforderungsfehler verursachen kann. Entgegen dem theoretischen Ansatz des Erklärungsmodells (vgl. Abbildung 4-8) beinhalten die Erklärungsmuster weder das Konstrukt des Irrtums noch relevante Rahmenbedingungen, da die untersuchten empirischen Literaturquellen kaum entsprechende Informationen enthielten.

Fehlende Anforderung

Die Ursachen für fehlende Anforderungen liegen insbesondere in der Erhebung und Dokumentation von Anforderungen, der Benutzerbeteiligung und dem Wissen über das Anwendungsgebiet der Software (vgl. Tabelle 4-39 und Abbildung 4-9).

Nr.	Ursache	Wirkung	Fundort
11	Datenbank mit Anforderungen ist schwierig zu durchsuchen.	fehlende Anforderung	Q5, S. 235
13	Datenbank mit Anforderungen ist nicht aktuell.	fehlende Anforderung	Q5, S. 235
21	Begründung einer Entscheidung wird vergessen oder nicht festgehalten.	fehlende Anforderung	Q21, S. 72
27	Es finden nur unzureichend wenige Rücksprachen mit Benutzern statt.	fehlende Anforderung	Q19, S. 144
42	Anforderungen werden vom Kunden an Entwickler nur in mündlicher Form übergeben.	fehlende Anforderung	Q21, S. 70
47	Mangelnde Erfahrungen im Anwendungsgebiet.	fehlende Anforderung	Q1, S. 1357
58	Mangel an Experten des Anwendungsgebiets im Entwicklungsteam.	fehlende Anforderung	Q19, S. 144
63	Bestehende Bedürfnisse werden nicht erkannt.	fehlende Anforderung	Q15, S. 41
64	Entdeckte Bedürfnisse werden ignoriert oder vergessen.	fehlende Anforderung	Q15, S. 41
70	Offene Punkte werden nicht direkt geklärt.	fehlende Anforderung	Q21, S. 72

Tabelle 4-39: Ursachen für fehlenden Anforderungen

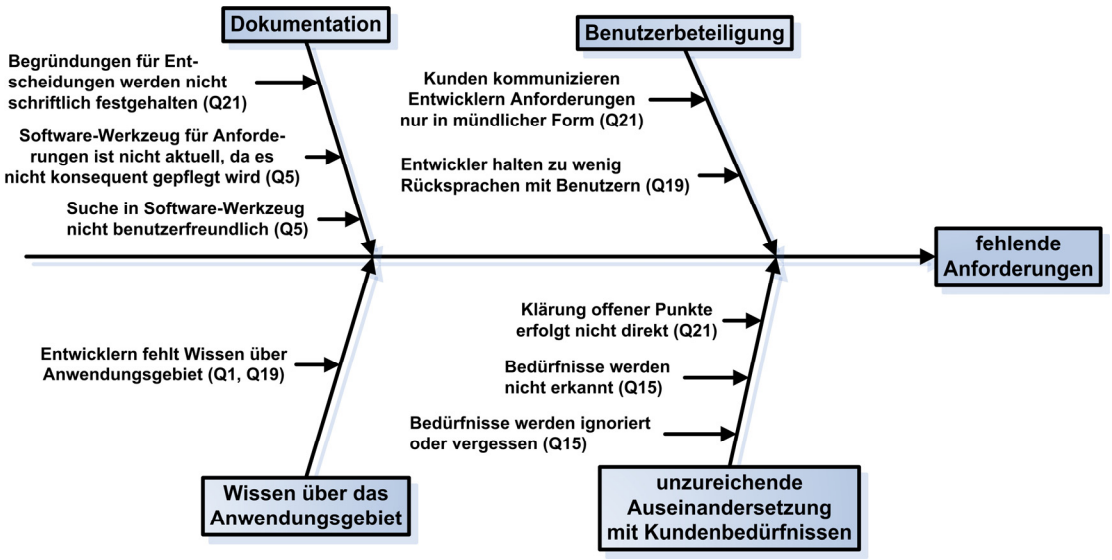


Abbildung 4-9: Ursachen für fehlende Anforderungen

Mehrdeutige Anforderungen

Die Ursachen für mehrdeutige Anforderungen liegen insbesondere in fehlendem Wissen über das Anwendungsgebiet der Software, der späten Klärung offener Punkte und der Gestaltung des Inhalts des Anforderungsdokuments (vgl. Tabelle 4-40 und Abbildung 4-10).

Nr.	Ursache	Wirkung	Fundort
5	Anforderungen werden nicht präzise dokumentiert, sondern oberflächlich in Form von grundlegenden Funktionen der Software kommuniziert.	mehrdeutige Anforderung	Q8, S. 260
48	Mangelnde Erfahrungen im Anwendungsgebiet.	mehrdeutige Anforderung	Q1, S. 1359
53	Wissen über spezielles Anwendungsgebiet erforderlich.	mehrdeutige Anforderung	Q5, S. 234
69	Entwickler und Benutzer sehen nicht die Notwendigkeit, mehrdeutige Anforderungen umgehend zu klären.	mehrdeutige Anforderung	Q4, S. 47 f.

Tabelle 4-40: Ursachen für mehrdeutige Anforderungen

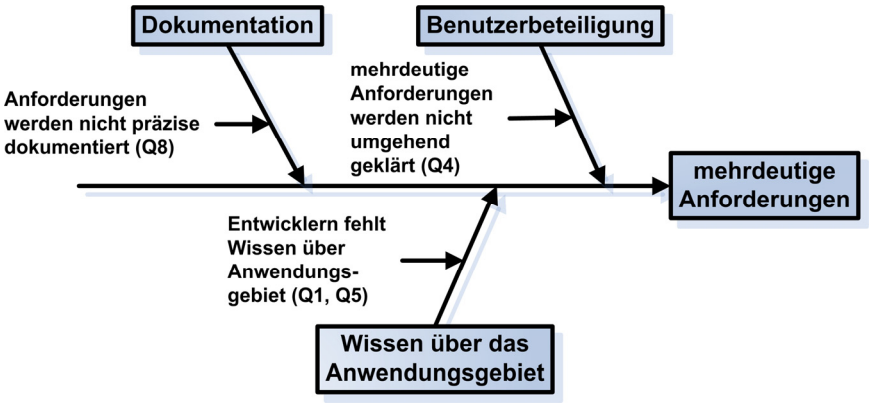


Abbildung 4-10: Ursachen für mehrdeutige Anforderungen

Falsche Anforderungen

Die Ursachen für falsche Anforderungen liegen insbesondere in dem Gestaltungsbereichen der Dokumentation von Anforderungen, Wissensmonopolen bei wenigen Entwicklern, einer unzureichenden Auseinandersetzung mit Kundenbedürfnissen und der Vielzahl und Heterogenität der Kunden (vgl. Tabelle 4-41 und Abbildung 4-11).

Nr.	Ursache	Wirkung	Fundort
12	Datenbank mit Anforderungen ist schwierig zu durchsuchen.	falsche Anforderung	Q5, S. 235
14	Datenbank mit Anforderungen ist nicht aktuell.	falsche Anforderung	Q5, S. 235
25	Begründung einer Anforderung wird nicht dokumentiert.	falsche Anforderung	Q10, S. 32
46	Entwickler vertrauen übermäßig auf ihr eigenes Wissen über das Anwendungsgebiet.	falsche Anforderung	Q5, S. 234
65	Softwareanforderungen werden unzureichend validiert.	falsche Anforderung	Q10, S. 31
72	Konflikte zwischen verschiedenen Kunden.	falsche Anforderung	Q16, S. 5

Tabelle 4-41: Ursachen für falsche Anforderungen

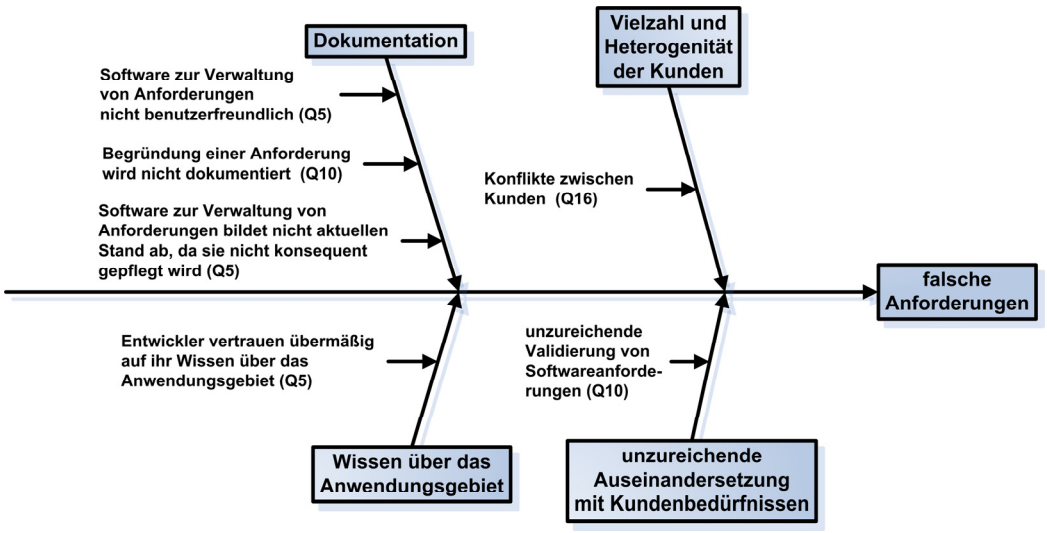


Abbildung 4-11: Ursachen für fehlende Anforderungen

Unvollständige Anforderungen

Die Ursachen für unvollständige Anforderungen liegen insbesondere im Projektmanagement, der späten Klärung offener Punkte, fehlendem Wissen über das Anwendungsgebiet der Software und der Benutzerbeteiligung (vgl. Tabelle 4-42 und Abbildung 4-12).

Nr.	Ursache	Wirkung	Fundort
43	Es ist nicht klar definiert, welche Rechten und Pflichten Benutzer bei der Prüfung von Dokumenten haben.	unvollständige Anforderung	Q4, S. 47
52	Wissen über spezielles Anwendungsgebiet erforderlich.	unvollständige Anforderung	Q5, S. 234
68	Entwickler und Benutzer sehen nicht die Notwendigkeit, unvollständige Anforderungen umgehend zu klären.	unvollständige Anforderung	Q4, S. 47 f.
83	Es unklar, wie ein Änderungsantrag für Anforderungen geschrieben werden soll, der sowohl für Entwickler als auch Benutzer geeignet ist.	unvollständige Anforderung	Q4, S. 47
86	Ein definierter Entwicklungsprozess wird nicht eingehalten.	unvollständige Anforderung	Q4, S. 47

Tabelle 4-42: Ursachen für unvollständige Anforderungen

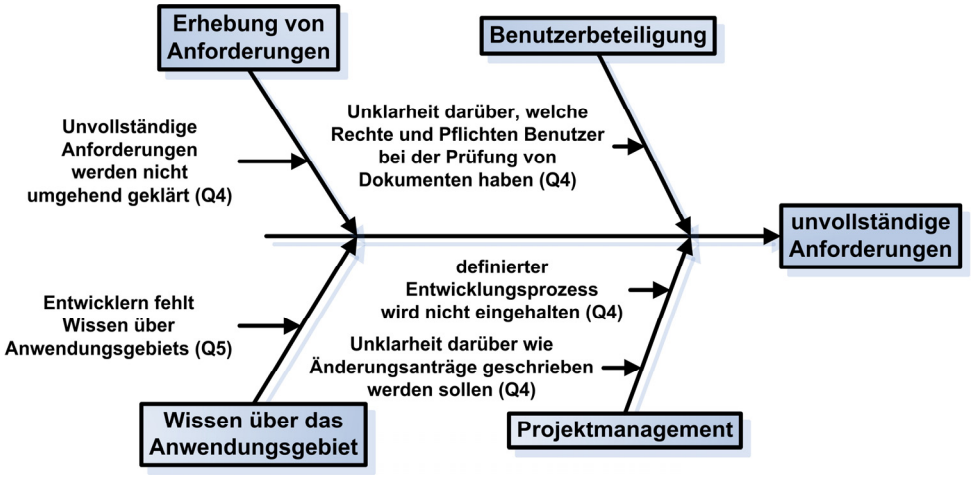


Abbildung 4-12: Ursachen für unvollständige Anforderungen

Missverständliche Anforderung

Die Ursachen für missverständliche Anforderungen liegen insbesondere in den Gestaltungsbereichen der Dokumentation von Anforderungen (hier insbesondere die Verfolgbarkeit) und der Benutzerbeteiligung (vgl. Tabelle 4-43 und Abbildung 4-13).

Nr.	Ursache	Wirkung	Fundort
2	Anforderungen sind natürlich-sprachlich beschrieben.	missverständliche Anforderung	Q2, S. 51
4	Softwareanforderungen werden unangemessen dokumentiert.	missverständliche Anforderung	Q18, S. 835
16	Anforderungen werden nicht vollständig verstanden, da ihre Begründung in einer Datenbank mit Kundenbedürfnissen "vergraben" sind	missverständliche Anforderung	Q7, S. 51
19	Abhängige Anforderungen, die mit der Strukturierten Analyse beschrieben worden sind, sind verteilt im Dokument beschrieben.	missverständliche Anforderung	Q20, S. 71
20	Unangemessene Dokumentation der Beziehungen zwischen Softwareanforderungen und Funktionen der Software.	missverständliche Anforderung	Q18, S. 835
24	Häufig fehlt die Begründung der Anforderungen.	missverständliche Anforderung	Q7, S. 51
28	Entwickler haben nicht ausreichend Kontakt mit Kunden.	missverständliche Anforderung	Q16, S. 5
29	Kunden werden nicht ausreichend eingebunden.	missverständliche Anforderung	Q13, S. 63
30	Entwickler haben keinen Kontakt zu Benutzern.	missverständliche Anforderung	Q19, S. 144
32	Kunden und Entwickler kommunizieren selten.	missverständliche Anforderung	Q19, S. 144
34	Aufgrund von Vermittlern gibt es eine Kommunikationslücke zwischen Kunden und Entwicklern.	missverständliche Anforderung	Q9, S. 156
35	Indirekte Kommunikation mit Kunden bei der Erhebung und Untersuchung der bestehenden Software.	missverständliche Anforderung	Q9, S. 154
56	Entwickler kennt ein Kundenbedürfnis, macht jedoch eine falsche Annahme darüber.	missverständliche Anforderung	Q15, S. 41
85	Eine mangelnde Betreuung des definierten Prozesses der Anforderungsanalyse führt dazu, dass einzelne Teams ihn nicht einhalten.	missverständliche Anforderung	Q9, S. 155
92	Entwickler kommunizieren im Wesentlichen über Dokumente.	missverständliche Anforderung	Q6, S. 1279 f.

Tabelle 4-43: Ursachen für missverständliche Anforderungen

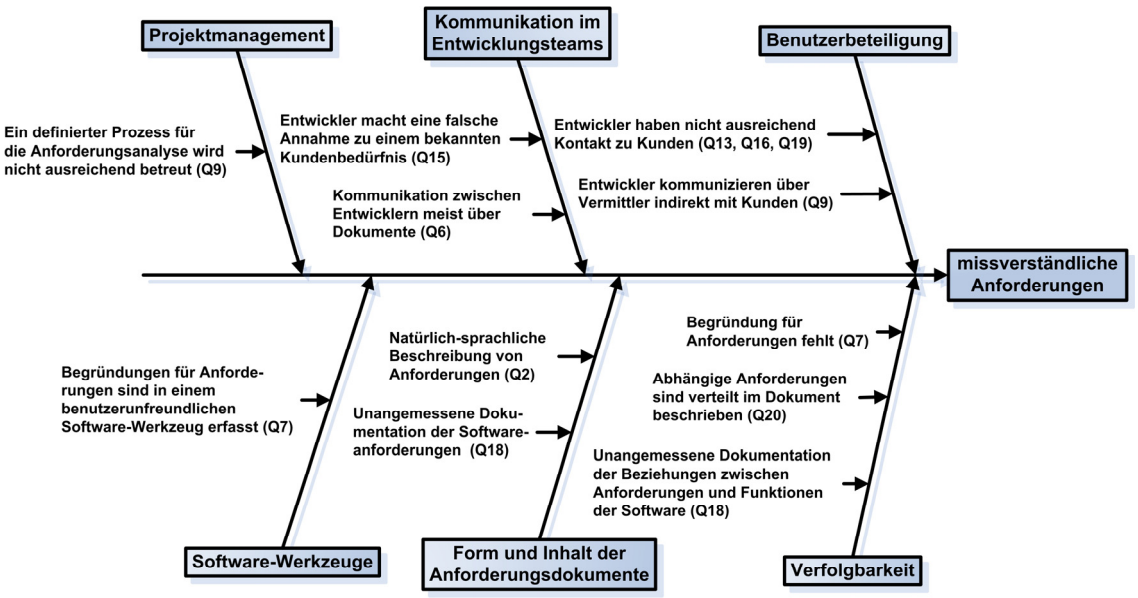


Abbildung 4-13: Ursachen für missverständliche Anforderungen

Inkonsistente Anforderungen

Die Ursachen für inkonsistente Anforderungen liegen in der Dokumentation von Anforderungen, der inhärenten Komplexität der Anforderungsanalyse, der Kommunikation im Entwicklungsteam und fehlendem Wissen über das Anwendungsgebiet (vgl. Tabelle 4-44 und Abbildung 4-14).

Nr.	Ursache	Wirkung	Fundort
6	Anforderungen werden in unterschiedlichem Detail beschrieben.	inkonsistente Anforderungen	Q9, S. 154
8	Anforderungen werden mit einer uneinheitlichen Terminologie beschrieben.	inkonsistente Anforderungen	Q9, S. 154
15	Datenbank mit Anforderungen zeigt Abhängigkeiten zwischen Anforderungen nicht zuverlässig an, weil diese nicht immer eingetragen werden.	inkonsistente Anforderungen	Q5, S. 236
18	Fehlende Verfolgbarkeit erschwert das Einfügen neuer Anforderungen.	inkonsistente Anforderungen	Q2, S. 55
49	Mangelnde Erfahrungen im Anwendungsgebiet.	inkonsistente Anforderungen	Q1, S. 1356
71	Verschiedene Interessensgruppen steuern Anforderungen bei.	inkonsistente Anforderungen	Q17, S. 131
75	Es gibt verschiedene Kunden.	inkonsistente Anforderungen	Q21, S. 64 f.
77	Die Anforderungsdokumente sind umfangreich.	inkonsistente Anforderungen	Q2, S. 50
78	Die Anzahl der Anforderungen wächst explosiv.	inkonsistente Anforderungen	Q6, S. 1277
90	Entwickler treffen verschiedene Annahmen und kommunizieren diese nicht untereinander.	inkonsistente Anforderungen	Q5, S. 236
91	Verschiedene Entwickler machen verschiedene Annahmen bei der Interpretation von Anforderungen.	inkonsistente Anforderungen	Q6, S. 1278

Tabelle 4-44: Ursachen für inkonsistente Anforderungen

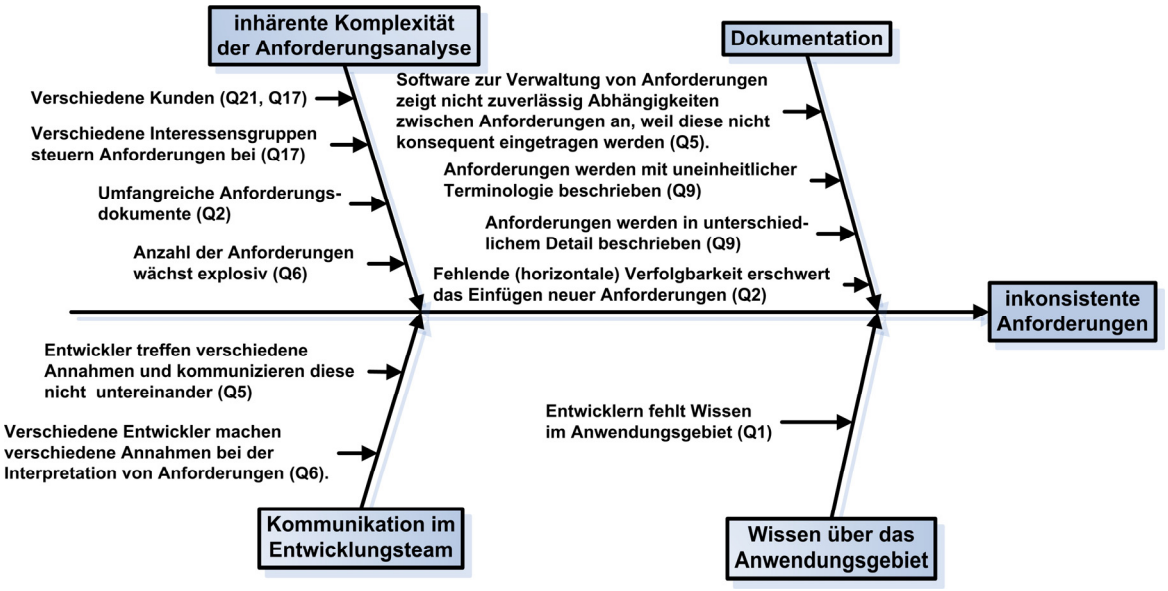


Abbildung 4-14: Ursachen für inkonsistente Anforderungen

Redundante Anforderungen

Die Ursachen für redundante Anforderungen liegen in der Dokumentation von Anforderungen (vgl. Tabelle 4-45).

Nr.	Ursache	Wirkung	Fundort
7	Anforderungen werden in unterschiedlichem Detail beschrieben.	redundante Anforderungen	Q9, S. 154
9	Anforderungen werden mit einer uneinheitlichen Terminologie beschrieben.	redundante Anforderungen	Q9, S. 154

Tabelle 4-45: Ursachen für redundante Anforderungen

Anforderungsfehler allgemein

Die Ursachen für Anforderungsfehler, deren Typ nicht bestimmt werden konnte, liegen in allen Ursachenkategorien mit Ausnahme der internen Kommunikation (vgl. Tabelle 4-46 bis Tabelle 4-51). Die Abbildung 4-15 gibt einen Überblick über diese Ursachen.

Nr.	Ursache	Wirkung	Fundort
1	Natürlich-sprachliche Anforderungsdokumente erschweren die Erkennung einzelner Softwareanforderungen aus dem Text heraus.	Anforderungsfehler	Q14, S. 47
3	Kunden fällt es schwer, Beschreibungen von Anforderungen zu verstehen, wenn ausschließlich Darstellungsmittel eingesetzt werden, die eine grafische Notation nutzen.	Anforderungsfehler	Q2, S. 51
10	Anforderungen werden aus einer technischen Perspektive dokumentiert.	Anforderungsfehler	Q14, S. 45
17	Beziehungen zwischen Anforderungen wurden nicht dokumentiert.	Anforderungsfehler	Q14, S. 47
22	Annahmen über Entscheidungen werden nicht festgehalten.	Anforderungsfehler	Q16, S. 6
23	Beziehungen zwischen geschäftlichen Informationen und Anforderungen fehlen.	Anforderungsfehler	Q14, S. 45
26	Sitzungen werden nicht dokumentiert, so dass die Verfolgbarkeit des Entscheidungsprozesses nicht möglich ist.	Anforderungsfehler	Q19, S. 145

Tabelle 4-46: Dokumentation von Anforderungen - Ursachen für Anforderungsfehler mit unbekanntem Typ

Nr.	Ursache	Wirkung	Fundort
31	Entwicklungsteam hat keinen Kontakt zu Benutzern	Anforderungsfehler	Q14, S. 47
33	Entwicklungsteam hat zu spät Kontakt zu Benutzern	Anforderungsfehler	Q14, S. 47
36	Wahren Benutzer werden nicht beteiligt	Anforderungsfehler	Q19, S. 20
37	Schlüsselpersonen werden nicht einbezogen.	Anforderungsfehler	Q11, S. 556
38	Erfahrene Benutzer haben keine Zeit.	Anforderungsfehler	Q19, S. 20
39	Kunden wählen nicht Mitarbeiter mit dem meisten Wissen im Anwendungsgebiet aus, um mit den Softwareentwicklern zu arbeiten	Anforderungsfehler	Q2, S. 6-8
40	Kunden verstehen selten die Komplexität des Entwicklungsprozesses und verlangen daher häufig Änderungen in den Anforderungen.	Anforderungsfehler	Q6, S. 1276
41	Wenn Kunden direkten Zugriff auf das Entwicklungsteam haben, verlangen sie oft Änderungen in den Anforderungen ohne eine formale Prüfung des Änderungsantrags	Anforderungsfehler	Q6, S. 1276

Tabelle 4-47: Benutzerbeteiligung - Ursachen für Anforderungsfehler mit unbekanntem Typ

Nr.	Ursache	Wirkung	Fundort
44	Es gibt „Kopfmonopole“ an kritischen Stellen.	Anforderungsfehler	Q12, S. 170
45	Informationen über Kunden und Benutzer sind implizites Wissen in den Köpfen erfahrener Personen.	Anforderungsfehler	Q14, S. 45
50	Entwickler verstehen die Bedürfnisse der Benutzer nicht.	Anforderungsfehler	Q4, S. 48
51	Mangelndes Wissen über das Anwendungsgebiet.	Anforderungsfehler	Q4, S. 47
54	Entwickler haben nicht ausreichendes Wissen über das Anwendungsgebiet, um die Absicht des Kunden aus einer Anforderung zu interpretieren.	Anforderungsfehler	Q6, S. 1271
55	Fehlende fachliche Erfahrungen der Projektgruppen-Mitglieder in dem mit dem Projekt verbundenen Geschäftsprozessen.	Anforderungsfehler	Q12, S. 170
57	Bei den Entwicklern fehlt Wissen über das Anwendungsgebiet.	Anforderungsfehler	Q19, S. 145

Tabelle 4-48: Wissen über das Anwendungsgebiet - Ursachen für Anforderungsfehler mit unbekanntem Typ

Nr.	Ursache	Wirkung	Fundort
59	Business Case wurde nicht gründlich bewertet.	Anforderungsfehler	Q11, S. 556
60	Eine klare Vision des Produkts ist erforderlich, um sie in klare Produktanforderungen zu übersetzen.	Anforderungsfehler	Q6, S. 1277
61	Vage Produktvision und –strategie	Anforderungsfehler	Q11, S. 556
62	Fehlende vollständige, eindeutige und durchführbare Zielvorgaben	Anforderungsfehler	Q12, S. 170
66	Entwickler gelingt es nicht inkonsistente Kundenaussagen aufzulösen.	Anforderungsfehler	Q15, S. 41
67	Kunden wünschen Anforderungen, die unzweckmäßig sind oder ihre wahren Bedürfnisse nicht erfüllen	Anforderungsfehler	Q16, S. 6

Tabelle 4-49: Erhebung von Anforderungen - Ursachen für Anforderungsfehler mit unbekanntem Typ

Nr.	Ursache	Wirkung	Fundort
73	Es gibt sehr verschiedene Kunden.	Anforderungsfehler	Q3, S. 191 f.
74	Verschiedene Kunden haben verschiedene Bedürfnisse.	Anforderungsfehler	Q6, S. 1276
76	Die Anzahl der Anforderungen erschwert ihre Validierung	Anforderungsfehler	Q17, S. 131
79	Kunden wünschen nachträglich zusätzliche Anforderungen, sobald sie im Projektverlauf besser verstehen, was die geplante Software leisten soll.	Anforderungsfehler	Q6, S. 1276
80	Die Bedürfnisse eines Kunden ändern sich im Verlauf der Zeit.	Anforderungsfehler	Q6, S. 1276

Tabelle 4-50: Inhärente Komplexität - Ursachen für Anforderungsfehler mit unbekanntem Typ

Nr.	Ursache	Wirkung	Fundort
81	Ein definierter Prozess der Anforderungsanalyse fehlt.	Anforderungsfehler	Q3, S. 203
82	Ein Anforderungsdokument existiert nicht und einzelne Teamleiter pflegen eigene Listen mit Anforderungen.	Anforderungsfehler	Q3, S. 191 f.
84	Anforderungen werden nicht ausreichend beschrieben und analysiert.	Anforderungsfehler	Q11, S. 556
87	Es existieren unbekannte Projektabhängigkeiten.	Anforderungsfehler	Q11, S. 556
88	Parallele Projekte beeinflussen sich gegenseitig.	Anforderungsfehler	Q12, S. 170
89	Das anfängliche Projektteam ist mehr daran interessiert, eine Ausschreibung zu gewinnen als präzise die erforderlichen Kosten und Ressourcen zu schätzen	Anforderungsfehler	Q6, S. 1276

Tabelle 4-51: Projektmanagement - Ursachen für Anforderungsfehler mit unbekanntem Typ

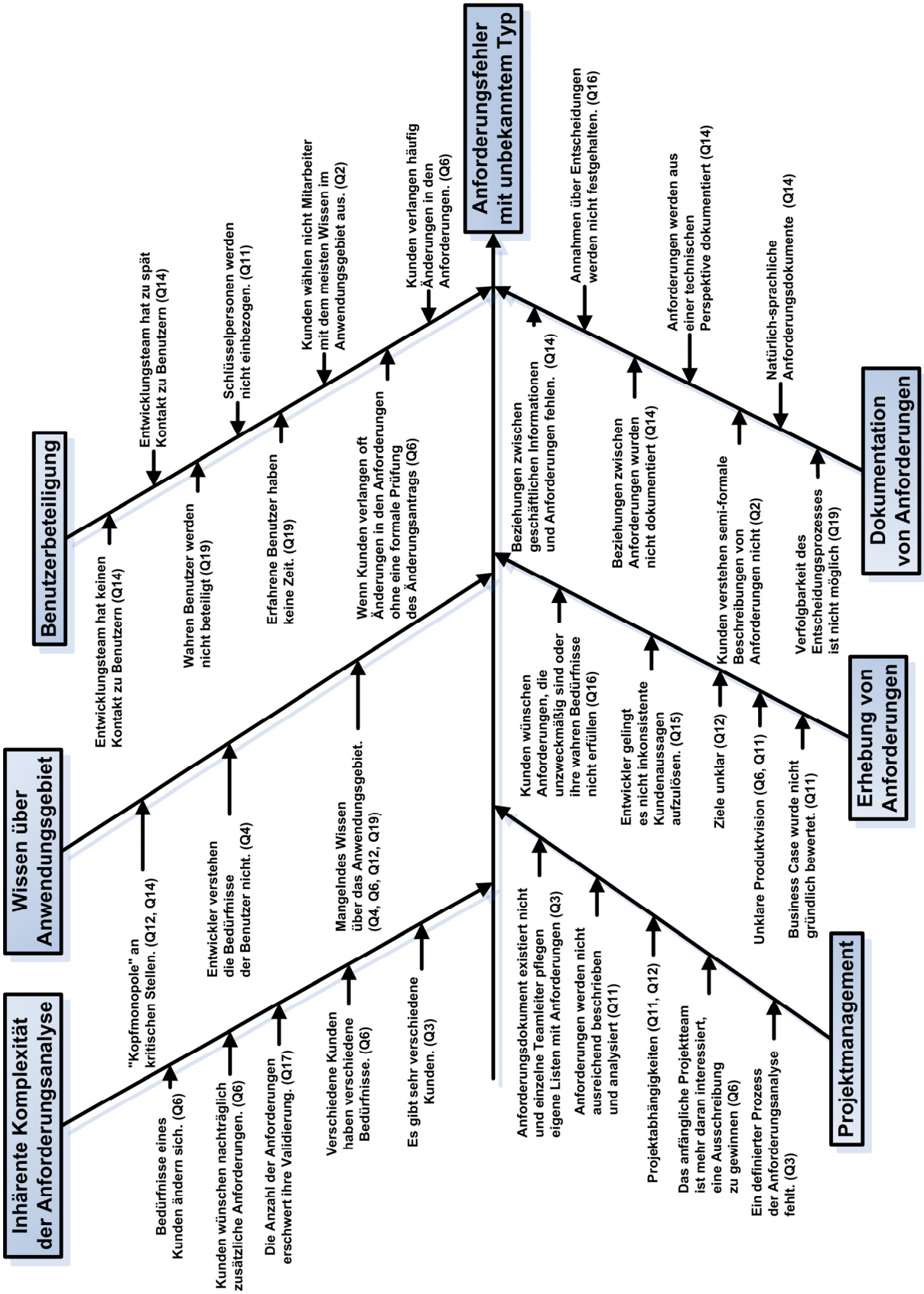


Abbildung 4-15: Ursachen für Anforderungsfehler mit unbekanntem Typ

5 Darstellung und kritische Würdigung der Orthogonal Defect Classification

5.1 Darstellung der Orthogonal Defect Classification

5.1.1 Einführung

Fast 20-jährige Geschichte von ODC

Die Orthogonal Defect Classification, kurz ODC, ist ein fehlerbasiertes Verfahren zur Verbesserung eines Softwareentwicklungsprozesses.[430] Sie wurde ursprünglich entwickelt, um einem Entwicklungsteam in einem laufenden Softwareentwicklungsprojekt Feedback über den Prozess und die Qualität der Software zur Verfügung zu stellen. Bisweilen wird sie aber zusätzlich und zum Teil auch ausschließlich auf nach Abschluss eines Projekts entdeckte Fehler angewandt, um für zukünftige Projekte Verbesserungen abzuleiten.[431]

Die Entwicklung von ODC beginnt 1989 bei IBM unter maßgeblicher Beteiligung von Ram Chillarege.[432] Zwischen 1990 und 1991 wird das Verfahren überarbeitet und in mehreren Pilotprojekten bei IBM untersucht. Ab 1992 beginnt die Umsetzung in mehreren IBM-Softwareentwicklungszentren. Seither wird das Verfahren kontinuierlich verbessert und auch außerhalb von IBM eingesetzt.[433] Gemäß einer Umfrage aus dem Jahre 1999 haben 14 von 37 Unternehmen (ca. 38%), die die vierte oder fünfte Reifestufe nach CMM erlangt haben, ODC als einen unternehmensweit als ein Standardverfahren institutionalisier oder setzen sie regelmäßig ein.[434]

Chillarege erhält 2004 für die Erfindung von ODC und seine Beiträge für die Softwaretechnik die Auszeichnung „IEEE Computer Society Technical Achievement Award“.[435]

---

[430] Vgl. hierzu und zum nachfolgenden Satz Halliday u. a. /Experiences/ 61 und Chillarege u. a. /In-process measurements/ 954 f.

[431] Vgl. z. B. Sullivan, Chillarege /Comparison/ und Dalal u. a. /Defect patterns/.

[432] Vgl. zu diesem Absatz Halliday u. a. /Experiences/ 61. Collofello und Balcom beschreiben bereits 1984 allgemein die Ideen, die ODC zu Grunde liegen. Vgl. Collofello, Balcom /Software error classification/ 537-545

[433] Vgl. hierzu die Zusammenfassungen der empirischen Untersuchungen zu ODC in Anhang A.2.

[434] Vgl. Paulk, Goldenson, White /Survey/ 51.

[435] Vgl. http://www.chillarege.com/odc/news/ieee-award.htm, Stand 07.06.2007.

Was verbirgt sich hinter dem Begriff ODC?

Autoren im Umfeld von ODC verwenden den Begriff ODC häufig mehrdeutig, so dass eine Klärung zweckmäßig ist:

- Bedeutung 1: Chillarege definiert für ein Klassifikationsschema[436] zur Beschreibung von Fehlern bestimmte Bedingungen. Diese sollen gewährleisten, dass das Schema eingesetzt werden kann, um Feedback über den Softwareentwicklungsprozess zu erhalten. Jedes Fehlerklassifikationsschema, das diese Bedingungen erfüllt, wird als orthogonale Fehlerklassifikation, also ODC, bezeichnet.[437]
- Bedeutung 2: Der von Chillarege bei der IBM entwickelte spezifische Vorschlag für ein Fehlerklassifikationsschema, dass nahe liegender Weise die von ihm formulierten Bedingungen erfüllt, wird als ODC bezeichnet.[438]
- Bedeutung 3: Unter ODC wird ein Verfahren verstanden, dass das von Chillarege entwickelte Fehlerklassifikationsschema nutzt, und implizit folgende Verfahrensschritte umfasst:[439]
    - *Konfiguration*: Fehlerklassifikationsschema für ein konkretes Softwareentwicklungsprojekt konfigurieren.
    - *Fehler klassifizieren*: Fehler auf der Grundlage des Klassifikationsschemas erfassen.
    - *klassifizierte Fehler auswerten*: Fehlermuster suchen und bewerten.
    - *Ursachenanalyse*: Ursachen für ausgewählte Fehlermuster ermitteln.

In der vorliegenden Arbeit wird unter der Bezeichnung ODC ein Verfahren und damit die Bedeutung 3 verstanden. Die Bedeutung 2 wird mit *(Fehler-)Klassifikationsschema von ODC* oder *ODC-Klassifikationsschema* umschrieben. ODC im Sinne der Bedeutung 1 wird in dieser Arbeit nicht verwendet.

Das Fehlerklassifikationsschema von ODC berücksichtigt zum einen Entwurfsfehler und zum anderen Implementierungsfehler in einem Programmcode. Für die sichtbaren Aspekte von grafischen Oberflächen, Benutzerdokumenten, Übersetzungen von Software sowie der Code-Kompilierung und Bindung an Bibliotheken (Build, Package, Merge) liegen Vorschläge zur Erweiterung des ODC-Klassifikationsschema vor.[440]

---

[436] Vgl. zum Begriff Fehlerklassifikationsschema 3.4.1.

[437] Vgl. Chillarege u. a. /In-process measurements/ 944-946, 955 und Chillarege /Defect/ 370-374.

[438] Vgl. Chillarege /Defect/ 376 und IBM Research /ODC/.

[439] Vgl. Butcher, Munro, Kratschmer /Software testing/ 32: , Bhandari u. a. /Case study/ 1159-1163 und Chaar u. a. /In-process evaluation/ 1058.

[440] Vgl. IBM Research /ODC Extensions/ o. S.

Nachfolgend wird in Kapitel 5.1.2 das ODC-Klassifikationsschema dargestellt. Das Schema ist insbesondere in IBM Research /ODC/ ausführlich dokumentiert. Die Verfahrensschritte von ODC sind dagegen nicht einheitlich beschrieben. Vielmehr werden einzelne Verfahrensschritte in verschiedenen Quellen in unterschiedlicher Detailtiefe erläutert. Dies erfolgt meist exemplarisch anhand von Fallbeispielen, so dass für eine allgemeine Beschreibung in bestimmten Bereichen offene Punkte bleiben.

5.1.2 ODC-Klassifikationsschema

Ein Fehlerklassifikationsschema besteht aus einer Menge von Attributen. Ein Attribut ist ein als relevant erachtetes Merkmal eines Fehlers.[441] Wird ein Klassifikationsschema angewandt, also Fehler klassifiziert, werden für jeden einzelnen Fehler die Attribute mit in der Regel vorgegebenen Werten gemessen. Nachfolgend wird das ODC-Klassifikationsschema beschrieben.

5.1.2.1 Attribute und Attributwerte

Das ODC-Klassifikationsschema erhebt nicht den Anspruch, ein umfassendes Klassifikationsschema für Fehler zu sein.[442] Es kann vielmehr als Erweiterung zu einem bestehenden Klassifikationsschema für Fehler eingesetzt werden. Das Klassifikationsschema umfasst lediglich die Attribute, die ODC benötigt, um das selbst gesetzte Ziel zu erreichen: Verbesserung eines Softwareentwicklungsprozesses, indem die Ursachen für Fehler aufgedeckt werden. ODC setzt dabei nicht einmal voraus, dass alle Attribute des Klassifikationsschemas eingesetzt werden müssen, um entscheidungsrelevante Erkenntnisse zu gewinnen.[443]

Das ODC-Klassifikationsschema besteht in der Version 5.11 aus insgesamt acht Attributen,[444] die alle nominal skaliert sind.[445] Die zu erfassenden Attribute zu einem Fehler sind nach zwei Zeitpunkten gruppiert:[446]

- Attribute, deren Ausprägung frühestens bestimmt werden kann, nachdem ein Fehler *entdeckt* worden ist.

---

[441] Vgl. Fenton, Pfleeger /Software metrics/ 5.

[442] Vgl. Halliday u. a. /Experiences/ 63 f.

[443] Vgl. Chillarege, Prasad /ODC Triggers/ 669. Implizit wird aber angenommen, dass mindestens ein kausales Attribut eingesetzt wird. Vgl. hierzu Kapitel 5.1.2.3.1.

[444] Vgl. IBM Research /ODC/ o. S.

[445] Vgl. Fenton, Pfleeger /Software metrics/ 47-49.

[446] Vgl. zu Folgendem IBM Research /ODC/ o. S.

- Attribute, deren Ausprägung frühestens bestimmt werden kann, nachdem ein Fehler *behoben* worden ist.

Die Darstellung des ODC-Klassifikationsschemas erfolgt in IBM Research /ODC/ wie folgt:

- Der Inhalt jedes Attributs wird kurz erläutert.
- Jeder Wert eines Attributs wird in der Regel in ein bis zwei Sätzen erläutert.
- Zu jedem Wert werden ein bis zwei Beispiele aufgeführt.

Die nachfolgende Darstellung verzichtet auf eine detaillierte Erläuterung der einzelnen Attribute und führt auch nicht die Beispiele aus IBM Research /ODC/ an, da die Darstellung lediglich einen Einblick in das ODC-Klassifikationsschema geben soll, so dass die weiteren Ausführungen zu ODC angemessen eingeordnet werden können. Zur ausführlichen Darstellung sei auf IBM Research /ODC/ verwiesen.

Abweichend von IBM Research /ODC/ werden die Inhalte der Attribute nachfolgend in Form von einfachen Fragen wiedergegeben, um ihr Verständnis zu erleichtern. Die Attribute und ihre Werte werden sinngemäß und nicht wortwörtlich ins Deutsche übersetzt. Die englische Bezeichnung sowohl der Attribute als auch ihrer Werte werden jeweils zusätzlich in Klammern aufgeführt. Die Darstellung erfolgt in tabellarischer Form.

ODC-Attribute nach einer Fehlerentdeckung

Drei ODC-Attribute zu einem Fehler können erfasst werden, sobald der Fehler entdeckt worden ist. Diese sind die Attribute Aktivität (activity), Auslöser (trigger) und Auswirkung (impact).[447]

Attributname	Aktivität (activity)
Frage	In welcher QS-Aktivität wurde der Fehler gefunden?
Attributwerte	Der Wertebereich dieses Attributs ist projektspezifisch und muss konfiguriert werden.

Tabelle 5-1: ODC-Attribut Aktivität (activity) und Werte

[447] Vgl. zu Folgendem IBM Research /ODC/ o. S. und Butcher, Munro, Kratschmer /Software testing/ 42 f.

Attributname	Auslöser (trigger)
Frage	Welche Bedingungen beziehungsweise Kriterien haben dazugeführt, dass der Fehler entdeckt wurde?
Attributwerte	▪ Entwurfskonformität (design conformance) ▪ Kontroll- und Datenfluss (logic / flow) ▪ Abwärtskompatibilität (backward compatibility) ▪ Umfeldkompatibilität (lateral compatibility) ▪ Steuerung gemeinsam genutzter Ressourcen (concurrency) ▪ Entwicklerdokumentation (internal document) ▪ Besonderheiten der Entwicklungssprache (language dependency) ▪ Nebenwirkung (side effect) ▪ seltene Situationen (rare Situations) ▪ einfacher Pfad (simple path) ▪ komplexer Pfad (complex path) ▪ Funktionsabdeckung (coverage) ▪ Funktionsvariation (variation) ▪ Funktionensequenz (sequencing) ▪ Funktioneninteraktion (interaction) ▪ Leistungs- oder Stresstest (workload / stress) ▪ Fehlerbehandlung / Ausnahmebehandlung (recovery / exception) ▪ Start / Neustart (startup / restart) ▪ Hardwarekonfiguration (hardware configuration) ▪ Softwarekonfiguration (software configuration) ▪ blockierter Test (blocked test)

Tabelle 5-2: ODC-Attribut Auslöser (trigger) und Werte

Attributname	Auswirkung (impact)
Frage	In welchem Bereich hätte der Fehler eine Einschränkung nach Auslieferung der Software bewirkt?
Attributwerte	▪ Installierbarkeit (installability) ▪ Analysierbarkeit von Fehlverhalten (serviceability) ▪ Standards für Softwareprodukte (standards) ▪ Integrität / Sicherheit (integrity / security) ▪ Versionswechsel (migration) ▪ Zuverlässigkeit (reliability) ▪ Zeitverhalten (performance) ▪ Benutzerdokumentation in der Software (documentation) ▪ unbekannte Softwareanforderung (requirements) ▪ Wartbarkeit (maintenance) ▪ Benutzbarkeit (usability) ▪ Barrierefreiheit (accessibility) ▪ bekannte Softwareanforderung (capability)

Tabelle 5-3: ODC-Attribut Auswirkung (impact) und Werte

ODC-Attribute nach einer Fehlerbehebung

Sobald ein Fehler behoben wurde, können fünf weitere ODC-Attribute erfasst werden. Diese sind im Einzelnen Ziel (target), Fehlertyp (defect type), Kennzeichner (qualifier), Ursprung (source) und Alter (age).[448]

Das Attibut Ziel (target) beschreibt, ob das ODC-Klassifikationsschema für Entwurfs- und Implementierungsfehler oder eine ODC-Erweiterung wie z. B. die Klassifikation von Fehlern in der Übersetzung von Software angewandt wird.[449] Sofern, wie in der vorliegenden Arbeit, nur das ODC-Klassifikationsschema für Entwurfs- und Implementierungsfehler berücksichtigt wird, hat das Attribut nur den Wert *Entwurf / Code (design / code)*. Es wird trotz der für diesen Fall geringen Aussagekraft aufgeführt, da dieses Attribut in einigen empirischen Untersuchungen zu ODC angepasst genutzt wird.[450]

Attributname	Ziel (target)
Frage	Welche Art von Arbeitsergebnissen enthielt den Fehler?
Attributwerte	▪ Entwurf / Code (design / code)

Tabelle 5-4: ODC-Attribut Ziel (target) und Werte

Attributname	Fehlertyp (defect type)
Frage	Was musste geändert werden, um den Fehler zu beheben?
Attributwerte	▪ Zuweisung / Initialisierung (assignment / initialization) ▪ Bedingung (checking) ▪ Algorithmus / Methode (algorithm / method) ▪ Funktion / Klasse / Objekt (function / class / object) ▪ Timing / Serialisierung (timing / serialization) ▪ interne Schnittstellen (interface / O-O Messages) ▪ Beziehung (relationship)

Tabelle 5-5: ODC-Attribut Fehlertyp (defect type) und Werte

Attributname	Kennzeichner (qualifier)
Frage	Wie wurde der Fehler behoben?
Attributwerte	▪ Etwas Fehlendes wurde eingefügt (missing) ▪ Etwas Bestehendes war falsch und wurde korrigiert (incorrect) ▪ Etwas Bestehendes war irrelevant und wurde entfernt (extraneous)

Tabelle 5-6: ODC-Attribut Kennzeichner (qualifier) und Werte

[448] Vgl. zu Folgendem IBM Research /ODC/ o. S. und Butcher, Munro, Kratschmer /Software testing/ 42 f.

[449] Vgl. hierzu Kapitel 5.1.1.

[450] Vgl. z. B. Lutz, Mikulski /Anomalies/ 175.

Attributname	Ursprung (source)
Frage	Woher stammt der fehlerbehaftete Teil des Arbeitsergebnisses ursprünglich?
Attributwerte	▪ Eigenentwicklung (developed in-house) ▪ Wiederverwendung aus Standard-Bibliothek (reused from library) ▪ Entwicklung durch externen Auftragnehmer (outsourced) ▪ portiert (ported)

Tabelle 5-7: ODC-Attribut Ursprung (source) und Werte

Attributname	Alter (age)
Frage	Welche Vorgeschichte weist der Fehler auf?
Attributwerte	▪ alter Fehler (base) ▪ neuer Fehler (new) ▪ Fehler infolge einer Überarbeitung (rewritten) ▪ Fehler infolge einer Fehlerkorrektur (refixed)

Tabelle 5-8: ODC-Attribut Alter (age) und Werte

5.1.2.2 Bedingungen an ein Fehlerklassifikationsschema

Nach Chillarege muss ein Fehlerklassifikationsschema zwei allgemeine Bedingungen erfüllen, wenn es einem Entwicklungsteam in einem laufenden Projekt Feedback liefern soll, das entscheidungsrelevant für die Verbesserung des Softwareentwicklungsprozesses ist.
Die erste Bedingung beschreibt Chillarege wie folgt:

> *„There exists a semantic classification of defects such that its distribution, as a function of process activities, changes as the product advances through the process.“*[451]
>
> oder in leicht abgewandelter Form
>
> *„There exists a semantic classification of defects from a product, such that the defect classes can be related to the process that can explain the process of the product through this process.“*[452]

Die **erste Bedingung** beinhaltet zwei Forderungen. Zum einen soll ein Klassifikationsschema auf einem Ursache-Wirkungszusammenhang zwischen einem Prozess und der Qualität einer Software aufbauen und die Entwicklung der Qualität im Verlauf des Prozesses abbilden können.[453] Zum anderen soll dies über eine *semantische Klassifikation* von Fehlern bewerkstelligt werden.

[451] Chillarege /Defect/ 371.
[452] Chillarege u. a. /In-process measurements/ 945.
[453] Vgl. zu diesem Absatz Chillarege u. a. /In-process measurements/ 945.

Was versteht Chillarege unter einer semantischen Klassifikation? Zum besseren Verständnis ist es zweckmäßig, zunächst das Gegenstück einer semantischen Klassifikation zu beleuchten. Die *meinungsbasierte Klassifikation* von Fehlern stellt einem Softwareentwickler direkt die Frage, was die Ursache eines Fehlers ist.[454] Dies äußert sich beispielsweise in einem Attribut, dessen Wert angeben soll, in welcher Teilaufgabe ein Fehler eingeführt worden ist. Ein Softwareentwickler muss einen vollständigen Überblick über alle Teilaufgaben einer Softwareentwicklung besitzen und den Prozess bis auf die Ursache des Fehlers zurückverfolgen können, wenn er dieses Attribut für einen Fehler mit einem Wert belegen soll. Bei einem arbeitsteiligen Vorhaben wie der Softwareentwicklung ist dies nur begrenzt zu erfüllen, so dass ein Entwickler tendenziell eher eine subjektive Meinung über die Ursache für einen Fehler äußert. Dieser Klassifikation mangelt es folglich an Genauigkeit und Aussagekraft.
Im Gegensatz dazu werden bei einer *semantischen Klassifikation* Ereignisse bestimmter Arbeitsschritte in einem Prozess erfasst.[455] Ein Beispiel ist das ODC-Attribut Fehlertyp (defect type) mit der Fragestellung „Was musste geändert werden, um den Fehler zu beheben?“. Dieses kann ein Softwareentwickler genauer beantworten, weil es sich auf eine kürzlich abgeschlossene Tätigkeit bezieht. In einem zweiten Schritt müssen bei einer semantischen Klassifikation die Werte eines Attributs durch Schlüsselpersonen des Projekts mit Prozessschritten verknüpft werden. Die Ursache eines Fehlers wird also indirekt ermittelt.
Zusammenfassend fordert die erste Bedingung, dass ein Ursache-Wirkungszusammenhang abgebildet wird. Nach Chillarege impliziert die erste Bedingung, dass die Werte eines Attributs jeweils unabhängig voneinander sind (Orthogonalität).[456]

Die zweite Bedingung erläutert Chillarege wie folgt:

> *„The set of all values of defect attributes must form a spanning set over the sub-space.“*[457]

Mit der **zweiten Bedingung** fordert Chillarege, dass die Werte eines Attributs mit dem Prozess verknüpft sind, über den Rückschlüsse gezogen werden sollen. Insbesondere sollen die Werte so gebildet werden, dass erforderliche Rückschlüsse gezogen werden können. Die Werte sollen zudem alle Möglichkeiten abdecken, die sich als Antwort auf die hinter einem Attribut stehende Fragestellung ergeben können.

---

[454] Vgl. zu diesem Absatz Chillarege u. a. /In-process measurements/ 945 f. und Chillarege /Defect/ 371-373.
[455] Vgl. zu diesem Absatz Chillarege u. a. /In-process measurements/ 945 f. und Chillarege /Defect/ 371-373.
[456] Vgl. Chillarege u. a. /In-process measurements/ 946.
[457] Chillarege u. a. /In-process measurements/ 946.

Es bleibt festzuhalten, dass Chillarege die erste und zweite Bedingung implizit nicht an alle Attribute eines Fehlerklassifikationsschemas stellt. Vielmehr fordert er, dass ein Fehlerklassifikationsschema mindestens ein Attribut enthält, das beide Bedingungen erfüllt.
Zusätzlich formuliert Chillarege drei weitere Bedingungen für ein Fehlerklassifikationsschema. Diese gelten implizit für alle Attribute des Klassifikationsschemas und sollen sicherstellen, dass aus den Erfahrungen verschiedener Projekte gelernt werden kann. Diese Bedingungen lauten im Einzelnen:[458]

- Bedingung 3: Die Werte jedes Attributes sind jeweils unabhängig voneinander (Orthogonalität). Attribute erfüllen diese Bedingung bereits, sofern sie die Bedingung 1 erfüllen.
- Bedingung 4: Die Attribute und ihre Werte sind unabhängig vom eingesetzten Softwareentwicklungsprozess.[459]
- Bedingung 5: Die Attribute und ihre Werte sind unabhängig von der Art der entwickelten Software.

5.1.2.3 Umsetzung der Bedingungen im ODC-Klassifikationsschema

5.1.2.3.1 Unterschiedliche Arten von Attributen

ODC besteht aus drei Arten von Attributen:[460]

- *Kausale Attribute* sind Attribute, die die erste und zweite Bedingung an ein Fehlerklassifikationsschema gemäß Kapitel 5.1.2.2 erfüllen. D. h. insbesondere, dass diese Attribute die Wirkung eines Prozesses abbilden und eine Verknüpfung mit dem Prozess ermöglichen.
- *Wirkungs-Attribute* sind Attribute, die die Wirkung eines Prozesses abbilden, jedoch für sich alleine nicht die erste und zweite Bedingung an ein Fehlerklassifikationsschema gemäß Kapitel 5.1.2.2 erfüllen
- *Kontext-Attribute* sind Attribute, die den Kontext eines Fehlers beschreiben und die Einteilung der Fehlermenge in Teilpopulationen ermöglichen.

Die Tabelle 5-9 führt auf, welcher Art die acht ODC-Attibute zuzuordnen sind.

---

[458] Vgl. zum Folgenden Chillarege u. a. /In-process measurements/ 945.
[459] Attribute, die die Bedingung 2 erfüllen, genügen per Definition auch der Bedingung 4.
[460] Vgl. Chillarege u. a. /In-process measurements/ 946.

ODC-Attribut	Art des Attributes
Aktivität (activity)	Kontext-Attribut
Auslöser (trigger)	kausales Attribut
Auswirkung (impact)	Wirkungs-Attribut
Ziel (target)	Kontext-Attribut
Fehlertyp (defect type)	kausales Attribut
Kennzeichner (qualifier)	Wirkungs-Attribut
Ursprung (source)	Kontext-Attribut
Alter (age)	Kontext-Attribut

Tabelle 5-9: Art der ODC-Attribute

Als kausale Attribute haben die ODC-Attribute Fehlertyp (defect type) und Auslöser (trigger) eine besondere Bedeutung im ODC-Klassifikationsschema. Sie müssen zwei Bedingungen genügen (vgl. Kapitel 5.1.2.2):

- Sie bauen zum einen mittels einer semantischen Klassifikation ein Ursache-Wirkungszusammenhang zwischen einem Prozess und der Qualität einer Software auf und bilden zum anderen die Entwicklung der Qualität im Verlauf des Prozesses ab.
- Ihre Werte ermöglichen jeweils Rückschlüsse über einen Prozess.

Inwieweit nach Chillarege die Attribute Fehlertyp (defect type) und Auslöser (trigger) diese Bedingungen erfüllen, ist Gegenstand der nachfolgenden Ausführungen.

5.1.2.3.2 Kausale Attribute Fehlertyp (defect type) und Auslöser (trigger)

ODC-Attribut Fehlertyp (defect type)

Dem ODC-Attribut Fehlertyp (defect type) liegt die Frage zu Grunde, was geändert werden musste, um einen Fehler zu beheben. Der Wertebereich dieses Attributs ist fest vorgegeben und umfasst sieben mögliche Werte (vgl. Tabelle 5-10).[461]

[461] Vgl. IBM Research /ODC/ o. S. und Chillarege /Defect/ 374 f.

Attributwert	Bedeutung des Attributwertes
Zuweisung / Initialisierung	Wenige Codezeilen wurden geändert: eine Zuweisung von Werten, die Initialisierung von Kontrollstrukturen oder Datenstrukturen fehlte oder war falsch. Sofern mehrere Änderungen dieser Art durchgeführt werden mussten, die zusammenhängen, liegt ein Algorithmusfehler [Algorithmus/Methode] vor.
Bedingung	Die Überprüfung von Parametern oder Daten in Bedingungen fehlte oder war falsch. Änderungen müssen sich nicht nur auf eine Bedingung beschränken, sondern können auch zur Folge haben, dass ein Block von Anweisungen für einen neuen Bedingungszweig eingefügt wird.
Algorithmus / Methode	Aufgrund von Problemen bzgl. der Effizienz oder Korrektheit wurde ein Algorithmus oder eine lokale Datenstruktur geändert, die nicht die Änderung einer Entscheidung des Grobentwurfs erfordert.
Funktion / Klasse / Objekt	Ein Funktionsfehler beeinträchtigt eine vom Kunden erwartete Funktion, Benutzerschnittstellen, externe Programmierschnittstellen der Software (API), Schnittstellen mit der Hardwarearchitektur oder globale Datenstrukturen.
Timing / Serialisierung	Die Abstimmung des Zugriffs auf eine gemeinsam genutzte Ressource fehlt oder ist fehlerhaft.
interne Schnittstellen	Änderungen sind erforderlich aufgrund von Problemen bei der Kommunikation zwischen ▪ Modulen, ▪ Komponenten, ▪ Gerätetreibern, ▪ Objekten, ▪ Prozeduren mittels ▪ Makros, ▪ Prozeduraufrufen, ▪ Kontrollblöcken, ▪ Parameterlisten.
Beziehung	Änderungen sind erforderlich aufgrund von Problemen in Beziehungen zwischen Prozeduren, Datenstrukturen oder Objekten.

Tabelle 5-10: Werte des ODC-Attributs Fehlertyp (defect type) und ihre Bedeutung

Das ODC-Attribut Fehlertyp (defect type) ist eine semantische Klassifikation, da es eine Tätigkeit beschreibt, die bei der Korrektur eines Fehlers erfolgt. Um die erste Bedingung vollständig zu erfüllen, muss dem ODC-Attribut Fehlertyp (defect type) ein Ursache-Wirkungszusammenhang zwischen einem Prozess und der Qualität der Software zu Grunde liegen. Hierzu verweist Chillarege auf die mit seinen Kollegen 1991 veröffentliche empirische Untersuchung Chillarege, Kao, Condit /Defect type/. Dieser Verweis erweist sich als diskussionsbedürftig, da sie streng genommen nicht einen Zusammenhang zwischen einem Prozess und der Qualität einer Software an sich untersucht, sondern zwischen den Werten des ODC-Attributs Fehlertyp (defect type) und der Qualität einer Software in Form von Zuverlässigkeitswachstumskurven.[462] Eine Wachstumskurve beschreibt jeweils den Verlauf der kumulativen Anzahl der Fehler über die Zeit gemessen in Kalendertagen. Durch die Unterscheidung von Fehlerwirkungen werden Fehlerteilmengen, deren Zuverlässigkeitswachstumskurven sich ähneln, zu

[462] Vgl. hierzu und zum Folgenden Chillarege, Kao, Condit /Defect type/ 246-254.

Teilpopulationen vereint. Als Ergebnis entstehen insgesamt vier verschiedene Teilpopulationen von Fehlern, die unterschiedliche Zuverlässigkeitswachstumskurven aufweisen. Für jede Teilpopulation wird eine repräsentative Anzahl von Fehlern nachträglich nach dem ODC-Attribut Fehlertyp klassifiziert. Schließlich kann für jede Teilpopulation von Fehlern die prozentuale Verteilung auf die Werte des ODC-Attributs Fehlertyp (defect type) bestimmt werden. Hier zeigt sich, dass Teilpopulationen von Fehlern, deren Zuverlässigkeitswachstumskurve eine starke Beugung aufweisen, in hohem Maße mit dem Fehlertypen „Zuweisung / Initialisierung“ verknüpft sind. Die Autoren begründen dies damit, dass Initialisierungsfehler früh in einem Codeausführungspfad auftreten und deshalb weitere, abhängige Fehler maskiert werden.[463] Dieses Ergebnis werten die Autoren als Nachweis, dass messbare Ursache-Wirkungszusammenhänge existieren.

Gegenstand der Untersuchung Chillarege, Kao, Condit /Defect type/ ist aber nicht ein Ursache-Wirkungszusammenhang zwischen einem Prozess und den Werten des ODC-Attributs Fehlertyp (defect type). Dieser Zusammenhang wird als Annahme formuliert, jedoch weder empirisch untersucht noch sachlogisch begründet: „[Defect type] needs to be mapped to the process. This mapping provides the relation between defect types and the process […].“[464] Chillarege macht einen Vorschlag, mit welchen Teilprozessen die einzelnen Werte des ODC-Attributs Fehlertyp (defect type) verknüpft werden können.[465] Es bleibt offen, ob diese Verknüpfungen für jedes Projekt neu zu definieren sind oder der Vorschlag allgemeingültigen Charakter hat:

> *„If a function defect is found, whether it be in system test or unit test, it still points to the high-level design phase that the defect should be associated with. Similarly, a timing error would be associated with low level design.“*[466]

Chillareges Vorschlag sieht folgende Verbindungen zwischen den Werten des ODC-Attributs Fehlertyp (defect type) und Prozessen der Teilaufgaben Grobentwurf, Feinentwurf und Implementierung vor:

---

[463] Vgl. Chillarege, Kao, Condit /Defect type/ 252-254.

[464] Chillarege u. a. /In-process measurements/ 945.

[465] Vgl. Chillarege u. a. /In-process measurements/ 948-951.

[466] Chillarege u. a. /In-process measurements/ 947.

	Teilprozesse		
Attributwert	Grobentwurf	Feinentwurf	Implementierung
Zuweisung / Initialisierung			x
Bedingung		x	x
Algorithmus / Methode		x	
Funktion / Klasse / Objekt	x		
Timing / Serialisierung		x	
interne Schnittstellen		x	
Beziehung			x

Tabelle 5-11: Verknüpfungen zwischen Fehlertypen und Teilprozessen

Zusammenfassend kann festgehalten werden, dass gemäß Chillarege das ODC-Attribut Fehlertyp (defect type) die erste Bedingung erfüllt. Hierzu führt er auf:

- Der Fehlertyp ist per Definition eine semantische Klassifikation.
- Chillarege unterstellt, dass der Fehlertyp einen Ursache-Wirkungszusammenhang zwischen einem Prozess und der Qualität der Software misst. Diese Hypothese besteht aus zwei Teilhypothesen (vgl. Abbildung 5-1). Chillarege nimmt an, dass jeder Fehlertyp mit einem Teilprozess verknüpft werden kann (Teilhypothese 1). Die Untersuchung Chillarege, Kao, Condit /Defect type/ weist nach, dass zwischen den Werten des ODC-Attributs Fehlertyp (defect type) und der Qualität einer Software in Form von Zuverlässigkeitswachstumskurven Ursache-Wirkungszusammenhänge existieren (Teilhypothese 2).

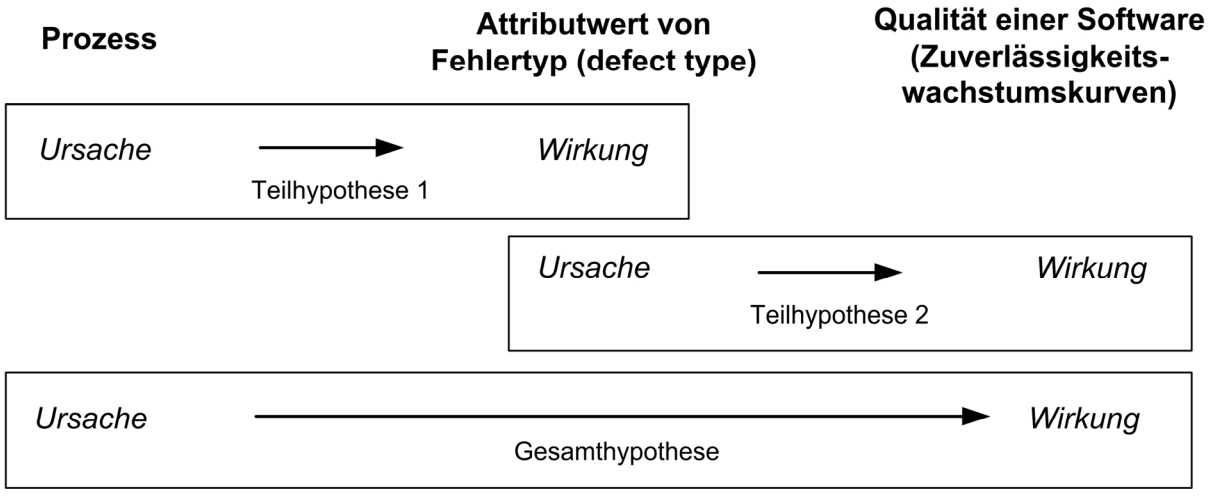


Abbildung 5-1: ODC-Attribut Fehlertyp und Ursache-Wirkungszusammenhang

Gemäß Chillarege erfüllt das ODC-Attribut Fehlertyp (defect type) auch die zweite Bedingung.[467] Dies impliziert zum einen, dass die Werte dieses Attributs unabhängig voneinander

[467] Vgl. Kapitel 5.1.2.2.

sind. Dies würde sich insbesondere darin äußern, dass die Klassifikation von Fehlern nach dem ODC-Attribut Fehlertyp (defect type) wiederholbar, d. h. unabhängig von einzelnen Personen ist. El Emam und Wieczorek bescheinigen diesem Attribut als Ergebnis eines Feldexperiments eine gute Wiederholbarkeit.[468]

Zum anderen müssen die Werte eines Attributs mit dem Prozess verknüpfbar sein, über den Rückschlüsse gemacht werden sollen. Dies unterstellt Chillarege ebenfalls ohne sachlogische Begründung mit Verweis auf seinen Vorschlag zu Verknüpfungen zwischen Fehlertypen und Teilprozessen (vgl. Tabelle 5-11). [469]

ODC-Attribut Auslöser (trigger)

Dem ODC-Attribut Auslöser (trigger) liegt die Frage zu Grunde, welche Bedingungen beziehungsweise Kriterien dazugeführt haben, dass der Fehler entdeckt wurde. Die Werte dieses Attributs stellen Kriterien für das Prüfen oder Testen[470] eines Untersuchungsobjekts dar.[471] Der Wertebereich dieses Attributs ist fest vorgegeben und umfasst die in Tabelle 5-24 aufgeführten Werte.[472] Entgegen der Darstellung in IBM /ODC/ wird die Bedeutung der Prüf- und Testkriterien in Form von Fragen wiedergegeben.

[468] Vgl. El Emam, Wieczorek /Repeatability/ 330.

[469] Vgl. Chillarege u. a. /In-process measurements/ 948-951.

[470] Vgl. Kapitel 2.4.2.

[471] Vgl. für detaillierte Definition und Abgrenzung des ODC-Attributs Auslöser (trigger) Chillarege, Bassin /Software triggers/ 329-331.

[472] Vgl. IBM Research /ODC/ o. S. und Chillarege, Bassin /Software triggers/ 331-335.

Attributwert	Bedeutung des Attributwertes
Entwurfskonformität	Wurde das Prüfobjekt dahingehend untersucht, ob es mit vorhergehenden Arbeitsergebnissen und für das Projekt gültigen Entwicklungsstandards konform ist?
Kontroll- und Datenfluss	Wurde das Prüfobjekt dahingehend untersucht, ob der Kontroll- und Datenfluss im Prüfobjekt korrekt und vollständig sind?
Abwärtskompatibilität	Wurde das Prüfobjekt dahingehend untersucht, ob die im Prüfobjekt beschriebenen Funktionen abwärtskompatibel mit früheren Versionen des Produkts sind?
Umfeldkompatibilität	Wurde das Prüfobjekt dahingehend untersucht, ob die beschriebenen Funktionen im Prüfobjekt kompatibel zu Softwareprodukten im Umfeld sind, mit denen sie interagieren müssen?
Steuerung gemeinsam genutzter Ressourcen	Wurde das Prüfobjekt dahingehend untersucht, ob die Steuerung gemeinsam genutzter Ressourcen abgestimmt ist?
Entwicklerdokumentation	Wurde das Prüfobjekt dahingehend untersucht, ob die Entwicklerdokumentation (z. B. Kommentare im Code) im oder zum Prüfobjekt eine falsche, widersprüchliche oder unvollständige Information enthält?
Besonderheiten der Entwicklungssprache	Wurde das Prüfobjekt dahingehend untersucht, ob Besonderheiten der Entwicklungssprache bei der Implementierung einer Komponente oder Funktion berücksichtigt sind?[473]
Nebenwirkung	Wurde das Prüfobjekt dahingehend untersucht, ob es bei einem gewöhnlichen Anwendungsfall unerwünschte Nebenwirkungen im Verhalten der Software verursacht, die außerhalb des Prüfobjekts liegen?
seltene Situation	Wurde das Prüfobjekt dahingehend untersucht, ob es bei einem nicht vorgesehenen Anwendungsfall unerwünschtes Verhalten der Software verursacht?
einfacher Pfad	Wurde das Testobjekt mit Testfällen getestet, um jeweils einen bestimmten Pfad im Kontrollfluss auszuführen?
komplexer Pfad	Wurde das Testobjekt mit Testfällen getestet, um mehrere Zweige unter verschiedenen Bedingungen auszuführen?
Funktionsabdeckung	Wurde die Software mit Testfällen getestet, um einzelne Funktionen jeweils ohne Eingabeparameter oder falls erforderlich mit genau einer Ausprägung von Eingabeparametern auszuführen?
Funktionsvariation	Wurde die Software mit Testfällen getestet, um einzelne Funktionen jeweils mit unterschiedlichen Ausprägungen von Eingabeparametern auszuführen?
Funktionensequenz	Wurde die Software mit Testfällen getestet, um mehrere Funktionen hintereinander in einer Sequenz auszuführen? Dieser Auslöser darf nur ausgewählt werden, wenn die Funktionen einzeln erfolgreich ausgeführt werden können.
Funktioneninteraktion	Wurde die Software mit Testfällen getestet, um mehrere Funktionen auszuführen, die miteinander interagieren? Die Interaktion beschränkt sich nicht darauf, dass sie in einer Sequenz ausgeführt werden. Dieser Auslöser darf nur ausgewählt werden, wenn die Funktionen einzeln erfolgreich ausgeführt werden können.
Leistungs- oder Stresstest	Wurde die Software mit Testfällen getestet, um das Verhalten der Software bei geringer und hoher Auslastung zu untersuchen?
Fehlerbehandlung / Ausnahmebehandlung	Wurde die Software mit Testfällen getestet, um das Verhalten der Software im Fall eines Fehlverhaltens oder einer Ausnahmesituation zu untersuchen?
Start / Neustart	Wurde die Software mit Testfällen getestet, um das Verhalten der Software beim Start oder Neustart zu untersuchen, nachdem sie ausgefallen oder beendet wurde?
Hardwarekonfiguration	Wurde die Software mit Testfällen getestet, um das Verhalten der Software unter verschiedenen Hardware-Konfigurationen zu untersuchen?
Softwarekonfiguration	Wurde die Software mit Testfällen getestet, um die Software unter verschiedenen Softwarekonfigurationen zu untersuchen?
blockierter Test	Wurde die Durchführung eines Testfalls im Systemtest beabsichtigt, jedoch durch das Auftreten eines Fehlers verhindert, der außerhalb des Fokus des Testfalls liegt?

Tabelle 5-12: Werte des ODC-Attributs Auslöser (trigger) und ihre Bedeutung

[473] Eine Besonderheit einer Entwicklungssprache ist zum Beispiel, dass in C statt dem Vergleichsoperator == versehentlich der Zuweisungsoperator = genutzt wird.

Chillarege definiert fünf allgemeine Qualitätssicherungsprozesse und ordnet die Werte des ODC-Attributs Auslöser (trigger) diesen Aktivitäten zu (vgl. Tabelle 5-13). Die QS-Aktivitäten sind im Einzelnen:[474]

- Prüfaktivität *Entwurfsreview* (design review): Entwurfsergebnisse werden geprüft.
- Prüfaktivität *Code-Inspektion* (code inspection): Programmcode wird geprüft.
- Teststufe *Komponententest* (unit test): Die nach der Implementierung erstellten Softwarebausteine werden einzeln gestestet.
- Teststufe *Funktionstest* (function test): Die Funktionalität der Software wird gestestet.
- Teststufe *Systemtest* (system test): Die integrierte Software wird möglichst unter den auf dem Zielsystem (Hardware und Software) geltenden Rahmenbedingungen getestet.

QS-Prozess	Zugeordnete Auslöser
Entwurfsreview	▪ Entwurfskonformität ▪ Kontroll- und Datenfluss ▪ Abwärtskompatibilität ▪ Umfeldkompatibilität ▪ Steuerung gemeinsam genutzter Ressourcen ▪ Entwicklerdokumentation ▪ Besonderheiten der Entwicklungssprache ▪ Nebenwirkung ▪ seltene Situation
Code-Inspektion	Die gleichen Auslöser wie für den QS-Prozess Entwurfsreview.
Komponententest	▪ einfacher Pfad ▪ komplexer Pfad
Funktionstest	▪ Funktionsabdeckung ▪ Funktionsvariation ▪ Funktionensequenz ▪ Funktioneninteraktion
Systemtest	▪ Leistungs- oder Stresstest ▪ Fehlerbehandlung / Ausnahmebehandlung ▪ Start / Neustart ▪ Hardwarekonfiguration ▪ Softwarekonfiguration ▪ blockierter Test

Tabelle 5-13: Verknüpfungen zwischen Auslösern und QS-Prozessen

Im Unterschied zu Verknüpfungen zwischen Fehlertypen und Teilprozessen (vgl. Tabelle 5-11) betont Chillarege, dass der Vorschlag für die Verknüpfungen zwischen Auslösern und QS-Prozessen als Beispiel zu verstehen ist und projektspezifisch angepasst werden muss.[475]

[474] Vgl. Chillarege, Bassin /Software triggers/ 331-335.

Das ODC-Attribut Auslöser (trigger) erfüllt nach Chillarege die erste Bedingung für Klassifikationsschemata gemäß Kapitel 5.1.2.2.[476] Demnach sei es unter anderem eine semantische Klassifikation. Im Fall der Qualitätssicherung in einem laufenden Projekt muss ein Prüfer oder Tester festhalten, nach welchem Kriterium geprüft oder getestet worden ist, als der Fehler entdeckt wurde. Auslöser sind folglich eng an die jeweilige QS-Tätigkeit gekoppelt. Die Klassifikation von Fehlern, die nach Auslieferung der Software durch Benutzer entdeckt werden, wird etwas anders bearbeitet.[477]

Die erste Bedingung erfordert des Weiteren, dass dem Attribut Auslöser (trigger) ein Ursache-Wirkungszusammenhang zwischen einem Prozess und der Qualität der Software zu Grunde liegt. Chillarege macht einen Vorschlag für Verknüpfungen zwischen Auslösern und QS-Prozessen (vgl. Tabelle 5-13) mit dem Hinweis, dass diese Verknüpfungen projektspezifisch angepasst werden müssen. Dieser Zusammenhang wird aber nicht weiter sachlogisch untermauert. Ein Zusammenhang zwischen dem ODC-Attribut Auslöser (trigger) und der Qualität der Software wird nur indirekt aufgebaut. Chillarege führt auf, dass nach Auslieferung entdeckte Fehler im Idealfall ausschließlich nur Auslöser haben sollten, die der Teststufe Systemtest zugeordnet worden sind (vgl. Tabelle 5-13).[478] Diese Aussage kann verallgemeinert werden: je mehr Fehler zu einem bestimmten Zeitpunkt existieren, deren Auslöser zeitlich vorgelagerten QS-Prozessen zugeordnet werden können, desto unreifer ist die Software. Chillarege und Bassin zeigen in einer explorativen Fallstudie auf, dass auch anhand des ODC-Attributs Auslöser die Entwicklung der Qualität der Software nach ihrer Auslieferung im Zeitverlauf abgebildet werden kann.[479] Bassin, Kratschmer und Santhanam demonstrieren in einer Untersuchung für ein laufendes Projekt, wie mit Werten des Attributs Auslöser die Entwicklung der Qualität einer Software im Verlauf des Prozesses abgebildet werden kann.[480]

Gemäß Chillarege erfüllt das ODC-Attribut Auslöser (trigger) auch die zweite Bedingung.[481] Dies impliziert zum einen, dass die Werte dieses Attributs unabhängig voneinander sind. Eine

---

[475] Vgl. IBM Research /ODC/ o. S.: "The table shown […] gives a generic example of an Activity to Trigger mapping and is not intended to be the actual mapping used by all organizations. One of the first things an organization must do once they have decided to implement ODC is to define the activities they perform and map the triggers to those activities. Note that although the organization defines their activities, they do not define or redefine the triggers."

[476] Vgl. Chillarege, Bassin /Software triggers/ 330 f.

[477] Vgl. hierzu Kapitel 5.1.3.3.

[478] Vgl. Chillarege, Bassin /Software triggers/ 336.

[479] Vgl. Chillarege, Bassin /Software triggers/ 335-341.

[480] Vgl. Bassin, Kratschmer, Santhanam /Software development/ 68.

[481] Vgl. Kapitel 5.1.2.2.

empirische Untersuchung dahingehend, ob die Klassifikation von Fehlern diesem Attribut wiederholbar ist, liegt nicht vor. Diese Annahme wird folglich indirekt nur von empirischen Untersuchungen gestützt, in denen das Attribut Auslöser angewandt und keine Probleme festgestellt wurden.[482]

Zum anderen müssen die Werte eines Attributs mit dem Prozess verknüpfbar sein, über den Rückschlüsse gemacht werden sollen. Im Fall des ODC-Attributs Auslöser (trigger) erfolgt die Verknüpfung mit QS-Prozessen. Dies unterstellt Chillarege ohne sachlogische Begründung mit Verweis auf seinen Vorschlag zu Verknüpfungen zwischen Auslösern und QS-Prozessen (vgl. Tabelle 5-13).

5.1.2.3.3 Wirkungs- und Kontext-Attribute

Chillarege nennt drei Bedingungen, die alle Attribute eines Fehlerklassifikationsschemas nach seiner Ansicht erfüllen müssen, damit aus den Erfahrungen verschiedener Projekte gelernt werden kann.[483] Folglich werden nicht nur für kausale Attribute, sondern auch für Wirkungs- und Kontext-Attributen Bedingungen formuliert. Diese Bedingungen lauten im Einzelnen:

- Bedingung 3: Die Werte jedes Attributes sind jeweils unabhängig voneinander (Orthogonalität).
- Bedingung 4: Die Attribute und ihre Werte sind unabhängig vom eingesetzten Softwareentwicklungsprozess.
- Bedingung 5: Die Attribute und ihre Werte sind unabhängig von der Art der entwickelten Software.

Für die kausalen Attribute Fehlertyp (defect type) und Auslöser (trigger) wird die dritte Bedingung bereits durch die zweite Bedingung abgedeckt.[484]

Abgesehen von einer Untersuchung, welche der Klassifikation nach dem Attribut Fehlertyp (defect type) eine gute Wiederholbarkeit bestätigt und damit auch eine Orthogonalität der Attributwerte nahe legt,[485] sind keine vergleichbaren empirischen Studien bekannt, die die Erfüllung der drei Bedingungen seitens der ODC-Attribute explizit thematisieren. Vielmehr erfolgt häufig ein Verweis auf die erfolgreichen Anwendungen von ODC für verschiedene Arten von Software und unter verschiedenen Prozessen.[486]

---

[482] Vgl. hierzu die in der Tabelle 9-1 (Anhang A.2) aufgeführten empirischen Quellen zu ODC, die das ODC-Attribut Auslöser (trigger) einsetzen.

[483] Vgl. Kapitel 5.1.2.2.

[484] Vgl. Kapitel 5.1.2.3.2.

[485] Vgl. El Emam, Wieczorek /Repeatability/ 330.

[486] Vgl. Bhandari u. a. /Improvement/ 182-184.

5.1.3 Verfahrensschritte

5.1.3.1 Überblick

ODC hat das Ziel, mittels der Analyse von Fehlern in Entwurf und Implementierung, Verbesserungspotenziale eines Softwareentwicklungsprozesses aufzuzeigen. Als Ergebnis der Anwendung von ODC stehen folglich identifizierte Prozessmängel. ODC unterstützt damit die Definition und Durchführung von Verbesserungsmaßnahmen, sie sind jedoch nicht im engeren Sinn Gegenstand von ODC.

Werden die Beschreibungen aus verschiedenen ODC-Literaturquellen integriert, besteht ein Prozessverbesserungzyklus mit ODC aus 7 Schritten (vgl. Abbildung 5-2):[487]

- Schritt 1: Ausgangspunkt für ODC ist ein gegebener Softwareentwicklungsprozess. Dieser verursacht zum einen Fehler und zum anderen liefert er erforderliche Informationen für die Konfiguration von ODC.
- Schritt 2: ODC muss für die Prozesse der Entwicklungsaufgaben und der Qualitätssicherung konfiguriert werden. Ergebnis dieser Konfiguration ist u. a. ein angepasstes ODC-Klassifikationsschema.
- Schritt 3: Fehler im Entwurf und der Implementierung werden auf der Grundlage des angepassten Klassifikationsschemas erfasst. Hierbei werden sowohl Informationen zur Entdeckung als auch zur Behebung eines Fehlers erfasst. Ergebnis dieses Schritts sind folglich nach ODC klassifizierte Fehler.
- Schritt 4: Klassifizierte Fehler werden ausgewertet, um auffällige Fehlermuster zu entdecken. Fehlermuster sind dabei Teilmengen von Fehlern, die für ein oder mehrere ODC-Attribute jeweils die gleichen Ausprägungen haben.
- Schritt 5: Für ausgewählte Fehlermuster werden die Ursachen im Softwareentwicklungsprozess (Prozessmängel) ermittelt.
- Schritt 6: Sofern möglich werden Verbesserungsmaßnahmen für die Prozesse der Entwicklungsaufgaben definiert, um vergleichbare Fehler zukünftig zu vermeiden. Sofern dies nicht möglich ist, werden Verbesserungsmaßnahmen für die Prozesse der Qualitätssicherung definiert, um vergleichbare Fehler früher zu finden.
- Schritt 7: Die definierten Verbesserungsmaßnahmen werden durchgeführt. Infolge liegt ein geänderter Softwareentwicklungsprozess vor.

---

[487] In Anlehnung an Bhandari u. a. /Case study/ 1157-1163, Chillarege u. a. /In-process measurements/ 946-954 und Butcher, Munro, Kratschmer /Software testing/ 32 f.

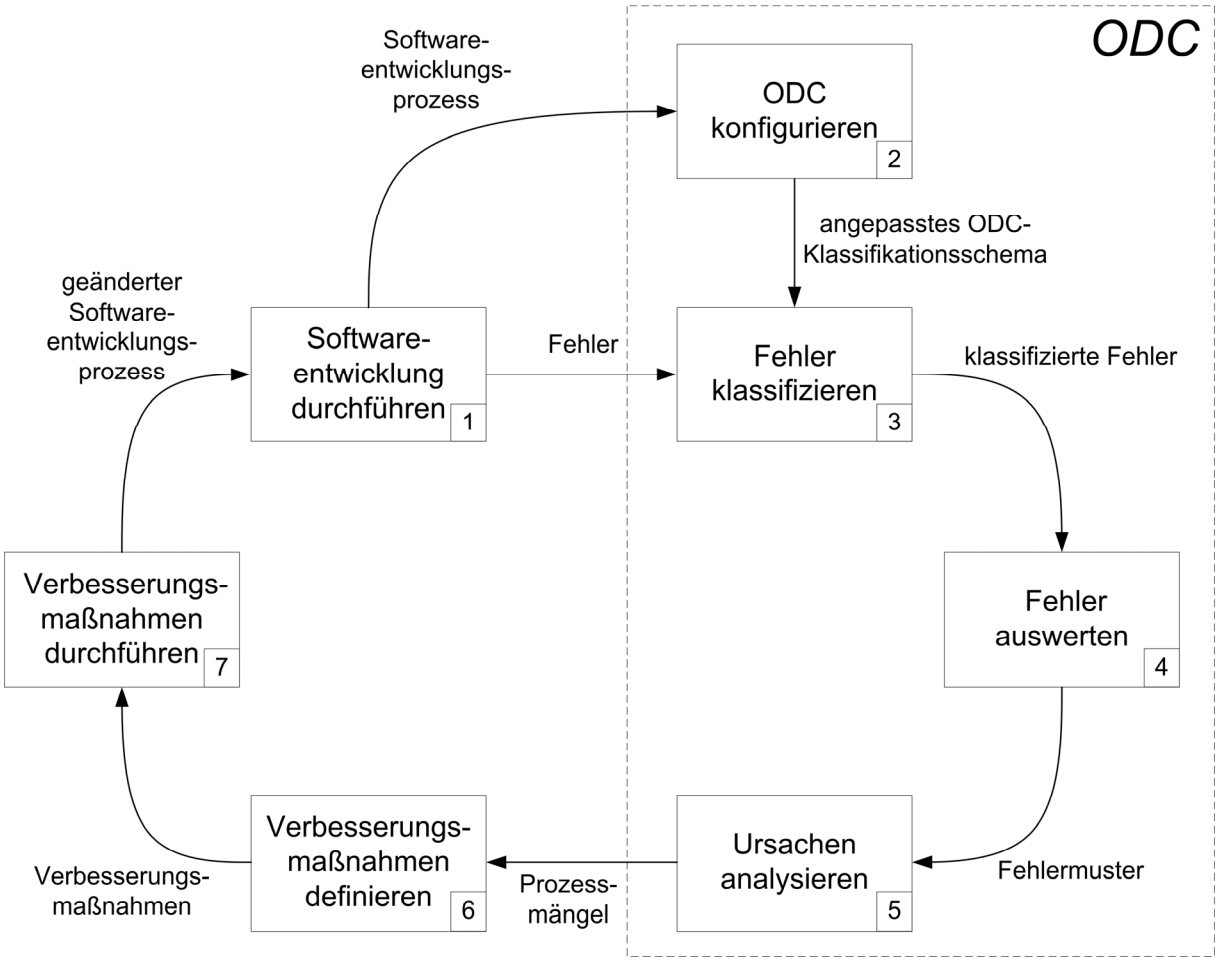


Abbildung 5-2: ODC-Verbesserungszyklus

ODC deckt als Verfahren die Schritte 2 bis 5 ab.[488] Nachfolgend werden diese im Detail erörtert. Während das ODC-Klassifikationsschema und damit das Klassifizieren von Fehlern ausführlich dokumentiert sind, werden die ODC-Konfiguration, die Fehlerauswertung und die Ursachenanalyse häufig nur rudimentär und beispielhaft beschrieben. In keiner dem Autor der vorliegenden Arbeit bekannten Quelle wird ODC als Verfahren in seiner Vollständigkeit dargestellt. Die nachfolgende Darstellung versucht diese Lücke zu schließen, in dem die Beschreibungen aus verschiedenen ODC-Literaturquellen integriert werden.

5.1.3.2 Konfiguration

ODC muss für die Klassifikation, Auswertung und Ursachenanalye von Fehlern vorbereitet werden, indem sie für die Prozesse der Entwicklungsaufgaben und der Qualitätssicherung ei-

[488] Chillarege u. a. verweisen in Chillarege u. a. /In-process measurements/ 954 darauf, dass es Bestrebungen gibt, ODC in Einklang mit dem ebenfalls bei IBM entwickelten Verfahren Defect Prevention Process, kurz DPP, zu bringen. DPP deckt die Schritte 5 bis 7 aus der Abbildung 5-2 ab und würde demnach ODC vervollständigen. Dem Autor der vorliegenden Arbeit sind keine Veröffentlichungen bekannt, die die Integration beider Verfahren behandeln. Das Verfahren DPP wird in Mays u. a. /Defect prevention/ behandelt.

nes gegebenen Projekts konfiguriert wird.[489] Diese Konfiguration konzentriert sich auf die kausalen Attribute Fehlertyp (defect type) und Auslöser (trigger). Für das ODC-Attribut Auslöser hat die Konfiguration bereits Auswirkungen auf die Klassifkation von Fehlern, wohingegen sie im Fall des Fehlertyps erst Auswirkungen für die Ursachenanalyse hat. Dies ist vermutlich der wesentliche Grund dafür, dass die Anpassung des ODC-Attributs Auslöser in der ODC-Literatur berücksichtigt wird,[490] während entsprechende Ausführungen für das ODC-Attribut Fehlertyp weitgehend ausbleiben. Bisweilen wird sogar der Eindruck erweckt, dass keine Konfiguration für das Attribut Fehlertyp erforderlich sei.[491]

Konfiguration des ODC-Attributs Fehlertyp (defect type)

Die Konfiguration des ODC-Attributs Fehlertyp wird in Bhandari u. a. /Process feedback/ diskutiert.[492] Zunächst muss ausgehend von einem existierenden Softwareentwicklungsprozess festgehalten werden, welche Prozesse für Entwicklungsaufgaben und die Qualitätssicherung existieren. Prozesse für Entwicklungsaufgaben fügen Fehler in eine Software ein, während Prozesse der Qualitätssicherung Fehler entdecken. Bhandari u. a. bezeichnen einen Prozess einer Entwicklungsaufgabe deshalb als „injection stage“ und assoziieren damit, dass der Prozess Fehler in die Software „injiziert“. Analog wird ein QS-Prozess „detection stage“ genannt. In einer Tabelle sind dann Fehlertypen mit den Prozessen für Entwicklungsaufgaben zu verknüpfen (vgl. als Beispiel die Tabelle 5-14). Die Fragestellung hierzu lautet: Welche Fehlertypen werden in welchen Prozessen für Entwicklungsaufgaben eingeführt? Hierbei ist jede Verknüpfung dadurch zu begründen, wie der jeweilige Prozess für eine Entwicklungsaufgabe in einem konkreten Projekt gelebt wird.

---

[489] Vgl. zu diesem Absatz IBM Research /ODC/ o. S und Bhandari u. a. /Process feedback/ 171-175.

[490] Vgl. z. B. Bassin, Santhanam /Maintenance/ 104 und Bassin, Biyani, Santhanam: /Sydney Olympics/ 265 f.

[491] Z. B. wird in Berling, Thelin /Case study/ 230 auf die Verknüpfungen von Fehlertypen und Prozessen von Entwicklungsaufgaben in Chillarege u. a. /In-process measurements/ 951 verwiesen. Eine Diskussion bezüglich einer projektspezifischen Anpassung bleibt aus.

[492] Vgl. Bhandari u. a. /Process feedback/ 170-173.

	Prozesse von Entwicklungsaufgaben		
Attributwert	Grobentwurf	Feinentwurf	Implementierung
Zuweisung / Initialisierung			x
Bedingung		x	x
Algorithmus / Methode		x	
Funktion / Klasse / Objekt	x		
Timing / Serialisierung		x	
interne Schnittstellen		x	
Beziehung			x

Tabelle 5-14: Beispielhafte Verknüpfungen von Fehlertypen und Prozessen von Entwicklungsaufgaben[493]

Eine analoge Tabelle ist für Fehlertypen und QS-Prozesse anzulegen. Die Verknüpfungen formulieren Erwartungen, welche Fehlertypen in welchen QS-Prozessen entdeckt werden sollen. Auch hier ist jede Verknüpfung darüber zu begründen, was in einem QS-Prozess inhaltlich gemacht wird.

	QS-Prozesse					
Attributwert	Inspektion des Grobentwurfs	Inspektion des Feinentwurfs	Code-inspektion	Komponenten-test	Funktionstest	Systemtest
Zuweisung / Initialisierung			x	x		
Bedingung			x	x		
Algorithmus / Methode			x	x	x	
Funktion / Klasse / Objekt	x				x	
Timing / Serialisierung		x				x
interne Schnittstellen		x	x			
Beziehung			x	x		

Tabelle 5-15: Beispielhafte Verknüpfungen von Fehlertypen und QS-Prozessen[494]

In der ODC-Literatur wird in der Regel in den Beispielen ein sequenzieller Entwicklungsansatz zu Grunde gelegt. Zugleich wird betont, dass die Konfiguration des ODC-Attributs Fehlertyps auch für agile Entwicklungansätze möglich ist.[495]

Die Konfiguration des ODC-Attributs Fehlertyp wird erst in der Ursachenanalyse bedeutsam. Sie hat insbesondere keine Auswirkung auf die Klassifikation von Fehlern.

---

[493] Das Beispiel ist angelehnt an Chillarege u. a. /In-process measurements/ 951.

[494] Das Beispiel ist angelehnt an Chillarege u. a. /In-process measurements/ 951

[495] Vgl. Bhandari u. a. /Process feedback/ 173. Zu agilen Entwicklungsansätzen vergleiche Trittmann /Agile Softwareprojekte/ 69-71.

Konfiguration des ODC-Attributs Auslöser (trigger)

Für die Konfiguration des ODC-Attributs Auslöser muss zunächst der projektspezifische Wertebereich des ODC-Attributs Aktivität bestimmt werden.[496] Folglich ist zu klären, welche QS-Aktivitäten geplant sind. Implizit wird unterstellt, dass die QS-Aktivitäten inhaltlich und zeitlich zusammenhängen.

Die alleinige Nennung der QS-Aktivitäten sagt nichts darüber aus, was innerhalb der QS-Aktivitäten geleistet wird.[497] Deshalb muss im nächsten Schritt für jede QS-Aktivität erhoben werden, welche Prüf- oder Testkriterien gemäß dem ODC-Attribut Auslöser zum Einsatz kommen werden. Hierbei ist zu beachten, dass Prüfkriterien nur mit Prüfaktivitäten und Testkriterien nur mit Testaktivitäten verknüpft werden. Das Ergebnis ist folglich eine Tabelle, in der projektspezifsche QS-Aktivitäten mit fest vorgegebenen Auslösern verknüpft werden (vgl. z.B. Tabelle 5-16).

QS-Aktivität	Zugeordnete Auslöser
Entwurfsreview	▪ Entwurfskonformität ▪ Kontroll- und Datenfluss ▪ Abwärtskompatibilität ▪ Umfeldkompatibilität ▪ Steuerung gemeinsam genutzter Ressourcen ▪ Entwicklerdokumentation ▪ Besonderheiten der Entwicklungssprache ▪ Nebenwirkung ▪ seltene Situation
Code-Inspektion	Die gleichen Auslöser wie für die QS-Aktivität Entwurfsreview.
Komponententest	▪ einfacher Pfad ▪ komplexer Pfad
Funktionstest	▪ Funktionsabdeckung ▪ Funktionsvariation ▪ Funktionensequenz ▪ Funktioneninteraktion
Systemtest	▪ Leistungs- oder Stresstest ▪ Fehlerbehandlung / Ausnahmebehandlung ▪ Start / Neustart ▪ Hardwarekonfiguration ▪ Softwarekonfiguration ▪ blockierter Test

Tabelle 5-16: Beispielhafte Verknüpfungen von QS-Aktivitäten mit Auslösern[498]

[496] Vgl. zu diesem Absatz IBM Research /ODC/ o. S.: "One of the first things an organization must do once they have decided to implement ODC is to define the activities they perform and map the triggers to those activities. Note that although the organization defines their activities, they do not define or redefine the triggers."

[497] Vgl. zu diesem Absatz IBM Research /ODC/ o. S.

[498] Vgl. Chillarege, Bassin /Software triggers/ 331-335.

Werden Fehler klassifiziert, ist anzugeben, in welcher projektspezifischen QS-Aktivität sie gefunden worden sind und welcher Auslöser dazu beigetragen hat. Aufgrund der Verknüpfung von QS-Aktivitäten und Auslösern wird die Auswahl eines Auslösers für einen Fehler erleichtert. Für Fehler, die in einer QS-Aktivität entdeckt werden, muss nämlich nur noch aus der Liste der mit der QS-Aktivität verknüpften Auslöser ausgewählt werden.

5.1.3.3 Fehler klassifizieren

Die Klassifikation von Fehlern nach ODC kann beginnen, sobald erste Entwurfs- oder Implementierungsfehler vorliegen.[499] ODC setzt nicht voraus, dass alle Attribute des Klassifikationsschemas eingesetzt werden müssen, um entscheidungsrelevante Erkenntnisse zu gewinnen.[500]

Zwei Zeitpunkte für die Fehlerklassifikation

ODC richtet sich an das Entwicklungsteam.[501] Dies äußert sich insbesondere darin, dass implizit erwartet wird, dass Programmierer, Softwaredesigner und Qualitätssicherer[502] Fehler klassifizieren. Für eine vollständige Klassifikation eines Fehlers sind zwei Zeitpunkte relevant:[503]

- Ein Fehler wurde entdeckt: die ODC-Attribute Aktivität (activity), Auslöser (trigger) und Auswirkung (impact) sind durch Mitarbeiter zu erfassen, die die Rolle des Qualitätssicherers wahrnehmen.
- Ein Fehler wurde behoben: die ODC-Attribute Ziel (target), Fehlertyp (defect type), Kennzeichner (qualifier), Ursprung (source) und Alter (age) sind durch den jeweiligen Programmierer oder Softwaredesigner zur erfassen.

Das Kapitel 5.1.2.1 gibt einen Überblick über das ODC-Klassifikationsschema mit seinen Attributen und Attributwerten.

Vorgehensweise bei der Fehlerklassifikation

Bei der Fehlerklassifikation können zwei Vorgehensweisen unterschieden werden:

- direkte Fehlerklassifikation: die ODC-Attribute werden erfasst, sobald ein Fehler entdeckt und behoben wird.

---

[499] Vgl. Chillarege u. a. /In-process measurements/ 955.

[500] Vgl. Chillarege, Prasad /ODC Triggers/ 669. Implizit wird aber angenommen, dass mindestens ein kausales Attribut eingesetzt wird. Vgl. hierzu Kapitel 5.1.2.3.1.

[501] Vgl. Halliday u. a. /Experiences/ 63 und Chillarege u. a. /In-process measurements/ 954.

[502] Vgl. zu diesen Rollen Kapitel 2.2.3.

[503] Vgl. zu Folgendem IBM Research /ODC/ o. S.

- indirekte Fehlerklassifikation: auf der Grundlage eines Fehlerberichts[504], in dem ein Fehler dokumentiert ist, werden nachträglich so weit wie möglich die ODC-Attribute zu dem Fehler erfasst.

ODC legt eine zeitnahe Klassifikation von Fehlern nahe und unterscheidet deshalb nicht die beiden Vorgehensweisen. Diese Unterscheidung ist aber zweckmäßig, da in empirischen Untersuchungen die Fehlerklassifikation direkt oder indirekt erfolgt.[505]

Nach Auslieferung entdeckte Fehler klassifizieren

ODC wurde ursprünglich entwickelt, um einem Entwicklungsteam in einem laufenden Softwareentwicklungsprojekt Feedback über den Prozess und die Qualität der Software zur Verfügung zu stellen.[506] Es wird aber zusätzlich auch auf nach Abschluss eines Projekts entdeckte Fehler angewandt, um für zukünftige Projekte Verbesserungen abzuleiten.[507]

Die Klassifikation von Fehlern, die nach Auslieferung der Software durch Benutzer entdeckt werden, wird etwas anders bearbeitet. Nach Chillarege ist der Auslöser für einen Fehler zu wählen, der am besten die Bedingung beschreibt, die beim Benutzer zur Fehleraktivierung führte.[508] Hierbei stehen folgende Attributwerte nicht zur Auswahl, da der Benutzer keinen Einblick in die interne Struktur einer Software hat:

- Kontroll- und Datenfluss,
- Steuerung gemeinsam genutzter Ressourcen,
- Entwicklerdokumentation,
- Besonderheiten der Entwicklungssprache,
- einfacher Pfad und
- komplexer Pfad.

Ebenso steht der Wert „blockierter Test“ nicht zur Verfügung, da er nur im Kontext einer Teststufe sinnvoll ist.

Chillarege empfiehlt im Fall eines ausgelieferten Fehlers für das ODC-Attribut Aktivität jene QS-Aktivität zu wählen, in der der Fehler am ehesten hätte entdeckt werden können.[509]

---

[504] Vgl. hierzu Kapitel 2.2.5.

[505] Vgl. hierzu die empirischen Quellen zu ODC in Anhang A.2.

[506] Vgl. Halliday u. a. /Experiences/ 61 f.

[507] Vgl. z. B. Sullivan, Chillarege /Comparison/ und Dalal u. a. /Defect patterns/.

[508] Vgl. hierzu und zu Folgendem IBM Research /ODC/ o. S.

[509] Vgl. hierzu und zu Folgendem IBM Research /ODC/ o. S.

Chillarege problematisiert nicht, inwieweit die Klassifikation des Auslösers und der Aktivität von Fehlern, die nach Auslieferung durch Benutzer entdeckt werden, einer semantischen Klassifikation[510] in seinem Sinne noch entspricht.

5.1.3.4 Klassifizierte Fehler auswerten

Gemäß ODC klassifizierte Fehler werden mit dem Ziel ausgewertet, Fehlermuster zu identifizieren.[511] Fehlermuster sind Teilmengen von Fehlern, die hinsichtlich ausgewählter Attribute die gleichen Ausprägungen haben.[512] Die Suche nach entsprechenden Fehlermustern kann manuell erfolgen oder durch die Data-Mining-Methode Attribute Focusing unterstützt werden. Diese beiden Varianten werden nachfolgend dargestellt.

Manuelle Auswertung

In den Veröffentlichungen von Chillarege und seinen Kollegen werden nur allgemeine Hinweise für eine systematische Suche nach Fehlermustern gegeben und beispielhafte Fehlermuster aus Untersuchungen dargestellt.[513] Die Beschreibung einer methodischen Vorgehensweise bleibt aus. Zu den wichtigsten Hinweisen zählen folgende Aussagen:[514]

- Fehlermuster zeigen sich insbesondere bei mehrdimensionalen Auswertungen mit zwei oder mehreren Attributen.
- Ein- oder mehrdimensionale Auswertungen, in denen das Attribut Auslöser (trigger) einbezogen wird, erlauben insbesondere Rückschlüsse über die Qualitätssicherungsmaßnahmen.[515]
- Ein- oder mehrdimensionale Auswertungen, in denen das Attribut Fehlertyp (defect type) einbezogen wird, erlauben insbesondere Rückschlüsse über die durchgeführten Entwicklungsaufgaben und die Qualität einer Software.[516]

Eine eindimensionale Auswertung hat die Verteilung einer Menge von Fehlern auf die Werte eines Attributs zum Gegenstand (absolut oder prozentual). In den empirischen Quellen zu ODC stehen bei eindimensionalen Auswertungen folgende Attribute im Vordergrund (in absteigender Häufigkeit):[517]

---

[510] Vgl. zur semantischen Klassifikation von Fehlern Kapitel 5.1.2.2.

[511] Vgl. Chillarege u. a. /In-process measurements/ 944 f.

[512] In Anlehnung an Dalal u. a. /Defect patterns/ 3 f.

[513] Vgl. hierzu die in Kapitel 5.2.2.1 aufgeführten empirischen Untersuchungen zu ODC.

[514] Vgl Chillarege u. a. /In-process measurements/ 944 f., Bhandari u. a. /Improvement/ 184 und Chillarege /Defect/ 393-396, 374-376, 384-393.

[515] Vgl. hierzu Kapitel Kausale Attribute Fehlertyp (defect type) und Auslöser (trigger)5.1.2.3.2.

[516] Vgl. hierzu Kapitel Kausale Attribute Fehlertyp (defect type) und Auslöser (trigger)5.1.2.3.2.

[517] Vgl. hierzu die Darstellung der empirischen Quellen zu ODC in Anhang A.2.

- Fehlertyp (defect type)
- Auslöser (trigger)
- Aktivität (activity)
- Kennzeichner (qualifier)
- Alter (age)
- Auswirkung (impact)

Bei einer mehrdimensionalen Auswertung wird untersucht, wie sich eine Menge von Fehler auf Tupel verteilt, die aus Werten von zwei oder mehr Attributen bestehen. Die empirischen Quellen zu ODC führen insbesondere zweidimensionale Auswertungen folgender Attribute auf (in absteigender Häufigkeit):[518]

- Fehlertyp (defect type) und Kennzeichner (qualifier)
- Fehlertyp (defect type) und Auslöser (trigger)
- Auslöser (trigger) und Zeitraum
- Aktivität (activity) und Auslöser (trigger)
- Fehlertyp (defect type) und Zeitraum

Auswertung mit Attribute Focusing

Im Zusammenhang mit ODC wurde eine Methode für die explorative Auswertung von Daten entwickelt.[519] Diese Methode trägt den Namen „Attribute Focusing“ und soll die Suche nach Fehlermustern in den Fehlerdaten automatisieren und insbesondere auch schwer erkennbare Zusammenhänge aufspüren.[520] Sie ist daher als Data-Mining-Methode[521] zu verstehen.

Wertet eine Person mit Attribut Focusing klassifizierte Fehler automatisch aus, muss sie hierfür keine spezifischen Kenntnisse in der Auswertung von Daten besitzen.[522] Vielmehr soll sie mit einem Domänenwissen über den Softwareentwicklungsprozess Erkenntnisse über Verbesserungspotenziale gewinnen, indem relevante Muster in den Daten angezeigt werden. Sie wird nachfolgend als Domänenspezialist bezeichnet.

Attribute Focus besteht im Kern aus zwei Schritten:

1. Die klassifizierten Fehler werden ausgewertet, um ihre potenzielle Relevanz („interestingness“) für einen Domänenspezialisten zu bestimmen.

---

[518] Vgl. hierzu die Darstellung der empirischen Quellen zu ODC in Anhang A.2.

[519] Vgl. hierzu und zum nächsten Satz Halliday u. a. /Experiences/ 62.

[520] Vgl. Bhandari u. a. /Improvement/ 186-190. Für eine allgemeine Darstellung dieser Methode siehe Bhandari /Attribute Focusing/.

[521] Vgl. Mertens, Wieczorrek /Strategien/ 18 f.

[522] Vgl. zu diesem Absatz Bhandari u. a. /Improvement/ 186.

2. Die Auswertungen werden nach ihrer potenziellen Relevanz sortiert.

Folgende Definitionen sind zur Bestimmung der potenziellen Relevanz eindimensionaler Auswertungen zweckmäßig:

- Für jeden Wert v_i eines Attributs q, das in der Fehlermenge auftritt, wird die Häufigkeit $N(q = v_i)$ des Auftretens bestimmt.
- $N(q)$ ist die Anzahl der Fehler, die für das Attribut q einen Wert haben.
- $choice(q)$ ist die Anzahl möglicher Werte für ein Attribut q.

Die beobachtete relative Häufigkeit des Auftretens eines Wertes v_i für ein Attribut q lässt sich wie folgt berechnen:

$$Observed(q=v_i)=\frac{N(q=v_i)}{N(q)}$$

Ausgehend von der Annahme, dass das Auftreten jedes Wertes v_i für ein Attribut q eine gleiche Wahrscheinlichkeit hat, ist die erwartete relative Häufigkeit wie folgt zu ermitteln:

$$Expected(q=v_i)=\frac{1}{choice(q)}$$

Die potenzielle Relevanz I_1 eindimensionaler Auswertungen wird bei einer gegebenen Fehlermenge wie folgt bestimmt:

$$I_1(q=v_i)=|Observed(q=v_i)-Expected(q=v_i)|$$

oder äquivalent dazu

$$I_1(q=v_i)=|\frac{N(q=v_i)}{N(q)}-\frac{1}{choice(q)}|$$

Nachfolgendes Beispiel veranschaulicht die Berechnung der potenziellen Relevanz I_1. Eine gegebene Fehlermenge besteht aus 120 Fehlern. Die Fehler verteilen sich auf die Werte des ODC-Attributs Kennzeichner (qualifier) folgendermaßen:[523]

- Etwas Fehlendes wurde eingefügt (missing): 70 Fehler
- Etwas Bestehendes war falsch und wurde korrigiert (incorrect): 40 Fehler
- Etwas Bestehendes war irrelevant und wurde entfernt (extraneous): 10 Fehler

[523] Vgl. hierzu Tabelle 5-6.

Die Tabelle 5-17 zeigt die Berechnungen für die beobachtete und erwartete relative Häufigkeit des Auftretens eines Wertes dieses Attributs, die Differenz dieser Berechnungen sowie die potenzielle Relevanz.

Attributwerte für das ODC-Attribut Kennzeichner	beobachtete relative Häufigkeit (%)	erwartete relative Häufigkeit (%)	Differenz (%)	potenzielle Relevanz (%)
Etwas Fehlendes wurde eingefügt.	58%	33%	25%	25%
Etwas Bestehendes war falsch und wurde korrigiert.	33%	33%	0%	0%
Etwas Bestehendes war irrelevant und wurde entfernt.	8%	33%	-25%	25%

Tabelle 5-17: Beispiel zur Bestimmung der potenziellen Relevanz bei eindimensionalen Auswertungen[524]

Folgende Definitionen sind zur Bestimmung der potenziellen Relevanz zweidimensionaler Auswertungen zweckmäßig:

- $N(q_1=v_i,\ q_2=u_j)$ ist die abolute Häufigkeit der Fehler in einer Fehlermenge, die für das Attributs q_1 den Wert v_i und für das Attribut q_2 den Wert u_j haben.
- $N(q_1,\ q_2)$ ist die Anzahl der Fehler, die für beide Attribut q_1 und q_2 einen Wert haben.

Die beobachtete relative Häufigkeit der Fehler, die für das Attribut q_1 den Wert v_i haben und für das Attribut q_2 den Wert u_j lässt sich wie folgt berechnen:

$$Observed(q_1=v_i,\ q_2=u_j)=\frac{N(q_1=v_i,\ q_2=u_j)}{N(q_1,\ q_2)}$$

Es wird die Annahme zu Grunde gelegt, dass die Werte zweier Attribute unabhängig voneinander sind. Unter dieser Annahme ist die erwartete relative Häufigkeit der Fehler, die für das Attribut q_1 den Wert v_i haben und für das Attribut q_2 den Wert u_j:

$$Expected(q_1=v_i,\ q_2=u_j)=Expected(q_1=v_i)\cdot Expected(q_2=u_j)$$

Die potenzielle Relevanz I_2 zweidimensionaler Auswertungen wird bei einer gegebenen Fehlermenge wie folgt bestimmt:

$$I_2(q_1=v_i,\ q_2=u_j)=|Observed(q_1=v_i,\ q_2=u_j)-Expected(q_1=v_i,\ q_2=u_j)|$$

Das Beispiel aus Tabelle 5-17 wird weiter ausgebaut, um die Berechnung der potenziellen Relevanz I_2 zu veranschaulichen. Eine gegebene Fehlermenge aus 120 Fehlern verteilt sich

[524] Die Werte in der Tabelle 5-19 wurden gerundet.

zum Beispiel wie in Tabelle 5-18 auf die Werte der ODC-Attribute Fehlertyp (defect type) und Kennzeichner (qualifier).

Fehlertyp / Kennzeichner	Zuweisung / Initialisierung	Bedingung	Algorithmus / Methode	Funktion / Klasse / Objekt	Timing / Serialisierung	interne Schnittstellen	Beziehung	Summe
Etwas Fehlendes wurde eingefügt.	0	23	0	45	2	0	0	70
Etwas Bestehendes war falsch und wurde korrigiert.	18	2	7	5	1	6	1	40
Etwas Bestehendes war irrelevant und wurde entfernt.	2	5	3	0	0	0	0	10
Summe	20	30	10	50	3	6	1	

Tabelle 5-18: Beispiel für eine Fehlerverteilung auf die Attribute Fehlertyp und Kennzeichner

Auf der Grundlage der Tabelle 5-18 wird in der Tabelle 5-19 die potenziellen Relevanz I_2 für zweidimensionale Auswertungen mit den Attributen Fehlertyp und Kennzeichner berechnet. Die Tabelle 5-19 hat folgende Spalten:

- Wert für das Attribut Fehlertyp
- Wert für das Attribut Kennzeichner
- beobachtete relative Häufigkeit für einen Wert des Attributs Fehlertyp
- beobachtete relative Häufigkeit für einen Wert des Attributs Kennzeichner
- beobachtete relative Häufigkeit für eine Wertekombination der Attribute Fehlertypen und Kennzeichner
- erwartete relative Häufigkeit für eine Wertekombination der Attribute Fehlertypen und Kennzeichner
- Differenz zwischen der beobachteten und erwarteten erwartete relative Häufigkeit für eine Wertekombination der Attribute Fehlertypen und Kennzeichner
- potenzielle Relevanz für eine Wertekombination der Attribute Fehlertypen und Kennzeichner

Werte fiwevv r Attribut Fehlertyp	Werte für Attribute Kennzeichner	beobachtete relative Häufigkeit Fehlertyp (%)	beobachtete relative Häufigkeit Kennzeichner (%)	beobachtete relative Häufigkeit Fehlertyp und Kennzeichner (%)	erwartete relative Häufigkeit Fehlertyp und Kennzeichner (%)	Differenz	potenzielle Relevanz
Algorithmus / Methode	eingefügt	8%	58%	0%	5%	-5%	5%
Algorithmus / Methode	geändert	8%	33%	6%	3%	3%	3%
Algorithmus / Methode	entfernt	8%	8%	3%	1%	2%	2%
Bedingung	eingefügt	25%	58%	19%	15%	5%	5%
Bedingung	geändert	25%	33%	2%	8%	-7%	7%
Bedingung	entfernt	25%	8%	4%	2%	2%	2%
Beziehung	eingefügt	1%	58%	0%	0%	0%	0%
Beziehung	geändert	1%	33%	1%	0%	1%	1%
Beziehung	entfernt	1%	8%	0%	0%	0%	0%
Funktion / Klasse / Objekt	eingefügt	42%	58%	38%	24%	13%	13%
Funktion / Klasse / Objekt	geändert	42%	33%	4%	14%	-10%	10%
Funktion / Klasse / Objekt	entfernt	42%	8%	0%	3%	-3%	3%
interne Schnittstellen	eingefügt	5%	58%	0%	3%	-3%	3%
interne Schnittstellen	geändert	5%	33%	5%	2%	3%	3%
interne Schnittstellen	entfernt	5%	8%	0%	0%	0%	0%
Timing / Serialisierung	eingefügt	3%	58%	2%	1%	0%	0%
Timing / Serialisierung	geändert	3%	33%	1%	1%	0%	0%
Timing / Serialisierung	entfernt	3%	8%	0%	0%	0%	0%
Zuweisung / Initialisierung	eingefügt	17%	58%	0%	10%	-10%	10%
Zuweisung / Initialisierung	geändert	17%	33%	15%	6%	9%	9%
Zuweisung / Initialisierung	entfernt	17%	8%	2%	1%	0%	0%

Tabelle 5-19: Beispiel zur Bestimmung der potenziellen Relevanz bei zweidimensionalen Auswertungen[525]

Die beschriebenen ein- und zweidimensionalen Auswertungen werden für alle Attribute bzw. Attributekombinationen durchgeführt. Bhandari schlägt vor, die Auswertungen anschließend nach ihrer Relevanz zu sortieren und mittels einer Filterfunktion, die Auswertungen mit der

[525] Die Werte in der Tabelle 5-19 wurden gerundet.

höchsten Relevanz zu filtern.[526] Hierzu werden für Auswertungen unterschiedlichen Typs verschiedene Tabellen erzeugt. Als Daumenregel empfiehlt Bhandari die Anzahl dieser Tabellen auf 20 zu beschränkwn werden, sofern die auszuwertende Person maximal 180 Minuten die Tabellen analysieren soll. Jede Tabelle soll maximal 7 Auswertungen (=Zeilen) umfassen.
Für das eingeführte Beispiel sind die nachfolgenden zwei Tabellen ein mögliches Ergebnis der Filterfunktion:

Attributwerte für das ODC-Attribut Kennzeichner	Differenz (%)	potenzielle Relevanz (%)
Etwas Fehlendes wurde eingefügt.	25%	25%
Etwas Bestehendes war irrelevant und wurde entfernt.	-25%	25%

Tabelle 5-20: Ergebnis der Filterfunktion (Teil 1)

Attributwerte für das ODC-Attribut Fehlertyp	Attributwerte für das ODC-Attribut Kennzeichner	Differenz (%)	potenzielle Relevanz (%)
Funktion / Klasse / Objekt	eingefügt	13%	13%
Funktion / Klasse / Objekt	geändert	-10%	10%
Zuweisung / Initialisierung	eingefügt	-10%	10%
Zuweisung / Initialisierung	geändert	9%	9%
Bedingung	geändert	-7%	7%

Tabelle 5-21: Ergebnis der Filterfunktion (Teil 2)

Um Fehlermuster zu erkennen, die einen Handlungsbedarf erfordern, empfiehlt Bhandari bei der Anwendung von Attribute Focusing die Ergebnisse wie folgt zu interpretieren:[527]

1. Für eindimensionale Auswertungen:
    - Für eine positive Differenz zwischen beobachteter und erwarteter relativer Häufigkeit: Warum ist die beobachtete relative Häufigkeit des Attributwertes v_i so hoch?
    - Für eine negative Differenz zwischen beobachteter und erwarteter relativer Häufigkeit: Warum ist die beobachtete relative Häufigkeit des Attributwertes v_i so niedrig?
2. Für zweidimensionale Auswertungen:
    - $(q_1=v)$ tritt häufig mit $(q_2=u)$ auf, aber selten mit $(q_2=w)$. Warum? Welches Ereignis kann hierzu geführt haben?

[526] Vgl. zu diesem Absatz Bhandari /Attribute Focusing/ 278-281 und Bhandari u. a. /Improvement/ 186-188.
[527] Vgl. zu Folgendem Bhandari /Attribute Focusing/ 282 f. und Bhandari u. a. /Improvement/ 188-190.

- $(q_1=v)$ tritt häufig mit $(q_2=u)$ auf, aber selten mit $(q_2=w)$. Ist das erstrebenswert? Was würde passieren, wenn nichts unternommen wird?

5.1.3.5 Ursachen analysieren

Für ausgewählte Fehlermuster werden die Ursachen im Softwareentwicklungsprozess (Prozessmängel) ermittelt.[528] Mit der Kenntnis der Prozessmängel eröffnet sich die Möglichkeit, Verbesserungsmaßnahmen für die Prozesse der Entwicklungsaufgaben zu definieren und einzuleiten, um ähnliche Fehler zukünftig zu vermeiden. Ebenso können die Prozesse der Qualitätssicherung verbessert werden, um vergleichbare Fehler früher zu finden.
Wie unterstützt ODC die Ursachenanalyse? Zur Beantwortung dieser Frage, ist die Arzt-Patient-Metapher hilfreich. Je genauer ein Patient die Symptome einer Krankheit beschreiben kann, desto mehr hilft er dem Arzt, auf die mögliche Ursache, eine bestimmte Krankheit, zu schließen. Analog funktioniert ODC. Sie hat nicht nur den Anspruch, die Existenz von Prozessmängeln aufzudecken.[529] Vielmehr soll durch die genaue Charakterisierung der Fehlermuster mittels verschiedenen ODC-Attributen und ihrer Werte eine Transparenz hinsichtlich der Symptome von Prozessmängeln geschaffen werden. Eine Person mit Wissen über das Softwareentwicklungsprojekt und seine Rahmenbedingungen kann diese Informationen nutzen, um auf die konkreten Prozessmängel zu schließen.[530]

Ausgehend von der Konfiguration des ODC-Attributs Fehlertyp (defect type) können Schlussfolgerungen über die Prozesse der Entwicklungsaufgaben und der Qualitätssicherung gezogen werden.[531] Die Zusammenhänge werden zunächst exemplarisch an einem Beispiel veranschaulicht. Der Fehlertyp „Funktion / Klasse / Objekt“, nachfolgend kurz als Funktionsfehler bezeichnet, sei mit dem Prozess der Entwicklungsaufgabe Grobentwurf (vgl. Tabelle 5-22) sowie mit den Qualitätssicherungsprozessen Inspektion des Grobentwurfs und Funktionstest (vgl. Tabelle 5-23) verknüpft. Das bedeutet, dass Funktionsfehler maßgeblich im Grobentwurf eingeführt werden. Zugleich wird die Erwartung formuliert, dass diese Funktionsfehler möglichst in den genannten zwei QS-Prozessen entdeckt werden. Folglich sollte der relative Anteil der Funktionsfehler an der Anzahl der in einem QS-Prozess entdeckten Fehler bei den verknüpften QS-Prozessen jeweils ein lokales Maximum erreichen.

---

[528] Vgl. zu diesem Absatz Bhandari u. a. /Case study/ 1157-1163.
[529] Vgl. hierzu und zum nächsten Satz Bhandari u. a. /Case study/ 1158 f.
[530] Vgl. z. B. Bhandari u. a. /Improvement/ 191.
[531] Vgl. zu diesem Absatz Bhandari u. a. /Process feedback/ 171-173.

	Prozesse von Entwicklungsaufgaben		
Attributwert	Grobentwurf	Feinentwurf	Implementierung
Zuweisung / Initialisierung			x
Bedingung		x	x
Algorithmus / Methode		x	
Funktion / Klasse / Objekt	x		
Timing / Serialisierung		x	
interne Schnittstellen		x	
Beziehung			x

Tabelle 5-22: Beispielhafte Verknüpfungen von Fehlertypen und Prozessen von Entwicklungsaufgaben[532]

	QS-Prozesse					
Attributwert	Inspektion des Grobentwurfs	Inspektion des Feinentwurfs	Code-inspektion	Komponenten-test	Funktionstest	Systemtest
Zuweisung / Initialisierung			x	x		
Bedingung			x	x		
Algorithmus / Methode			x	x	x	
Funktion / Klasse / Objekt	x				x	
Timing / Serialisierung		x				x
interne Schnittstellen		x	x			
Beziehung			x	x		

Tabelle 5-23: Beispielhafte Verknüpfungen von Fehlertypen und QS-Prozessen[533]

Für jeden Fehlertyp kann ein binärer Baum definiert werden.[534] Die Knoten stellen jeweils QS-Prozesse dar. Von jedem Knoten gehen zwei Kanten aus. Die linke Kante steht jeweils für den Fall, dass der Anteil des gefundenen Fehlertyps im jeweiligen QS-Prozess hoch ist, und die rechte Kante analog für den Fall, dass der Anteil niedrig ist. Sofern keine historischen Daten vorliegen, um zu bestimmen, wann der Anteil eines gefundenen Fehlertyps hoch oder niedrig ist, kann dies, wie im Folgenden erläutert, durch Vergleich der relativen Anteile ermittelt werden.

Sei `q(t)` der Anteil der Fehler vom Typ `t`, die in einem QS-Prozess `q` entdeckt werden. Dann ist der Anteil hoch, wenn eines der folgenden Aussagen wahr ist:[535]

---

[532] Das Beispiel ist angelehnt an Chillarege u. a. /In-process measurements/ 951.

[533] Das Beispiel ist angelehnt an Chillarege u. a. /In-process measurements/ 951.

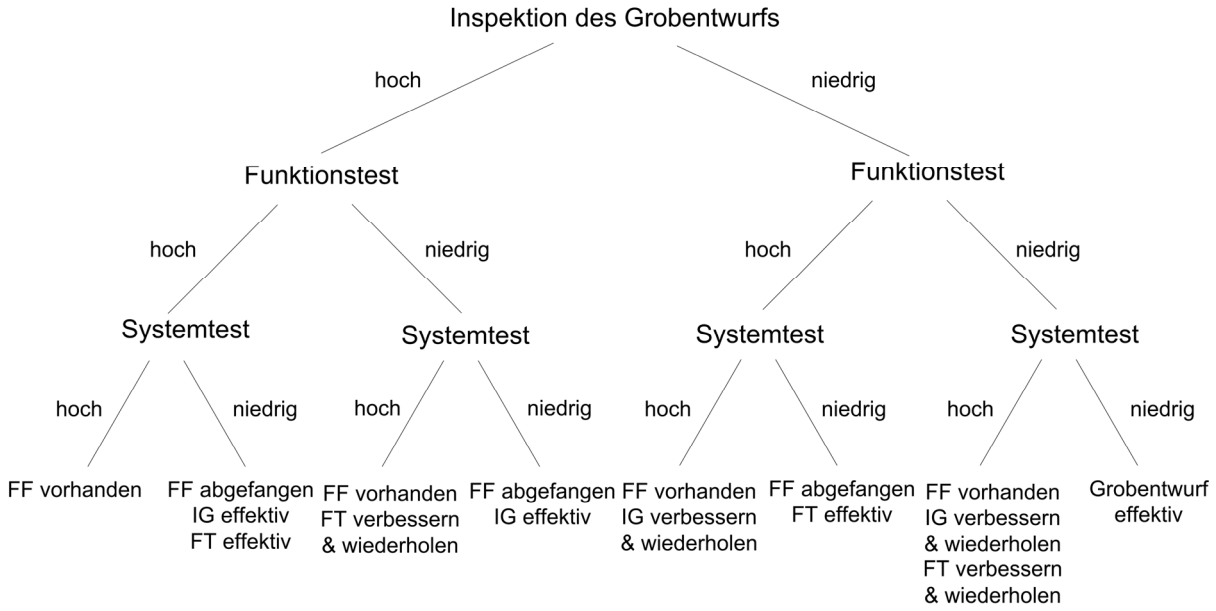
[534] Vgl. zu diesem Absatz Bhandari u. a. /Process feedback/ 171 f.

[535] Vgl. zu Folgendem Bhandari u. a. /Process feedback/ 175.

1. Für alle Fehlertypen u, die nicht mit q verknüpft sind, gilt: q(t) > q(u) und t ist mit q verknüpft.
2. Für alle Fehlertypen u, die mit q verknüpft sind, gilt: q(t) ≥ q(u) und t ist nicht mit q verknüpft.

Sind beide Aussagen falsch ist der Anteil niedrig.

Die Abbildung 5-3 veranschaulicht einen entsprechenden binären Baum für Funktionsfehler. Aus Vereinfachungsgründen werden nur die zwei mit Funktionsfehlern verknüpften QS-Prozesse gemäß Tabelle 5-23 sowie der zeitlich anschließende QS-Prozess (im vorliegenden Fall der Systemtest) abgebildet.



Abkürzungen: FF = Funktionsfehler, IG = Inspektion des Grobentwurfs, FT = Funktionstest

Abbildung 5-3: Binärer Baum für Funktionsfehler (Beispiel)[536]

In den Blättern des binären Baums stehen Schlussfolgerungen, die sich aufgrund der Konfiguration des ODC-Attributs Fehlertyp und der jeweiligen Anteile eines Fehlertyps in den QS-Prozessen sachlogisch ergeben. Die Schlussfolgerungen stehen unter der Annahme, dass die im letzten QS-Prozess entdeckten Fehler repräsentativ sind für die in einer Software verbliebenen Fehler. Folgende grundlegende Schlussfolgerungen sind möglich:

1. Funktionsfehler sind noch vorhanden.
2. Die Inspektion des Grobentwurfs war effektiv.
3. Der Funktionstest war effektiv.

[536] In Anlehnung an Bhandari u. a. /Process feedback/ 172.

4. Die Inspektion des Grobentwurfs sollte verbessert und wiederholt werden.
5. Der Funktionstest sollte verbessert und wiederholt werden.

Anhand allgemeiner Regeln können solche Schlussfolgerungen wie in Abbildung 5-3 abgeleitet werden. Sei $D_t(d_1)$ eine Funktion, welche die Anzahl der in einem QS-Prozess d_1 entdeckten Fehler vom Typ t normalisiert (z. B. Anzahl Codeanweisungen).[537] Sofern durch Vergleich der relativen Anteile der Fehler eines bestimmten Typs t ermittelt wird, ob der Anteil in einem QS-Prozess q hoch oder niedrig ist, sind folgende Definitionen zweckmäßig:[538]

Sei $q(t)$ der Anteil der Fehler vom Typ t, die in einem QS-Prozess q entdeckt werden. Es gilt $D_t(q) := \text{hoch}$, wenn eine der folgenden zwei Aussagen wahr ist:

1. Für alle Fehlertypen u, die nicht mit q verknüpft sind, gilt: $q(t) > q(u)$ und t ist mit q verknüpft.
2. Für alle Fehlertypen u, die mit q verknüpft sind, gilt: $q(t) \geq q(u)$ und t ist nicht mit q verknüpft.

Sind beide Aussagen falsch, gilt $D_t(q) := \text{niedrig}$. Zusätzlich wird $\text{hoch} > \text{niedrig}$ angenommen.

Nachfolgende allgemeine Schlussfolgerungen können bei einer gegebenen Konfiguration des ODC-Attributs Fehlertyp abgeleitet werden:[539]

- Der QS-Prozess d_1 sollte verbessert und wiederholt werden.
  1. d_1 und d_2 sind QS-Prozesse.
  2. d_1 ist mit dem Fehlertyp t verknüpft.
  3. d_2 ist nicht mit dem Fehlertyp t verknüpft.
  4. d_2 folgt zeitlich nach d_1.
  5. $D_t(d_1) \leq D_t(d_2)$.

- Der Prozess der Entwicklungsaufgabe i hat einen Prozessmangel und verursacht Fehler vom Typ t.
  1. d_1, d_2 sind QS-Prozesse, i der Prozess einer Entwicklungsaufgabe.
  2. i und d_1 sind mit dem Fehlertyp t verknüpft.

---

[537] Vgl. Bhandari u. a. /Process feedback/ 173. Die Anzahl der in einem QS-Prozess entdeckten Fehler können z. B. mittels der geprüften oder getesten Anzahl Codezeilen normalisiert werden.

[538] Vgl. zu Folgendem Bhandari u. a. /Process feedback/ 175 f.

[539] Vgl. zu Folgendem Bhandari u. a. /Process feedback/ 173 f.

3. d_2 ist nicht mit dem Fehlertyp t verknüpft.
4. d_2 folgt zeitlich nach d_1, d_1 folgt zeitlich nach i.
5. $D_t(d_1) \leq D_t(d_2)$.
6. $D_t(k) \leq D_t(d_2)$, für alle QS-Prozesse k, die zeitlich nach d_2 folgen.

- Fehler vom Typ t wurden abgefangen, die durch den Prozess der Entwicklungaufgabe i verursacht worden sind.
  1. d_1, d_2, d_3 sind QS-Prozesse, i der Prozess einer Entwicklungsaufgabe.
  2. i, d_1, d_3 sind mit dem Fehlertyp t verknüpft.
  3. d_2 ist nicht mit dem Fehlertyp t verknüpft.
  4. d_3 folgt zeitlich nach d_2, d_2 folgt zeitlich nach d_1, d_1 folgt zeitlich nach i.
  5. $D_t(d_1) \leq D_t(d_2) \leq D_t(d_3)$.
  6. $D_t(k) \leq D_t(d_3)$, für alle QS-Prozesse k, die zeitlich nach d_3 folgen.

- Der QS-Prozess d_3 war effektiv, Fehler vom Typ t zu entdecken.
  1. d_1, d_2, d_3 sind QS-Prozesse.
  2. d_1 und d_3 sind mit dem Fehlertyp t verknüpft.
  3. d_2 ist nicht mit dem Fehlertyp t verknüpft.
  4. d_3 folgt zeitlich nach d_2, d_2 folgt zeitlich nach d_1.
  5. $D_t(d_1) \leq D_t(d_2) \leq D_t(d_3)$.
  6. $D_t(k) \leq D_t(d_3)$, für alle QS-Prozesse k, die zeitlich nach d_3 folgen und nicht mit dem Fehlertyp t verknüpft sind.

5.2 Empirische Befunde zur Orthogonal Defect Classification

Wie praxistauglich ist die Orthogonal Defect Classification? Um einen Beitrag zur Beantwortung dieser Frage zu leisten, wird nachfolgend eine systematische Literaturanalyse durchgeführt und empirische Befunde zu ODC zusammengetragen.

In Kapitel 5.2.1 wird dargelegt, wie die systematische Literaturanalyse als Forschungsmethode eingesetzt wird. Das Ergebnis der Literaturrecherche sind empirische Literaturquellen, die bestimmte Kriterien erfüllen. Die Literaturquellen werden in Kapitel 5.2.2 dargestellt und bewertet. Aus den Literaturquellen werden in Kapitel 5.2.3 relevante empirische Befunde zur ODC beschrieben.

5.2.1 Vorgehensweise: systematische Literaturanalyse

Zielfragen

Die systematische Literaturanalyse wurde bereits in Kapitel 4.3.1 als Forschungsmethode eingeführt. Um die Praxistauglichkeit von ODC einzuschätzen, soll die systematische Literaturanalyse zu ODC folgende Zielfragen beantworten:

1. Welche Anwendungsgebiete von ODC wurden empirisch untersucht? Welcher Nutzen wurde dabei festgestellt?
2. Welche Voraussetzungen sind gemäß den empirischen Untersuchungen erforderlich, um ODC einzuführen und anzuwenden? Welcher Aufwand wurde für die Einführung und Anwendung von ODC festgestellt?
3. Welche Probleme und Herausforderungen sind bei der Einführung und Anwendung von ODC empirisch festgestellt worden?

Auswahl der Literaturquellen

Es werden ausschließlich Literaturquellen in der Literaturanalyse berücksichtigt, die die folgenden zwei Kriterien erfüllen:

1. Die Literaturquelle ist empirisch, d. h. bestimmte Aussagen beruhen auf Beobachtungen der Realität.
2. Die empirischen Aussagen beziehen sich auf die Anwendung von ODC bei der Entwicklung oder Wartung von Anwendungssoftware. Sofern sich Aussagen auf Systemsoftware oder systemnahe Software beziehen,[540] werden sie nur dann berücksichtigt, sofern sie auch auf Anwendungssoftware übertragbar sind.

Auswahl der wissenschaftlichen Zeitschriften und Konferenzbände

In die Literaturanalyse wurden 16 wissenschaftliche Zeitschriften und die Konferenzbände von 8 Konferenzen aus dem Zeitraum Januar 1990 bis einschließlich Dezember 2005 einbezogen. Es handelt um die gleichen Zeitschriften und Konferenzen, die bereits in der Literaturanalyse in Kapitel 4.3.2.2 berücksichtigt worden sind. Die Tabelle 5-24 und Tabelle 5-25 wiederholen, um welche Zeitschriften und Konferenzen es sich handelt. Weitere Informationen zu beiden Tabellen enhält das Kapitel 4.3.2.2.

[540] Vgl. hierzu Kapitel 2.2.1.

Nr.	Titel der Zeitschrift	Abkürzung	Rang	Untersuchte Ausgaben			Anmerkungen
Z1	Communications of the AIS	CAIS	B	Nr. 1, Jg. 1, 1999	bis	Nr. 41, Jg. 15, 2005	1
Z2	Communications of the ACM	CACM	A	Nr. 1, Jg. 33, 1990	bis	Nr. 12, Jg. 48, 2005	-
Z3	Empirical Software Engineering	ESE	-	Nr. 1, Jg. 1, 1996	bis	Nr. 4, Jg. 10, 2005	1
Z4	European Journal of Information Systems	EJIS	B	Nr. 1, Jg. 6, 1997	bis	Nr. 4, Jg. 14, 2005	2
Z5	IBM Systems Journal	IBMSJ	C	Nr. 1, Jg. 29, 1990	bis	Nr. 4, Jg. 44, 2005	-
Z6	IEEE Software	IEEESw	C	Nr. 1, Jg. 7, 1990	bis	Nr. 6, Jg. 22, 2005	-
Z7	IEEE Transactions on Software Engineering	IEEETSE	B	Nr. 1, Jg. 16, 1990	bis	Nr. 12, Jg. 31, 2005	-
Z8	Information Systems Journal	ISJ	C	Nr. 1, Jg. 6, 1990	bis	Nr. 4, Jg. 15, 2005	-
Z9	Information Systems Research	ISR	A	Nr. 1, Jg. 1, 1990	bis	Nr. 4, Jg. 16, 2005	-
Z10	Journal of Management Information Systems	JMIS	A	Nr. 4, Jg. 6, 1990	bis	Nr. 2, Jg. 22, 2005	-
Z11	Journal of the Association for Information Systems	JAIS	C	Nr.1, Jg. 1, 2000	bis	Nr. 9, Jg. 6, 2005	1
Z12	Management Science	MS	A	Nr. 1, Jg. 36, 1990	bis	Nr. 12, Jg. 51, 2005	-
Z13	MIS Quarterly	MISQ	A	Nr. 1, Jg. 14, 1990	bis	Nr. 4, Jg. 29, 2005	-
Z14	Requirements Engineering	RE	-	Nr. 1, Jg. 1, 1996	bis	Nr. 4, Jg. 10, 2005	1
Z15	Software Quality Journal	SQJ	-	Nr. 1, Jg. 6, 1997	bis	Nr. 4, Jg. 13, 2005	2
Z16	Wirtschafts-informatik	WIRT	C	Nr. 1, Jg. 37, 1995	bis	Nr. 6, Jg. 47, 2005	2

Tabelle 5-24: Auswahl der wissenschaftlichen Zeitschriften[541]

[541] Zu den Anmerkungen siehe das Kapitel 4.3.2.2.

Nr.	Titel der Konferenz	Abkürzung	Untersuchte Konferenzbände	herausgebende Körperschaft	Anmerkungen
K1	International Symposium on Software Metrics	METRICS	1993 bis 2005	IEEE	1
K2	International Symposium on Software Reliability Engineering	ISSRE	1991 bis 2005	IEEE	2
K3	International Symposium on Empirical Software Engineering	ISESE	2002 bis 2005	IEEE	1
K4	IEEE International Conference on Software Maintenance	ICSM	1990 bis 2005	IEEE	-
K5	International Conference on Software Engineering	ICSE	1990 bis 2005	IEEE, ACM	-
K6	IEEE International Symposium on Requirements Engineering	ISRE	1993 bis 2001	IEEE	1, 3
K7	IEEE International Conference on Requirements Engineering	ICRE	1994 bis 2005	IEEE	1, 3
K8	International Conference on Information Systems	ICIS	1990 bis 2005	Association for Information Systems	-

Tabelle 5-25: Auswahl der Konferenzbände[542]

Ablauf der Literaturrecherche

Der Ablauf der Suche nach relevanten Literaturquellen orientiert sich an den Empfehlungen von Webster und Watson:[543]

1. *Ausgangssuche (AS)*: Im ersten Schritt wurde im definierten Literaturspektrum mit den 16 Zeitschriften und 8 Konferenzen nach Literaturquellen gesucht, die die Auswahlkriterien erfüllen. Die Quellen, die den Kriterien genügen, sind die Ausgangstreffer.
2. *Rückwärtssuche (RS)*: Die Literaturverzeichnisse der Ausgangstreffer wurden nach weiteren Literaturquellen durchsucht, die die Auswahlkriterien erfüllen.
3. *Vorwärtssuche (VS)*: Sowohl über Web of Science[544] als auch Citeseer[545] wurde ermittelt, welche Literaturquellen die Ausgangstreffer zitieren. Jede einzelne Literaturquelle wurde dahingehen überprüft, ob sie die Auswahlkriterien erfüllt.

In allen drei Suchschritten wurden die Zusammenfassungen und, falls angegeben, die Schlüsselwörter jeder Literaturquelle untersucht. Zusätzlich wurden in der Ausgangssuche in drei

[542] Zu den Anmerkungen siehe das Kapitel 4.3.2.2.

[543] Vgl. Webster, Watson /Literature review/ xvi.

[544] Web of Science ist unter der URL http://scientific.thomson.com/products/wos/ zu finden (Stand 26.07.2006). Es ist ein kostenpflichtiges Produkt der Thomson Scientific.

[545] Die CiteSeer Scientific Literature Digital Library ist unter der URL http://citeseer.ist.psu.edu/ zu finden (Stand 26.07.2006). Die Eigendarstellung lautet: „CiteSeer is a scientific literature digital library and search engine that focuses primarily on the literature in computer and information science.“ Citeseer wurde ursprünglich durch das NEC Research Institute entwickelt. Die Nutzung ist kostenfrei.

Literaturdatenbanken[546] die ausgewählten Zeitschriften und Konferenzen nach dem Suchbegriff „Orthogonal Defect Classification“ durchsucht, um auch solche empirische Literaturquellen zu identifzieren, die ODC nicht in ihrer Zusammenfassung oder den Schlüsselwörtern erwähnen. Sofern anhand dieser Informationen nicht geklärt werden konnte, ob eines der Auswahlkriterien verletzt wird, wurden ausgewählte Abschnitte des Beitrags gelesen.
In der Rückwärts- und Vorwärtssuche können auch Literaturquellen gefunden werden, die sich in Zeitschriften oder Konferenzbänden befinden, die nicht in der Ausgangssuche berücksichtigt worden sind. Ebenso müssen entsprechende Quellen nicht im Zeitraum von 1990 bis 2005 liegen. Entscheidend sind für Treffer in der Rückwärts- und Vorwärtssuche ausschließlich die genannten Auswahlkriterien.
Sofern zu einem Beitrag in einem Konferenzband ein nahezu inhaltlich identischer Beitrag in einer wissenschaftlichen Zeitschriften erscheint, wird nur der Beitrag in der Zeitschrift berücksichtigt.
Die Literaturrecherche wurde im April 2006 durchgeführt.

Auswertung der Literaturquellen

Um im Rahmen der systematischen Literaturanalyse eine Synthese relevanter Aussagen der betrachteten Literatur zu erreichen, wurde in Anlehnung an die Zielfragen der Literaturanalyse jeweils eine Mindmap erstellt:

- Eine Mindmap zu den Anwendungsgebieten und Nutzen von ODC.
- Eine Mindmap zu den erforderlichen Voraussetzungen und dem Aufwand für die Einführung und Anwendung von ODC.
- Eine Mindmap zu den Problemen und Herausforderungen bei der Einführung und Anwendung von ODC.

Im ersten Schritt wurden hierzu die identifizierten Literaturquellen nach Abschluss der Literaturrecherche mit dem Ziel gelesen, für die Mindmaps relevante Aussagen zu extrahieren. Immer wenn eine neue relevante Aussage gefunden worden ist, wurde die jeweilige Mindmap erweitert.[547] Bei jeder Erweiterung wurden die Aussagen im Mindmap inhaltlich zu Gruppen zusammengefasst. Die Mindmaps wurden im nächsten Schritt überarbeitet.

---

[546] Es handelt sich um folgende Literaturdatenbanken: EBSCOhost, ACM Digital Library, IEEE Xplore digital library.

[547] Vgl. zu dieser Vorgehensweise Webster, Watson /Literature review/ xvi f.

5.2.2 Ergebnisse der Literaturrecherche

5.2.2.1 Überblick über die identifizierten empirischen Quellen

In der Ausgangssuche wurden 20 Quellen identifiziert, die die Auswahlkriterien erfüllen.[548] Die Rückwärtssuche erweitert diese Menge um weitere 5 Literaturquellen,[549] die Vorwärtssuche um 2 Literaturquellen. Folglich liegen insgesamt 27 Treffer vor (vgl. Tabelle 5-26).

[548] Der Beitrag Lutz, Mikulski /Requirements/ (ODC25) in der IEEE Software deckt inhaltlich die beiden Konferenzbeiträge Lutz, Mikulski /Testing/ und Lutz, Mikulski /Requirements Discovery/ ab. Ebenso umfasst der Beitrag Bhandari u. a. /Case study/ (ODC07) aus den IEEE Transactions on Software Engineering den ODC-relevanten Teil des Konferenzbeitrags Poulin, Brown /Quality improvement/. Die Konferenzbeiträge werden deshalb nicht berücksichtigt. Vgl. zu dieser Vorgehensweise Kapitel 5.2.1.

[549] In der Rückwärtssuche wurden 6 Quellen gefunden, die die Auswahlkriterien erfüllen. Ein Beitrag von Bridge und Miller mit dem Titel „Orthogonal Defect Classification: Using Defect Data to Improve Software Development“ konnte nicht beschafft werden, da die bibliographischen Angaben in Bassin, Kratschmer, Santhanam /Software development/ 74 unzureichend sind.

Nr.	Kurzzitat	gefunden während	Veröffent-lichungsjahr	gefunden in
ODC01	Bassin, Biyani, Santhanam /Metrics/	AS	2002	IBMSJ
ODC02	Bassin, Biyani, Santhanam: /Sydney Olympics/	AS	2001	ISSRE
ODC03	Bassin, Kratschmer, Santhanam /Software development/	AS	1998	IEEESw
ODC04	Bassin, Santhanam /Maintenance/	AS	2001	ICSM
ODC05	Bassin, Santhanam /Software triggers/	AS	1997	ISSRE
ODC06	Berling, Thelin /Case study/	AS	2003	METRICS
ODC07	Bhandari u. a. /Case study/	AS	1993	IEEETSE
ODC08	Bhandari u. a. /Improvement/	AS	1994	IBMSJ
ODC09	Buczilowski /Defect reduction/	AS	1995	ISSRE
ODC10	Butcher, Munro, Kratschmer /Software testing/	AS	2002	IBMSJ
ODC11	Chaar u. a. /In-process evaluation/	AS	1993	IEEETSE
ODC12	Chernak /Statistical approach/	AS	1996	IEEETSE
ODC13	Chillarege u. a. /In-process measurements/	AS	1992	IEEETSE
ODC14	Chillarege, Bassin /Software triggers/	RS	1995	DCCA
ODC15	Chillarege, Biyani /Risk/	AS	1994	ISSRE
ODC16	Chillarege, Kao, Condit /Defect type/	RS	1991	ICSE
ODC17	Chillarege, Prasad /ODC Triggers/	RS	2002	DSN
ODC18	Christmansson, Santhanam /Error injection/	AS	1996	ISSRE
ODC19	Dalal u. a. /Defect patterns/	RS	1999	ASM
ODC20	Damm, Lundberg /Test process improvement/	AS	2005	ISESE
ODC21	El Emam, Wieczorek /Repeatability/	AS	1998	ISSRE
ODC22	Henningsson, Wohlin /Fault classification agreement/	AS	2004	ISESE
ODC23	Lutz, Mikulski /Anomalies/	AS	2004	IEEETSE
ODC24	Lutz, Mikulski /Requirements/	AS	2004	IEEESw
ODC25	Sullivan, Chillarege /Comparison/	RS	1992	FTCS
ODC26	Halliday u. a. /Experiences/	VS	1994	JSS
ODC27	Zheng u. a. /Static analysis/	VS	2006	IEEETSE

Tabelle 5-26: In der Literaturrecherche gefundene ODC-Quellen[550]

Die Tabelle 5-26 führt neben einer identifizierenden Nummer

- ein Kurzzitat des Beitrags,
- die Angabe, in welchen Schritt der Literaturrecherche der Beitrag gefunden wurde (AS = Ausgangssuche, RS = Rückwärtssuche, VS = Vorwärtssuche),
- das Jahr der Veröffentlichung und

[550] Im Rahmen der Rückwärts- und Vorwärtssuche wurden eine Literaturquelle zusätzlich in der Zeitschrift Journal of Systems and Software (JSS) und 4 Literaturquellen in Konferenzbänden folgender Konferenzen gefunden: IFIP Working Conference on Dependable Computing for Critical Applications (DCCA), International Conference on Applications of Software Measurement (ASM), International Conference on Dependable Systems and Networks (DSN) und International Symposium on Fault Tolerant Computing (FTCS).

- die Abkürzung der Zeitschrift oder des Konferenzbandes, in der oder dem der Beitrag veröffentlicht worden ist.[551]

Die Tabelle 5-27 zeigt auf, in welchem Jahr die 27 Literaturquellen jeweils veröffentlicht worden sind. Demnach wurden 16 der gefundenen Literaturquellen (59%) in den neunziger Jahren und 11 der Literaturquellen (41%) im Zeitraum von 2000 bis 2006 veröffentlicht. ODC ist demnach keine Modeerscheinung, wenn berücksichtigt wird, dass die erste Veröffentlichung zu ODC aus dem Jahr 1991 stammt.

Veröffentlichungsjahr	Absolute Häufigkeit Literaturquellen	Relative Häufigkeit Literaturquellen
1990	0	0,00%
1991	1	3,70%
1992	2	7,41%
1993	2	7,41%
1994	3	11,11%
1995	2	7,41%
1996	2	7,41%
1997	1	3,70%
1998	2	7,41%
1999	1	3,70%
2000	0	0,00%
2001	2	7,41%
2002	3	11,11%
2003	1	3,70%
2004	3	11,11%
2005	1	3,70%
2006	1	3,70%
	Summe: 27	Summe: 100%

Tabelle 5-27 : Veröffentlichungsjahre der ODC-Literaturquellen

Unter den 27 Treffern sind 12 Beiträge aus wissenschaftlichen Zeitschriften (vgl. Tabelle 5-28) und 15 aus Konferenzen (vgl. Tabelle 5-29). Es fällt auf, dass viele Beiträge (44%) in der Zeitschrift IEEE Transactions on Software Engineering und Konferenzbänden der International Symposium on Software Reliability Engineering erschienen sind.

[551] Vgl. Tabelle 5-24 und Tabelle 5-25.

Zeitschrift	Absolute Häufigkeit Literaturquellen	Relative Häufigkeit Literaturquellen
IEEETSE	6	50,00%
IBMSJ	3	25,00%
IEEESW	2	16,67%
JSS	1	8,33%
RE	0	0,00%
ESE	0	0,00%
CACM	0	0,00%
Winfo	0	0,00%
ISJ	0	0,00%
SQJ	0	0,00%
	Summe: 12	Summe: 100%

Tabelle 5-28: Verteilung der ODC-Literaturquellen auf Zeitschriften

Konferenz	Absolute Häufigkeit Literaturquellen	Relative Häufigkeit Literaturquellen
ISSRE	6	40,00%
ISESE	2	13,33%
Metrics	1	6,67%
ASM	1	6,67%
ICSE	1	6,67%
FTCS	1	6,67%
DSN	1	6,67%
DCCA	1	6,67%
ICSM	1	6,67%
ICRE	0	0,00%
ISRE	0	0,00%
	Summe: 15	Summe: 100%

Tabelle 5-29: Verteilung der ODC-Literaturquellen auf Konferenzen

Insgesamt 48 Autoren tragen zu den 27 Literaturquellen bei. Eine kleine Gruppe von Autoren ist besonders aktiv. Die Tabelle 5-30 listet die Autoren auf, die an mehr als einem der 27 ODC-Beiträge mitgewirkt haben.

Nachname und Vorname des Autors	Organisation	Anzahl Quellen, an der der Autor mitgewirkt hat	Prozentualer Anteil an der Gesamtzahl der gefundenen Quellen
Chillarege, Ram[552]	IBM	10	37,04%
Bassin, Kathryn	IBM	6	22,22%
Bhandari, Inderpal	IBM	5	18,52%
Chaar, Jarir	IBM	5	18,52%
Halliday, Michael J.	IBM	5	18,52%
Santhanam, Padmanabhan	IBM	5	18,52%
Biyani, Shriram Ram	IBM	3	11,11%
Kratschmar, Theresa	IBM	2	7,41%
Lutz, Robyn R.	Jet Propulsion Laboratory	2	7,41%
Mikulski, Inés Carmen	Jet Propulsion Laboratory	2	7,41%
Tarver, Eric D.	IBM	2	7,41%
Weitere 37 Autoren	verschiedene	jeweils 1	jeweils 3,70%

Tabelle 5-30: Autoren der ODC-Literaturquellen[553]

Ram Chillarege, der Erfinder der ODC, ist bei ca. 37% der Beiträge als Autor oder Co-Autor aufgeführt. Ihm folgt Bassin mit 6 Beiträgen (ca. 22%). Bhandari, Chaar, Halliday und Santhanam arbeiten an jeweils 5 Beiträgen (jeweils ca. 19%) mit. Diese 5 Autoren wirken an 16 von 27 Literaturquellen mit (ca. 59%).

Neben der Autorenkonzentration fällt zu dem auf, dass 50% der 48 Autoren zum Zeitpunkt der Veröffentlichung der IBM angehören (vgl. Abbildung 5-4).

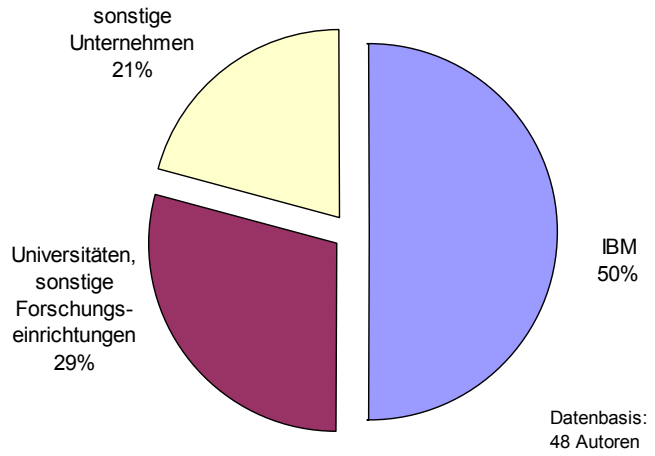


Abbildung 5-4: Organisationszugehörigkeit der Autoren der ODC-Literaturquellen

[552] Ram Chillarege hat während seiner Zeit als Mitarbeiter bei IBM ODC entwickelt und weiterentwickelt. In einer Veröffentlichung aus dem Jahre 2002 (vgl. Chillarege, Prasad /ODC Triggers/) wird deutlich, dass er nicht mehr bei IBM angestellt ist. Da seine meisten ODC-Veröffentlichungen aber während seiner IBM-Zeit stammen, wird er in der Tabelle 5-30 mit einer Organisationszugehörigkeit zu IBM aufgeführt.

[553] Da Literaturquellen auch mehrere Autoren haben können, entspricht die Summe der Anzahl der Quellen, an denen jeweils ein Autor mitgewirkt hat, nicht der Gesamtanzahl der Quellen.

ODC wurde ursprünglich bei IBM entwickelt. Insgesamt 17 der 27 Literaturquellen (ca. 63%), also fast zwei Drittel der Quellen, führen einen ersten Autor auf, der zum Zeitpunkt der Veröffentlichung bei IBM tätig ist. Es ist jedoch eine Entwicklung festzustellen, dass zunehmend Forscher außerhalb der IBM ODC aufgreifen und empirische Untersuchungen durchführen (vgl. Abbildung 5-5). Während im Zeitraum zwischen 1990 bis 1999 12 von 16 Quellen einen Autor mit IBM-Zugehörigkeit nennen, sind es im Zeitraum zwischen 2000 bis 2006 6 von 11 Quellen. Der prozentuale Anteil fällt also von 75% auf ca. 55%. Folglich ist genau eine gegenläufige Beobachtung bei Quellen zu beobachten, die eine Person als ersten Autor nennen, die einem anderen Unternehmen als IBM oder einer Universität oder sonstigen Forschungseinrichtung angehören. Der prozentuale Anteil steigt von 25% (im Zeitraum 1990 bis 1999) auf 45% (Zeitraum 2000 bis 2006).

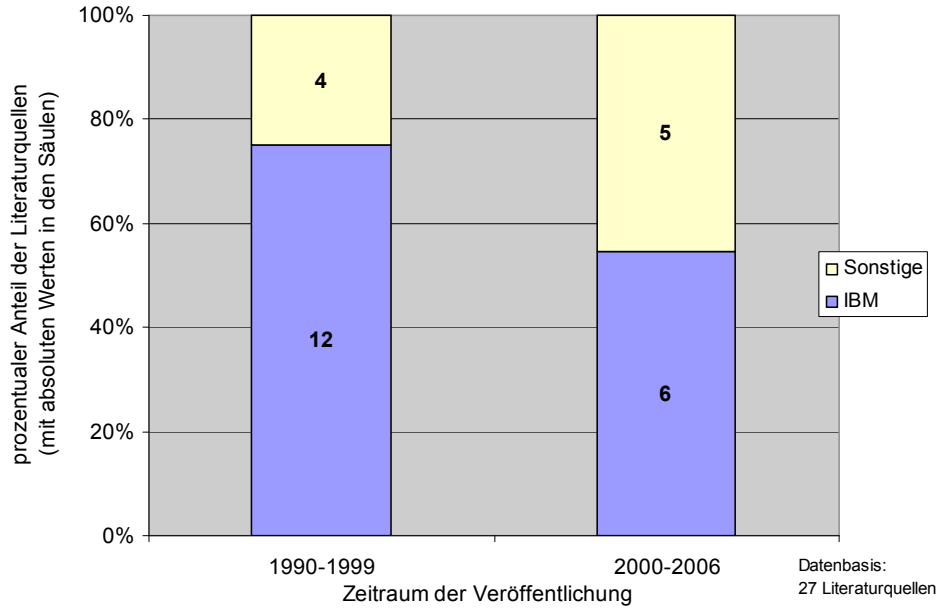


Abbildung 5-5: Organisationszugehörigkeit des ersten Autors der ODC-Literaturquellen

Im Rahmen der Literaturanalyse muss bei der Interpretation der Ergebnisse empirischer Untersuchungen das Risiko berücksichtigt werden, dass tendenziell eher Beiträge veröffentlicht werden, die von erfolgreichen Projekten oder erfolgreichen Anwendungen von Methoden, Verfahren, Werkzeuge etc. berichten.[554] Dieses Risiko ist tendenziell größer, wenn wie im vorliegenden Fall

- eine kleine Autorengruppe zu vielen Literaturquellen beiträgt,

[554] Dieses Verhalten entspricht dem Widerwillen in kritischen Softwareentwicklungsprojekten schlechte Nachrichten zu berichten. Vgl. hierzu auch die Literaturanalyse Smith, Keil /Reluctance/.

- sehr viele Autoren einer Organisation angehören, in der ein Verfahren entwickelt worden ist, das Gegenstand der Untersuchungen ist und
- viele Beiträge in wenigen Konferenzbänden oder Zeitschriften erscheinen.

5.2.2.2 Darstellung der empirischen Quellen

In der empirischen Literaturanalyse werden 27 empirische Quellen zu ODC berücksichtigt. Um dem Leser die Möglichkeit zu geben, sich einen schnellen Überblick über diese einzelnen Quellen zu verschaffen, werden sie im Anhang A.2 anhand einheitlicher Beschreibungskriterien zusammengefasst. Zu diesen Beschreibungskriterien zählen zum einen die bereits in Kapitel 4.3.3.2 eingeführten Kriterien:

- Ziele der Untersuchung
- Vorgehensweise der Untersuchung
- Subjekte bzw. Objekte der Untersuchung
- wesentliche Ergebnisse der Untersuchung

Zum anderen werden zusätzlich folgende für den ODC-Kontext spezifischen Beschreibungskriterien eingesetzt:

- *Softwareart*:
  Auf welche Art von Software wurde ODC angewandt? Folgende Ausprägungen werden unterschieden: Anwendungssoftware, systemnahe Software oder Systemsoftware.[555]
- *Eingesetzte ODC-Attribute für die Fehlerfindung*:
  Welche ODC-Attribute wurden eingesetzt, nachdem ein Fehler gefunden worden ist? Folgende ODC-Attribute kommen in Frage: Aktivität (activity), Auslöser (trigger) und Auswirkung (impact).
- *Eingesetzte ODC-Attribute für die Fehlerkorrektur*:
  Welche ODC-Attribute wurden eingesetzt, nachdem ein Fehler korrigiert worden ist? Folgende ODC-Attribute kommen in Frage: Ziel (target), Fehlertyp (defect type), Kennzeichner (qualifier), Quelle (source), Alter (age).
- *Zusätzliche Attribute für Fehler*:
  Werden Fehler durch weitere Attribute beschrieben? Wenn ja, durch welche?
- *Fehlerauswertung mit Attribute Focusing*:
  Wird Attribute Focusing eingesetzt, um klassifizierte Fehler auszuwerten?

[555] Vgl. hierzu auch Kapitel 2.2.1.

- *Veränderung des ODC-Verfahrens*:
  Wurde das bestehende ODC-Verfahren verändert? Wenn ja, wie?
- *Erweiterung des ODC-Verfahrens*:
  Wurde das bestehende ODC-Verfahren erweitert? Wenn ja, wie?
- *Quantitative ODC-Daten*:
  Welche quantitativen ODC-Auswertungen werden aufgeführt?
- *Anzahl Fehler*:
  Wie viele Fehler wurden nach ODC klassifiziert und ausgewertet?
- *Entwicklungsfehler*:
  Sind unter den nach ODC klassifizierten und ausgewerteten Fehlern auch Entwicklungsfehler enthalten?
- *Produkionsfehler*:
  Sind unter den nach ODC klassifizierten und ausgewerteten Fehlern auch Produktionsfehler enthalten?
- *Vorgehensweise bei Fehlerklassifikation*:[556]
  Wurden die Fehler direkt oder indirekt klassifiziert?

5.2.2.3 Bewertung der empirischen Quellen

Die in der Literaturanalyse berücksichtigten empirischen Quellen zu ODC werden nachfolgend bewertet. Es besteht hierbei nicht der Anspruch, die Literaturquellen inhaltlich und forschungsmethodisch eingehend zu bewerten. Vielmehr werden die Literaturquellen dahingehend bewertet, inwieweit sie überhaupt grundsätzliche Voraussetzungen erfüllen, um inhaltlich und forschungsmethodisch bewertet werden zu können. Für eine entsprechende Bewertung wurden bereits in Kapitel 4.3.3.3.1 Bewertungskriterien eingeführt, welche ebenfalls auf die ODC-Quellen angewandt werden. Die Tabelle 5-31 und Tabelle 5-32 geben das Ergebnis wieder. In beiden Tabellen steht „j“ für „ja“ und „n“ für „nein“.

[556] Vgl. hierzu Kapitel 5.1.3.3.

	Gegenstand der Untersuchung				Kontext der Unter-suchung			Durchführung der Untersuchung					
ODC-Nr. der empirischen Quelle	explizite Formulierung der Untersuchungsziele	Typ der Untersuchung nach Eigendarstellung	Typ der Untersuchung nach Typologie dieser Arbeit	Zweck der Untersuchung	Rahmenbedingungen	Hypothesen	Vergleich mit verwandten Forschungsergebnissen	Auswahl der Untersuchungs-subjekte bzw. –objekte	Definition der Untersuchungs-einheit	Datenerhebung	Art der erhobenen Daten	Datenauswertung	Einschränkungen der Untersuchung
01	n	keine Angaben	Fallstudie	deskriptiv	j	-	n	n	j	j	quantitativ, qualitativ	n	n
02	n	keine Angaben	Fallstudie	deskriptiv	j	-	n	n	j	j	quantitativ, qualitativ	n	n
03	n	Fallstudie	Ex-post-Analyse	deskriptiv	n	-	n	n	j	n	quantitativ, qualitativ	n	n
04	n	keine Angaben	Ex-post-Analyse	deskriptiv	n	-	n	n	j	n	quantitativ, qualitativ	n	n
05	j	keine Angaben	Ex-post-Analyse	deskriptiv	n	-	n	n	j	n	quantitativ, qualitativ	n	n
06	j	Fallstudie	Fallstudie	explorativ	j	-	j	n	j	j	quantitativ	j	j
07	j	Fallstudie	Ex-post-Analyse	deskriptiv	j	-	j	n	j	j	quantitativ, qualitativ	j	n
08	j	keine Angaben	Ex-post-Analyse	deskriptiv	j	-	j	n	j	n	quantitativ, qualitativ	j	n
09	j	keine Angaben	Ex-post-Analyse	deskriptiv	n	-	n	n	n	n	quantitativ, qualitativ	n	n
10	j	Fallstudie	Ex-post-Analyse	deskriptiv	j	-	n	n	j	n	quantitativ, qualitativ	n	n
11	j	keine Angaben	Ex-post-Analyse	deskriptiv	j	-	n	n	j	j	quantitativ, qualitativ	j	n
12	j	Fallstudie	Fallstudie	deskriptiv	j	-	n	n	j	j	quantitativ, qualitativ	j	n
13	j	keine Angaben	Ex-post-Analyse	deskriptiv	n	-	n	n	j	n	quantitativ	n	n
14	j	keine Angaben	Fallstudie	explorativ	j	-	n	n	j	j	quantitativ	n	n
15	j	keine Angaben	Ex-post-Analyse	deskriptiv	n	-	n	n	j	n	quantitativ, qualitativ	n	n

Tabelle 5-31: Bewertung der empirischen ODC-Literaturquellen

	Gegenstand der Untersuchung				Kontext der Unter-suchung			Durchführung der Untersuchung					
ODC-Nr. der empirischen Quelle	explizite Formulierung der Untersuchungsziele	Typ der Untersuchung nach Eigendarstellung	Typ der Untersuchung nach Typologie dieser Arbeit	Zweck der Untersuchung	Rahmenbedingungen	Hypothesen	Vergleich mit verwandten Forschungsergebnissen	Auswahl der Untersuchungs-subjekte bzw. –objekte	Definition der Untersuchungs-einheit	Datenerhebung	Art der erhobenen Daten	Datenauswertung	Einschränkungen der Untersuchung
16	j	keine Angaben	Fallstudie	explorativ	j	-	j	n	j	j	quantitativ	j	j
17	j	Fallstudie	Fallstudie	deskriptiv	j	-	n	n	j	j	quantitativ, qualitativ	n	n
18	j	keine Angaben	Fallstudie	deskriptiv	n	-	n	n	j	j	quantitativ	j	n
19	j	Fallstudie	Ex-post-Analyse	deskriptiv	n	-	n	n	n	n	qualitativ	n	j
20	j	Fallstudie	Fallstudie	deskriptiv	j	-	j	n	j	j	quantitativ, qualitativ	n	j
21	j	keine Angaben	Feld-experiment	explana-torisch	j	j	j	n	j	j	quantitativ	j	j
22	j	keine Angaben	Labor-experiment	explana-torisch, explorativ	j	j	j	j	j	j	quantitativ, qualitativ	j	n
23	n	keine Angaben	Fallstudie	deskriptiv	j	-	j	j	j	j	quantitativ	j	j
24	n	keine Angaben	Fallstudie	deskriptiv	j	-	n	j	j	j	quantitativ	n	n
25	n	keine Angaben	Fallstudie	deskriptiv	j	-	n	j	j	j	quantitativ	n	n
26	j	keine Angaben	Ex-post-Analyse	deskriptiv	n	-	n	n	n	n	qualitativ	n	n
27	j	keine Angaben	Fallstudie	explorativ	j	-	j	j	j	j	quantitativ	j	j

Tabelle 5-32: Bewertung der empirischen ODC-Literaturquellen (Fortsetzung)

Die Literaturquellen werden in der Tabelle 5-31 und Tabelle 5-32 jeweils anhand einer eindeutigen Nummer identifiziert, welche in der Tabelle 5-26 bereits aufgeführt wurde. Um ein schnelles Auffinden der Quelle zu ermöglichen, werden die eindeutigen Nummern und die dazugehörigen Kurzzitate der empirischen Literaturquellen in der Tabelle 5-33 wiederholt.

Nr.	Kurzzitat
ODC01	Bassin, Biyani, Santhanam /Metrics/
ODC02	Bassin, Biyani, Santhanam: /Sydney Olympics/
ODC03	Bassin, Kratschmer, Santhanam /Software development/
ODC04	Bassin, Santhanam /Maintenance/
ODC05	Bassin, Santhanam /Software triggers/
ODC06	Berling, Thelin /Case study/
ODC07	Bhandari u. a. /Case study/
ODC08	Bhandari u. a. /Improvement/
ODC09	Buczilowski /Defect reduction/
ODC10	Butcher, Munro, Kratschmer /Software testing/
ODC11	Chaar u. a. /In-process evaluation/
ODC12	Chernak /Statistical approach/
ODC13	Chillarege u. a. /In-process measurements/
ODC14	Chillarege, Bassin /Software triggers/
ODC15	Chillarege, Biyani /Risk/

Nr.	Kurzzitat
ODC16	Chillarege, Kao, Condit /Defect type/
ODC17	Chillarege, Prasad /ODC Triggers/
ODC18	Christmansson, Santhanam /Error injection/
ODC19	Dalal u. a. /Defect patterns/
ODC20	Damm, Lundberg /Test process improvement/
ODC21	El Emam, Wieczorek /Repeatability/
ODC22	Henningsson, Wohlin /Fault classification agreement/
ODC23	Lutz, Mikulski /Anomalies/
ODC24	Lutz, Mikulski /Requirements/
ODC25	Sullivan, Chillarege /Comparison/
ODC26	Halliday u. a. /Experiences/
ODC27	Zheng u. a. /Static analysis/

Tabelle 5-33: Eindeutige Nummern und Kurzzitate der Literaturquellen

Ergebnis der Bewertung der empirischen ODC-Literaturquellen

Die Bewertung der 27 empirischen Literaturquellen führt zu folgenden Ergebnissen:

- Der überwiegende Anteil der Literaturquellen verfolgt einen deskriptiven Zweck (75%).[557] Folglich versuchen diese Untersuchungen durch die präzise Beschreibung einer bekannten Untersuchungseinheit neue Erkenntnisse zu gewinnen. 18% der Quellen haben eine explorative Ausrichtung. Nur ca. 7% der Untersuchungen versuchen ausgewählte Ausschnitte der Realität zu erklären (explanatorischer Zweck). Eine ähnliche Feststellung machen auch Dubé und Paré bei ihrer Literaturanalyse zu positivistischen Fallstudien in der Wirtschaftsinformatik.[558]
- Als Untersuchungstyp werden die Fallstudie (ca. 48%) oder die Ex-post-Analyse (ca. 44%) bevorzugt. Interessant in diesem Zusammenhang ist, dass Autoren von Ex-Post-Analysen ihre Untersuchung entweder als Fallstudie einordnen (4 von 12) oder keine Angaben zum Untersuchungstyp machen (8 von 12). Zelkowitz und Wallace stellen in

[557] Die Quelle Henningsson, Wohlin /Fault classification agreement/ (ODC22) beschreibt zwei Untersuchungen, die einen unterschiedlichen Zweck verfolgen. Bei der Auswertung des Zwecks der Untersuchungen wird diese Quelle deshalb zweimal berücksichtigt.

[558] Vgl. Dubé, Paré /Case research/ 605.

ihrer Literaturanalyse fest, dass Autoren empirischer Untersuchungen häufig den Typ ihrer Untersuchung unzureichend einordnen.[559]

- 56% der Untersuchungen erheben sowohl qualitative als auch quantitative Daten. An zweiter Stelle kommen Untersuchungen mit ausschließlich quantitativen Daten mit ca. 37%. Folglich werden in ca. 93% der Untersuchungen quantitative Daten erhoben und ausgewertet. Dieser hohe prozentuale Anteil ist nicht überraschend, da im Kontext von ODC schließlich mit Fehlerkennzahlen gearbeitet wird.

5.2.3 Ergebnisse der Auswertung der empirischen Quellen

Ausgangspunkt der systematischen Literaturanalyse zu empirischen Befunden zu ODC sind drei Zielfragen:

1. Welche Anwendungsgebiete von ODC wurden empirisch untersucht? Welcher Nutzen wurde dabei festgestellt?
2. Welche Voraussetzungen sind gemäß den empirischen Untersuchungen erforderlich, um ODC einzuführen und anzuwenden? Welcher Aufwand wurde für die Einführung und Anwendung von ODC festgestellt?
3. Welche Probleme und Herausforderungen sind bei der Einführung und Anwendung von ODC empirisch festgestellt worden?

In Kapitel 5.2.1 wurde bereits erläutert, wie die Quellen im Rahmen der Literaturanalyse ausgewertet werden, um die aufgeführten Fragen zu beantworten. Die Darstellung der Ergebnisse der Literaturanalyse wird nachfolgend nach diesen drei Zielfragen strukturiert.

5.2.3.1 Anwendungsgebiete und Nutzen von ODC

Anwendungsgebiete von ODC

Die Auswertung der empirischen Quellen zeigt unterschiedliche Anwendungsgebiete von ODC auf:

- Aufzeigen von Verbesserungspotenzialen in Entwicklungsprozessen (Anforderungsanalyse, Softwareentwurf und Implementierung)[560] für ein laufendes Softwareentwicklungsprojekt.
- Aufzeigen von Verbesserungspotenzialen in Entwicklungsprozessen (Anforderungsanalyse, Softwareentwurf und Implementierung) für zukünftige Softwareentwicklungsprojekte.

[559] Vgl. Zelkowitz, Wallace /Models/ 30.
[560] Vgl. Kapitel 2.2.3.

- Aufzeigen von Verbesserungspotenzialen in Prozessen der Qualitätssicherung[561] für ein laufendes Softwareentwicklungsprojekt.
- Aufzeigen von Verbesserungspotenzialen in Prozessen der Qualitätssicherung für zukünftige Softwareentwicklungsprojekte.
- Planung und Steuerung der Qualitätssicherungsprozesse in laufenden Softwareentwicklungsprojekten.
- Steuerung der Beziehungen zwischen Auftraggeber und -nehmer bei einer ausgelagerten Softwareentwicklung.
- Bestimmung der Qualität einer Software und Risikobewertung.
- Bewertung der Effektivität von Verbesserungsmaßnahmen.
- Bestimmung der Effektivität von Qualitätssicherungsprozessen.

Vier empirische Quellen zu ODC führen kein Anwendungsgebiet von ODC für den praktischen Einsatz an, sondern untersuchen bestimmte Aspekte von ODC oder nutzen ODC als Instrument, um eine Forschungsfrage zu beantworten. Die Zielsetzungen dieser Untersuchungen sind im Einzelnen:

- ODC14: Welche Bedingungen führen bei einer ausgelieferten Software dazu, dass latente Fehler aktiv werden und damit Fehlverhalten der Software auslösen? Wie verteilen sich diese Bedingungen über die Zeit?[562]
- ODC16: Existieren messbare Ursache-Wirkungszusammenhänge zwischen Typen von Fehlern und dem Zuverlässigkeitswachstum (reliability growth) einer Software?[563]
- ODC21 und ODC22: Ist die Klassifikation von Fehlern nach dem ODC-Attribut Fehlertyp (defect type) wiederholbar, d. h. unabhängig von einzelnen Personen?[564]

Die Tabelle 5-34 gibt einen Überblick darüber, wie häufig die Anwendungsgebiete von ODC in den ausgewerteten 27 empirischen Quellen thematisiert werden.

[561] Vgl. Kapitel 2.4.

[562] Vgl. Chillarege, Bassin /Software triggers/ 327 f.

[563] Vgl. Chillarege, Kao, Condit /Defect type/ 246, 254.

[564] Vgl. El Emam, Wieczorek /Repeatability/322 und Henningsson, Wohlin /Fault classification agreement/ 95.

ODC-Anwendungsgebiet	Anzahl Quellen mit dem jeweiligen Anwendungsgebiet (von insgesamt 27 Quellen)	prozentualer Anteil der Quellen mit dem jeweiligen Anwendungsgebiet (von insgesamt 27 Quellen)
Aufzeigen von Verbesserungspotenzialen in Entwicklungsprozessen für zukünftige Projekte	11	40,74%
Aufzeigen von Verbesserungspotenzialen in Qualitätssicherungsprozessen für zukünftige Projekte	9	33,33%
Bestimmung der Effektivität von Qualitätssicherungsprozessen	8	29,63%
Aufzeigen von Verbesserungspotenzialen in Qualitätssicherungsprozessen für laufende Projekte	8	29,63%
Planung und Steuerung der Qualitätssicherungsprozesse in laufenden Projekten	6	22,22%
Bestimmung der Qualität einer Software und Risikobewertung	6	22,22%
Bewertung der Effektivität von Verbesserungsmaßnahmen	4	14,81%
Aufzeigen von Verbesserungspotenzialen in Entwicklungsprozessen für laufende Projekte	4	14,81%
Steuerung der Beziehungen bei einer ausgelagerten Softwareentwicklung	2	7,41%
Empirische Untersuchung ohne Anwendungsgebiet	4	14,81%

Tabelle 5-34: Verteilung ODC-Anwendungsgebiete auf die Quellen

Die Tabelle 5-35 und Tabelle 5-36 listen auf, welche Anwendungsgebiete von ODC Gegenstand welcher empirischen Quelle sind. In den Zeilenköpfen stehen jeweils die einzelnen Quellen mit ihren eindeutigen Nummern gemäß Tabelle 5-33. Die Anwendungsgebiete werden in den Spaltenköpfen aufgeführt. In einer Zelle der Tabelle werden die relevanten Seitenzahlen der jeweiligen Quellen genannt, unter denen ein Anwendungsgebiet von ODC beschrieben wird.

ODC-Nr. der empirischen Quelle	Verbesserungspotenziale in Entwicklungsprozessen für zukünftige Projekte	Verbesserungspotenziale in Qualitätssicherungsprozessen für zukünftige Projekte	Effektivität von Qualitätssicherungsprozessen	Verbesserungspotenziale in Qualitätssicherungsprozessen für laufende Projekte	Planung und Steuerung der Qualitätssicherungsprozesse in laufenden Projekten	Qualität einer Software und Risikobewertung	Effektivität von Verbesserungsmaßnahmen	Verbesserungspotenziale in Entwicklungsprozessen für laufende Projekte	Steuerung der Beziehungen bei einer ausgelagerten Softwareentwicklung	Empirische Untersuchung ohne Anwendungsgebiet
01			S. 24 f.		S. 17	S. 25				
02										
03		S. 68 f., 70, 73 f.	S. 72	S. 68	S. 68	S. 67 f.		S. 68	S. 68	
04	S. 732	S. 732					S. 732		S. 732	
05	S. 107 f.	S. 106 f.					S. 107 f.			
06	S. 234			S. 236						
07				S. 1164			S. 1164 f.	S. 1164		
08	S. 191 f., 202 f. 203-205, 207, 209			S. 191 f., 200 f., 202 f.				S. 191, 194-198		
09	S. 275									
10	S. 35	S. 35, 42	S. 36-38	S. 40	S. 36	S. 36, 40				
11			S. 1061-1063, 1065-1067,	S. 1061-1063	S. 1060 f., 1063-1065, 1066, 1067-1069					
12		S. 868-875								
13			S. 952-954.	S. 948, 952	S. 948, 952 f.	S. 948				
14										S. 327-342
15					S. 288	S. 287 f.				
16										S. 251-254
17	S. 677	S. 677	S. 677				S. 677			
18		S. 182								
19	S. 8	S. 8								
20		S. 158 f.	S. 159							

Tabelle 5-35: Anwendungsgebiete von ODC in den empirischen Quellen

ODC-Nr. der empirischen Quelle	Verbesserungspotenziale in Entwicklungsprozessen für zukünftige Projekte	Verbesserungspotenziale in Qualitätssicherungsprozessen für zukünftige Projekte	Effektivität von Qualitätssicherungsprozessen	Verbesserungspotenziale in Qualitätssicherungsprozessen für laufende Projekte	Planung und Steuerung der Qualitätssicherungsprozesse in laufenden Projekten	Qualität einer Software und Risikobewertung	Effektivität von Verbesserungsmaßnahmen	Verbesserungspotenziale in Entwicklungsprozessen für laufende Projekte	Steuerung der Beziehungen bei einer ausgelagerten Softwareentwicklung	Empirische Untersuchung
21										S. 322-330
22										S. 95-104
23	S. 179									
24	S.24									
25	S. 483 f.					S. 475, 478-482				
26				S. 61 f.				S. 61 f.		
27			S. 246-249							

Tabelle 5-36: Anwendungsgebiete von ODC in den empirischen Quellen (Fortsetzung)

Nutzen von ODC

ODC konnte gemäß den empirischen Untersuchungen für die oben aufgeführten Anwendungsgebiete erfolgreich durchgeführt werden. Für jede Anwendung werden nachfolgend Beispiele aus den Untersuchungen aufgeführt:

- Aufzeigen von Verbesserungspotenzialen in Entwicklungsprozessen für ein laufendes Softwareentwicklungsprojekt
  - Die Anwendung von ODC zeigt auf, dass zwecks Fehlerkorrektur erforderliche Codeänderungen häufiger durchzuführen sind, um die Stabilität bei der Ausführung des Codes zu erhöhen.[565]
  - Um den mit ODC identifizierten Mangel an Kommunikation zwischen verschiedenen Entwicklergruppen entgegenzuwirken, werden regelmäßige Abstimmungssitzungen eingeführt.[566]
  - Mit Hilfe von ODC wird transparent, dass der Entwurf einer Softwarekomponente nicht vollständig ist und vervollständigt werden muss.[567]

[565] Vgl. Bassin, Kratschmer, Santhanam /Software development/ 68 (ODC03).

[566] Vgl. Bhandari u. a. /Improvement/ 191 (ODC08).

[567] Vgl. Bhandari u. a. /Case study/ 1164 (ODC07).

- Aufzeigen von Verbesserungspotenzialen in Entwicklungsprozessen für zukünftige Softwareentwicklungsprojekte
    - Bei der Entwicklung einer Anwendungssoftware für Großrechner wird der Code einer Komponente von einer bestehenden Software für eine andere Plattform portiert.[568] Die Anwendung von ODC führt zu einer Überarbeitung der Kriterien für Portierungsentscheidungen.
    - Nach der Anwendung von ODC wird der Prozess für die Abnahme und Bewertung von Anforderungen überarbeitet.[569] Prozesse des Grob- und Feinentwurfs werden im Hinblick auf produktübergreifende Protokolle und dokumentierte Schnittstellen verbessert.
    - Die Anwendung von ODC zeigt Verbesserungsbedarf im Feinentwurfsprozess auf.[570]
    - Mit Hilfe von ODC wird der Entwicklungsprozess dahingehend verbessert, dass der Aufwand für den Entwurf von 40% auf 50% des Projektgesamtaufwands erhöht wird, die frühe Einbindung von internen und externen Kunden einer Software sichergestellt wird und Prototypen für ein frühes Feedback genutzt werden.[571]
    - Änderungen in Softwarekomponenten werden zukünftig vollständig dokumentiert.[572]
    - Die Anforderungsanalyse wird auch in die Wartung einer Software eingebunden.[573]
- Aufzeigen von Verbesserungspotenzialen in Prozessen der Qualitätssicherung für ein laufendes Softwareentwicklungsprojekt
    - Die Anwendung von ODC macht transparent, dass insbesondere Tests für die Interaktion zwischen Softwarekomponenten, Last- und Stresstests sowie Tests für die Wiederherstellbarkeit der Software erforderlich sind.[574]

[568] Vgl. zu diesem Absatz Bassin, Santhanam /Maintenance/ 732 (ODC04).
[569] Vgl. zu diesem Absatz Bassin, Santhanam /Software triggers/ 107 (ODC05).
[570] Vgl. Berling, Thelin /Case study/ 234 (ODC06).
[571] Vgl. Buczilowski /Defect reduction/ 175 (ODC09).
[572] Vgl. Butcher, Munro, Kratschmer /Software testing/ 35 (ODC10).
[573] Vgl. Lutz, Mikulski /Anomalies/ 179 (ODC23).
[574] Vgl. Bassin, Kratschmer, Santhanam /Software development/ 68 (ODC03).

- Aufgrund fehlender Softwareanforderungen sind Ergebnisse des Feinentwurfs unvollständig.[575] Mit ODC wird der Bedarf für eine zusätzliche Inspektion des Anforderungsdokuments auf Vollständigkeit erkannt.
- Eine Softwarekomponente muss einer zusätzlichen Inspektion unterzogen werden, um die Abwärtskompatibilität der Software mit Vorgängerversionen zu gewährleisten.[576]
- Um die Testeffektivität zu steigern, werden zusätzliche Testfälle eingeführt. Diese erfüllen bestimmte Testkriterien, die zuvor nur unzureichend abgedeckt worden sind.[577]

- Aufzeigen von Verbesserungspotenzialen in Prozessen der Qualitätssicherung für zukünftige Softwareentwicklungsprojekte
  - ODC verdeutlicht, dass umfangreichere Komponententests vor der Codeintegration erforderlich sind. Zusätzlich wird eine separate Teststufe für Performanztests der Software zukünftig vorgesehen.[578]
  - Die Teststrategie wird geändert, um portierten Softwarecode angemessener testen zu können.[579] Zusätzlich wird ein neuer Mitarbeiter eingestellt, um Codeinspektionen im Hinblick auf einfache Probleme durchzuführen und damit erfahrene Mitarbeiter zu entlasten.
  - Tester werden geschult, um im Funktionstest mehr Testkriterien abdecken zu können.[580]
  - Für zukünftige Codeinspektionen werden auf der Grundlage der ODC-Ergebnisse Checklisten entwickelt.[581]
- Planung und Steuerung der Qualitätssicherungsprozesse in laufenden Softwareentwicklungsprojekten.
  - Eine angepasste ODC-Anwendung erlaubt die Vollständigkeit der Testbemühungen hinsichtlich mehrerer Dimensionen (z. B. Testtiefe und Vielfalt der

[575] Vgl. zu diesem Absatz Bhandari u. a. /Case study/ 1164 (ODC07).
[576] Vgl. Bhandari u. a. /Improvement/ 202 f. (ODC08).
[577] Vgl. Butcher, Munro, Kratschmer /Software testing/ 35 (ODC10).
[578] Vgl. Bassin, Kratschmer, Santhanam /Software development/ 68 (ODC03).
[579] Vgl. zu diesem Absatz Bassin, Santhanam /Maintenance/ 732 (ODC04).
[580] Vgl. Butcher, Munro, Kratschmer /Software testing/ 35 (ODC10).
[581] Vgl. Chernak /Statistical approach/ 875 (ODC12).

Testkriterien) besser zu bewerten und Erwartungen für die Ausführung der Testfälle zu setzen.[582]

- Mit ODC können in laufenden Projekten erforderliche Verbesserungsmaßnahmen identifiziert werden, was die Terminplanung unterstützt.[583]
- ODC erlaubt es, quantitative Eingangs- und Ausgangskriterien für Teststufen zu definieren.[584]
- Mit ODC kann der Fortschritt der Qualität einer Software bewertet werden.[585]
- ODC unterstützt die angemessene Besetzung eines Inspektionsteams, da durch Prüf- und Testkriterien Anforderungen an die Erfahrungen und Fähigkeiten eines Inspektionsteilnehmers formuliert werden.[586] Analoges gilt für die Besetzung eines Testteams, das Testfälle erstellt.[587]

- Steuerung der Beziehungen zwischen Auftraggeber und -nehmer bei einer ausgelagerten Softwareentwicklung
  - Mit ODC kann der Auftragnehmer quantitative Freigabekriterien für die Auslieferung einer Software definieren.[588]
  - ODC unterstützt Vertragsverhandlungen für die Auslagerung einer Softwareentwicklung, indem es die Definition quantitativer Erwartungen erlaubt. [589]
  - Ein ausgelagertes Entwicklungsteam verbessert mit ODC seinen Vertragsverhandlungsprozess, um sicherzustellen, dass Entwurfsdokumente vom Auftraggeber in angemessener Granularität zur Verfügung gestellt werden. [590]
- Bestimmung der Qualität einer Software und Risikobewertung
  - Mit Hilfe von ODC wird die Zuverlässigkeit mehrerer Softwareprodukte bewertet und verglichen.[591]

---

[582] Vgl. Bassin, Biyani, Santhanam /Metrics/ 17 (ODC01).

[583] Vgl. Bassin, Kratschmer, Santhanam /Software development/ 68 (ODC03) und Chillarege, Biyani /Risk/ 288 (ODC15).

[584] Vgl. Butcher, Munro, Kratschmer /Software testing/ 36 (ODC10) und Bassin, Kratschmer, Santhanam /Software development/ 68-70 (ODC03).

[585] Vgl. Bassin, Kratschmer, Santhanam /Software development/ 68-70 (ODC03) und Chaar u. a. /In-process evaluation/ 1063-1065 (ODC11).

[586] Vgl. Chaar u. a. /In-process evaluation/ 1060 f. (ODC11) und Chillarege u. a. /In-process measurements/ 952 f. (ODC13).

[587] Vgl. Chaar u. a. /In-process evaluation/ 1066 (ODC11).

[588] Vgl. Bassin, Kratschmer, Santhanam /Software development/ 68 (ODC03).

[589] Vgl. Bassin, Kratschmer, Santhanam /Software development/ 68 (ODC03).

[590] Vgl. zu diesem Absatz Bassin, Santhanam /Maintenance/ 732 (ODC04).

[591] Vgl. Sullivan, Chillarege /Comparison/ 478-482 (ODC 25).

  - ODC-Daten zeigen in einem laufenden Projekt, dass die Qualität einer Software ausreichend ist, um den Systemtest zu beginnen.[592]
  - Die Auswertungen mit ODC unterstützen dabei, die Frage zu beantworten, ob eine Software reif ist für die Auslieferung.[593]
  - ODC-Daten zeigen in einem laufenden Projekt, dass die Entwicklung der Zuverlässigkeit der Software nicht den Erwartungen entspricht.[594]
- Bewertung der Effektivität von Verbesserungsmaßnahmen
  - ODC-Auswertungen eines Projekts werden als Vergleichsdaten festgehalten, um zu bewerten, ob geplante Verbesserungsmaßnahmen effektiv sein werden.[595]
  - Der Vergleich der ODC-Daten zweier Versionen einer Software verdeutlicht, dass aufgrund durchgeführter Verbesserungsmaßnahmen deutlich weniger Fehler ausgeliefert worden sind.[596]
  - Mittels ODC-Auswertungen wird untersucht, ob getroffene Maßnahmen in einem laufenden Projekt erfolgreich waren, um bestimmte Fehlermuster zu verhindern.[597]
- Bestimmung der Effektivität von Qualitätssicherungsprozessen
  - Anhand von ODC wird die Frage beantwortet, wie effektiv bestehende Testfallsequenzen sind.[598]
  - Mit ODC wird bewertet, wie vollständig eine Menge von Testfällen hinsichtlich der Abdeckung von Testkriterien ist.[599]
  - ODC wird eingesetzt, um nachträglich zu bewerten, wie konsequent eine definierte Teststrategie umgesetzt worden ist.[600]
  - ODC unterstützt dabei die Effektivität von Entwurfs- und Codeinspektionen zu bestimmen.[601]

---

[592] Vgl. Butcher, Munro, Kratschmer /Software testing/ 36 (ODC10).

[593] Vgl. Butcher, Munro, Kratschmer /Software testing/ 40 (ODC10) und Bassin, Kratschmer, Santhanam /Software development/ 67 f. (ODC03).

[594] Vgl. Chillarege u. a. /In-process measurements/ 948 (ODC13).

[595] Vgl. Bassin, Santhanam /Maintenance/ 732 (ODC04).

[596] Vgl. Bassin, Santhanam /Software triggers/ 107 f. (ODC05).

[597] Vgl. Bhandari u. a. /Case study/ 1164 f. (ODC07).

[598] Vgl. Butcher, Munro, Kratschmer /Software testing/ 36-38 (ODC10).

[599] Vgl. Chaar u. a. /In-process evaluation/ 1066 f. (ODC11).

[600] Vgl. Damm, Lundberg /Test process improvement/ 159.

[601] Vgl. Chaar u. a. /In-process evaluation/ 1061-1063 (ODC11) und Chillarege u. a. /In-process measurements/ 952-954 (ODC13).

- Die Bewertung des Testprozesses mit ODC macht transparent, dass ein Testkriterium einseitig betont wird und andere wiederum vernachlässigt werden.[602]
- ODC-basierte Auswertung erlaubt es, den Nutzen von Testfällen zu bewerten, die in mehreren Teststufen ausgeführt worden sind.[603]

Neben der erfolgreichen Nutzung von ODC für verschiedene Anwendungsgebiete werden in den empirischen Untersuchungen weitere Nutzenaspekte von ODC genannt.
ODC erlaubt demnach nicht nur die Existenz von Prozessmängeln festzustellen, sondern über die Analyse von Fehlern als deren Wirkung einzelne Prozessmängel einzugrenzen und zu charakterisieren.[604] Da ODC die erforderlichen Informationen zur Identifikation von Prozessmängeln bereitstellt, wird die Auswahl und Priorisierung erforderlicher Maßnahmen unterstützt.[605] Insbesondere haben Entwicklungsteams mit ODC die Möglichkeit auf die Probleme zu fokussieren, die am ehesten die Arbeit der Benutzer mit einer Software beeinträchtigen können.[606] In dem Zusammenhang ist beispielsweise die Bestimmung von Nutzungsprofilen der Kunden für eine Software auf der Grundlage ausgelieferter Fehler zu sehen.[607]
ODC stärkt Entwicklungsteams dabei, Sachverhalte bzgl. des Entwicklungsprozesses schlüssig zu erklären und aufgrund quantitativer Fakten überzeugende Verbesserungsmaßnahmen vorzuschlagen.[608]
Als eine besondere Stärke von ODC wird der Umstand gesehen, dass mit ODC alle bekannten Entwurfs- und Implementierungsfehler einer Software berücksichtigt werden und nicht nur Teilmengen von Fehlern davon.[609] Dies reduziert die Wahrscheinlichkeit, dass nicht offensichtliche, aber trotzdem relevante Fehlermuster übersehen werden.[610] Zugleich liegt der Fokus nicht auf der aufwändigen Analyse einzelner Fehler, sondern auf der von Fehlermustern und Trends in der Grundgesamtheit der Fehler.[611]

---

[602] Vgl. Chillarege, Prasad /ODC Triggers/ 667 (ODC17).
[603] Vgl. Bassin, Biyani, Santhanam /Metrics/ 24 f. (ODC01).
[604] Vgl. Bhandari u. a. /Case study/ 1158 f.
[605] Vgl. Bassin, Kratschmer, Santhanam /Software development/ 68 und Butcher, Munro, Kratschmer /Software testing/ 33 f.
[606] Vgl. Bassin, Kratschmer, Santhanam /Software development/ 73.
[607] Vgl. Bassin, Santhanam /Software triggers/ 108-112.
[608] Vgl. Dalal u. a. /Defect patterns/ 1.
[609] Vgl. Bhandari u. a. /Case study/ 1159
[610] Vgl. Bhandari u. a. /Improvement/ 208.
[611] Vgl. Butcher, Munro, Kratschmer /Software testing/ 33.

ODC bietet eine Menge von Fehlerkennzahlen, deren Zielgruppe das Entwicklungsteam[612] ist. Dies unterscheidet ODC von Kennzahlensystemen, die ebenfalls den Anspruch erheben, Mängel in einem Entwicklungsprozess aufzuzeigen, da solche Systeme sich in der Regel nicht an Entwicklungsteams richten.[613] Die Nähe von ODC zu Entwicklungsteams äußert sich darin, dass insbesondere die Fähigkeiten von Entwicklern und Testern sich kontiuierlich verbessern.[614]

Da Auswertungen mit ODC bereits begonnen werden können, sobald erste Fehler erfasst werden, kann ODC dem Entwicklungsteam schnell Feedback zu den Fragestellungen der diskutierten Anwendungsgebiete liefern und dies gegebenenfalls bereits in einem laufenden Projekt.[615]

Wird ODC in Kombination mit Attribute Focusing eingesetzt, kann die Abwesenheit von vorliegenden, jedoch noch nicht entdeckten Fehlern in der Grundgesamtheit der bekannten Fehler durch systematische Auswertungen aufgezeigt werden.[616]

5.2.3.2 Voraussetzungen und Aufwand für die Einführung und Anwendung von ODC

Im Rahmen der Literaturanalyse zu ODC wurden neben den Anwendungsgebieten und dem Nutzen von ODC auch folgende Fragen näher untersucht:

- Welche Voraussetzungen sind gemäß den empirischen Untersuchungen erforderlich, um ODC einzuführen und anzuwenden?
- Welcher Aufwand wurde für die Einführung und Anwendung von ODC festgestellt?

Voraussetzung für die Einführung und Anwendung von ODC

Tenor der empirischen Untersuchungen ist, dass wenige Voraussetzungen gegeben sein müssen, damit ODC eingeführt und angewandt werden kann. Im Mittelpunkt von ODC steht das Klassifikationsschema für Entwurfs- und Implementierungsfehler. Alle Mitglieder eines Entwicklungsteams müssen in ODC geschult werden, um Fehler korrekt gemäß ODC erfassen zu können.[617] Die Erfassung ist zweckmäßig durch ein Software-Werkzeug zu unterstützen. Es wird empfohlen, die ODC-Attribute in eine bestehende Fehlerverfolgungssoftware zu integrieren.

---

[612] Zur Definition eines Entwicklungsteams siehe Kapitel 2.2.3.

[613] Vgl. Chillarege u. a. /In-process measurements/ 943.

[614] Vgl. Butcher, Munro, Kratschmer /Software testing/ 42.

[615] Vgl. Chillarege u. a. /In-process measurements/ 954.

[616] Vgl. Bhandari u. a. /Improvement/ 200 f.

[617] Vgl. zu diesem Absatz Halliday u. a. /Experiences/ 63 f.

Die geringen Voraussetzungen für die Einführung und Anwendung von ODC äußern sich in den empirischen Untersuchungen vor allem dadurch, dass aufgeführt wird, was keine notwendige Voraussetzung ist. Hierzu zählen:

- ODC setzt nicht einen bestimmten Softwareentwicklungsprozess voraus und ist unabhängig davon, welche Art von Software entwickelt wird.[618] Diese Eigenschaft wird als Prozess- und Produktinvarianz von ODC bezeichnet.
- ODC muss nicht zu Beginn eines Softwareentwicklungsprojekts eingeführt werden.[619] Vielmehr kann die Erfassung von Fehlern gemäß ODC zu jedem Zeitpunkt im Softwareentwicklungsprozess gestartet werden.
- ODC kann unabhängig von historischen Projektdaten eingesetzt werden.[620] Folglich müssen zum einen keine historischen Projektdaten für einen zweckmäßigen Einsatz von ODC vorliegen. Sofern Daten vorliegen, erübrigen sich Fragestellungen, inwieweit sie für das aktuelle Projekt gültig sind.
- ODC kann unabhängig von Umfangskennzahlen zur Normierung wie z. B. der Anzahl Codezeilen und damit verbundenen Schwierigkeiten angewandt werden.[621]
- Werden erfasste Fehler automatisch mit Attribute Focusing ausgewertet, können auch Mitglieder eines Entwicklungsteams Fehlermuster identifizieren, ohne dass hierfür spezielle Kenntnisse für die Auswertung quantitativer Daten erforderlich sind.[622]

Aufwand von ODC

Anfänglicher Aufwand für die Einführung von ODC fließt insbesondere in die Schulung eines Entwicklungsteams, der erforderlichen Anpassung von Software-Werkzeugen wie z. B. einer Fehlerverfolgungssoftware und die Integration der ODC-Verfahrensschritte in den Softwareentwicklungsprozess.[623] Folgende quantitativen Erfahrungswerte werden in diesem Zusammenhang in den empirischen Untersuchungen aufgeführt:

- Der Aufwand für eine Schulung in ODC beträgt 3 Stunden pro Person.[624]
- Mitglieder eines Entwicklungsteams erhalten eine einstündige Einführung in ODC und eine zweistündige detaillierte Schulung darin, wie Fehler zu erfassen sind. [625]

[618] Vgl. zu diesem Absatz Bhandari u. a. /Improvement/ 182, 184.
[619] Vgl. zu diesem Absatz Butcher, Munro, Kratschmer /Software testing/ 42.
[620] Vgl. zu diesem Absatz Bhandari u. a. /Improvement/ 1159, 1162.
[621] Vgl. Bassin, Biyani, Santhanam /Metrics/ 17.
[622] Vgl. Bhandari u. a. /Improvement/ 186.
[623] Vgl. Chillarege u. a. /In-process measurements/ 954.
[624] Vgl. Bassin, Biyani, Santhanam /Sydney Olympics/ 268 und Chillarege u. a. /In-process measurements/ 954.

Ausgewählten Testern und Entwicklern wird in einer zweistündigen Schulung vermittelt, wie ODC-Daten validiert werden können. Zusätzlich werden einzelne Mitglieder in einer zweistündig geschult, wie Fehlerdaten ausgewertet und für die Entscheidungsfindung genutzt werden können.

Regelmäßiger Aufwand entsteht für die Erfassung und Auswertung von Fehlern. Hierzu werden in den empirischen Untersuchungen folgende Aussagen getätigt:

- Die nachträgliche Klassifikation von 38000 Testfällen nach dem ODC-Attribut Auslöser (trigger) dauert durchschnittlich 0,5 Minuten pro Testfall.[626]
- Der Aufwand für die Erfassung eines Fehlers gemäß ODC beträgt durchschnittlich eine bis vier Minuten pro Fehler in Abhängigkeit von der eingesetzten Fehlerverfolgungssoftware.[627]
- Sofern die Auswertung der erfassten Fehler automatisch mit Attribute Focusing erfolgt, ist der Aufwand für die Aufbereitung der Daten vernachlässigbar gering.[628] Die Interpretation von ungefähr 20 aufbereiteten Auswertungen erfolgt in einer zweistündigen Sitzung mit Schlüsselpersonen. Der Aufwand pro Sitzung entspricht folglich der Anzahl der Teilnehmer multipliziert mit 2.
- Ein Mitglied des Entwicklungsteams hat die Verantwortung dafür, dass Daten gesammelt, ausgewertet und die erforderlichen Sitzungen zeitlich eingeplant werden. [629] Diese Aktivitäten beanspruchen 10 bis 20% seiner Arbeitszeit.

5.2.3.3 Probleme und Herausforderungen bei der Einführung und Anwendung von ODC

Im Rahmen der Literaturanalyse zu ODC wurde auch die Frage untersucht, welche Probleme und Herausforderungen bei der Einführung und Anwendung von ODC empirisch festgestellt worden sind.

Als kritische Faktoren für eine erfolgreiche Einführung von ODC werden folgende Punkte genannt:[630]

---

[625] Vgl. zu diesem Absatz Butcher, Munro, Kratschmer /Software testing/ 36.

[626] Vgl. Bassin, Biyani, Santhanam: /Sydney Olympics/ 268. Statt Fehler gemäß dem Attribut Auslöser (trigger) zu klassifizieren, werden alle Testfälle gemäß diesem Attribut klassifiziert. Alle Fehler, die durch einen Testfall entdeckt werden, erhalten die gleiche Ausprägung für dieses Attribut wie der Testfall.

[627] Vgl. Chillarege u. a. /In-process measurements/ 954.

[628] Vgl. zu diesem Absatz Bhandari u. a. /Case study/ 1166 f. und Bhandari u. a. /Improvement/ 187 f.

[629] Vgl. zu diesem Absatz Bhandari u. a. /Case study/ 1166.

[630] Vgl. zum Folgenden Chillarege u. a. /In-process measurements/ 944.

- Verfügbarkeit einer zweckmäßigen Fehlerverfolgungssoftware
- Schulung des Entwicklungsteams
- Durchführung von Pilotprojekten mit ODC

Für eine unternehmensweite Einführung von ODC müssen sowohl das Management als auch Mitarbeiter auf Projektebene von dem Nutzen überzeugt werden.[631] Erfolgreiche Pilotprojekte gewährleisten alleine nicht eine erfolgreiche Umsetzung von ODC in anderen Softwareentwicklungsprojekten.[632]

Die Bedeutung der Schulungen für ODC verdeutlicht das Laborexperiment von Henningsson und Wohlin. Sie untersuchen die Frage, ob die Klassifikation von Fehlern nach dem ODC-Attribut Fehlertyp (defect type) wiederholbar, d. h. unabhängig von einzelnen Personen ist.[633] Während die Fallstudie von El Emam und Wieczorek mit der gleichen Zielsetzung zum Ergebnis hat, dass eine gute Wiederholbarkeit des Fehlerklassifikationsschemas gegeben ist und es deshalb für den praktischen Einsatz geeignet ist,[634] stellen Henningsson und Wohlin eine geringe Wiederholbarkeit fest. Als ein Grund für dieses Ergebnis wird die zu kurze Einweisung der Teilnehmer des Laborexperiments in ODC aufgeführt.[635]

Die Schulung des Entwicklungsteams stellt eine besondere Herausforderung dar, wenn eine hohe Personalfluktuation im Team vorliegt.[636]

Um eine korrekte Klassifikation von Fehlern zu gewährleisten, sind neben Schulungen die Prozesse der Datensammlung und die Fehlerdaten fortlaufend zu begutachten.[637] Sofern nicht ab dem Beginn der Datensammlung alle Fehler erfasst werden, ist die Wahrscheinlichkeit hoch, dass die Auswertungen der unvollständigen Fehlerdaten zu statistischen Verzerrung führen.

In den empirischen Untersuchungen wird vereinzelt über Schwierigkeiten bei der Anwendung des Fehlerklassifikationsschemas in bestimmten Situationen berichtet.[638]

---

[631] Vgl. Halliday u. a. /Experiences/ 62 f.

[632] Vgl. Halliday u. a. /Experiences/ 65 f.

[633] Vgl Henningsson, Wohlin /Fault classification agreement/ 95.

[634] Vgl. El Emam, Wieczorek /Repeatability/ 330.

[635] Vgl. Henningsson, Wohlin /Fault classification agreement/ 100-104.

[636] Vgl. z. B. Bassin, Biyani, Santhanam: /Sydney Olympics/ 265, 268.

[637] Vgl. zu diesem Absatz Dalal u. a. /Defect patterns/ 9.

[638] Vgl. Bhandari u. a. /Case study/ 1163 f., Lutz, Mikulski /Anomalies/ 176 und Damm, Lundberg /Test process improvement/ 154 f.

6 Herleitung eines Verfahrens zur Identifikation von Prozessmängeln

6.1 Überblick über das Verfahren

Das Hauptziel der vorliegenden Arbeit lautet:[639]

> *Wie können Mängel im Prozess der Anforderungsanalyse bereits während der Entwicklung einer Anwendungssoftware gefunden werden, indem Fehler erfasst und ausgewertet werden?*

In dem vorliegenden Kapitel wird ein entsprechendes Fehleranalyseverfahren mit dieser Zielsetzung vorgeschlagen.

Ein Prozessmangel in der Anforderungsanalyse verursacht Anforderungsfehler, welche wiederum in Entwurfs- und Implementierungsfehler als Folgefehler münden können. Folglich sollte ein zweckmäßiges Fehleranalyseverfahren zur Aufdeckung von Prozessmängeln in der Anforderungsanalyse neben Anforderungsfehlern auch Entwurfs- und Implementierungsfehler berücksichtigen.

ODC-Klassifikationsschema für Entwurfs- und Implementierungsfehler

Für Entwurfs- und Implementierungsfehler liegt mit der Orthogonal Defect Classification ein Vorschlag mit folgenden Merkmalen vor:[640]

- ODC baut auf einem Fehlerklassifikationsschema auf, das seit Anfang der Neunziger auf der Grundlage empirischer und praktischer Erfahrungen ständig weiterentwickelt wurde.
- Es gibt eine Vielzahl empirischer Untersuchungen zu ODC. Tenor dieser Untersuchungen ist nahezu übereinstimmend, dass ODC in der Praxis erfolgreich anwendbar ist.
- ODC kann für neun unterschiedliche Anwendungsgebiete in der Praxis eingesetzt werden.[641] Diese lauten im Einzelnen:

[639] Vgl. Kapitel 1.2.
[640] Vgl. Kapitel 5.1.
[641] Vgl. Kapitel 5.2.3.1.

- Aufzeigen von Verbesserungspotenzialen in Entwicklungsprozessen (Anforderungsanalyse, Softwareentwurf und Implementierung)[642] für ein laufendes Softwareentwicklungsprojekt.
- Aufzeigen von Verbesserungspotenzialen in Entwicklungsprozessen (Anforderungsanalyse, Softwareentwurf und Implementierung) für zukünftige Softwareentwicklungsprojekte.
- Aufzeigen von Verbesserungspotenzialen in Prozessen der Qualitätssicherung[643] für ein laufendes Softwareentwicklungsprojekt.
- Aufzeigen von Verbesserungspotenzialen in Prozessen der Qualitätssicherung für zukünftige Softwareentwicklungsprojekte.
- Planung und Steuerung der Qualitätssicherungsprozesse in laufenden Softwareentwicklungsprojekten.
- Steuerung der Beziehungen zwischen Auftraggeber und -nehmer bei einer ausgelagerten Softwareentwicklung.
- Bestimmung der Qualität einer Software und Risikobewertung.
- Bewertung der Effektivität von Verbesserungsmaßnahmen.
- Bestimmung der Effektivität von Qualitätssicherungsprozessen.

Die Orthogonal Defect Classification ist ein etabliertes Verfahren, um die Ursachen von Entwurfs- und Implementierungsfehlern zu bestimmen. Diese Ursachen können, müssen aber nicht im Prozess der Anforderungsanalyse liegen. Folglich ist ODC nicht ein Verfahren, dass ausschließlich für die Aufdeckung von Prozessmängeln in der Anforderungsanalyse gedacht ist. Dies ist nicht als Schwäche, sondern als Stärke zu werten. Es erscheint nicht zweckmäßig, für verschiedene Teilproblemstellungen jeweils separate Verfahren einzusetzen.
Jedoch ist ODC nur rudimentär theoretisch fundiert. Ebenso gibt es keine Hinweise, wie mittels dieses Verfahrens gezielt die Suche nach Prozessmängeln in der Anforderungsanalyse unterstützt werden kann. Die vorliegende Arbeit versucht diese Lücken zu schließen.

Fehlerklassifikationsschema für Anforderungsfehler

Zur Klassifikation von Fehlern existieren zahlreiche Vorschläge in der wissenschaftlichen Literatur.[644] Die Klassifikation einzelner Anforderungsfehler gestaltet sich schwierig.[645] Im Ge-

[642] Vgl. Kapitel 2.2.3.
[643] Vgl. Kapitel 2.4.
[644] Vgl. z. B. Sakthivel /Survey/ 68-70, Hayes /Requirement fault/ 51 f. und Schneider, Martin, Tsai /Study/ 193 f.
[645] Vgl. Lanubile, Shull, Basili /Requirements documents/ 116.

gensatz zu klar definierbaren Konstrukten bei Entwurfs- und Implementierungsfehlern (wie z. B. Algorithmus, Bedingung, Schnittstelle etc.) ist dies ohne weiteres bei Anforderungsfehlern nicht möglich. Anforderungen können z. B. natürlich-sprachlich oder semiformal beschrieben sein und ihre Beschreibung kann sich hinsichtlich der Granularität unterscheiden. So adaptieren beispielsweise Freimut und Denger ausgewählte ODC-Attribute für die Anforderungsfehler.[646] Jedoch geht das vorgeschlagene Klassifikationsschema von bestimmten Darstellungsmitteln wie z. B. Anwendungsfällen (Use Cases) aus.
In der vorliegenden Arbeit wird vorgeschlagen für Anforderungsfehler nur den Typ gemäß der in Kapitel 4.3.4.2 hergeleiteten Typologie zu erfassen.

Auswertung klassifizierter Fehler

Die Auswertung klassifizierter Fehler lehnt sich an ODC an (vgl. Kapitel 5.1.3.4). Neben Attribute Focusing werden Auswertungen für die gezielte manuelle Suche nach Fehlermustern vorgeschlagen, die die Suche nach Prozessmängeln in der Anforderungsanalyse unterstützen.

Ursachenanalyse

Hinweise für die Ursachenanalyse fallen in der ODC-Literatur sehr knapp aus. Um diesem Problem entgegenzuwirken, wird ein Leitfaden für die Ursachenanalyse empfohlen, der insbesondere die Identifizierung von Prozessmängeln in der Anforderungsanalyse zum Anspruch hat. Die Ursachenanalyse kann durch Rückgriff auf die Erklärungsmustern für bestimmte Typen von Anforderungsfehlern in Kapitel 4.4.2 unterstützt werden. Diese Erklärungsmuster sind einzelne hypothetische Gesetzmäßigkeiten, die erklären, warum bestimmte Typen von Anforderungsfehlern entstehen.

Theoretische Fundierung des Verfahrens

Das Verfahren baut auf dem systematisch hergeleiteten Erklärungsmodell in Kapitel 4 auf (vgl. Abbildung 6-1).

[646] Vgl. Freimut, Denger /Defect classification scheme/ 9-14.

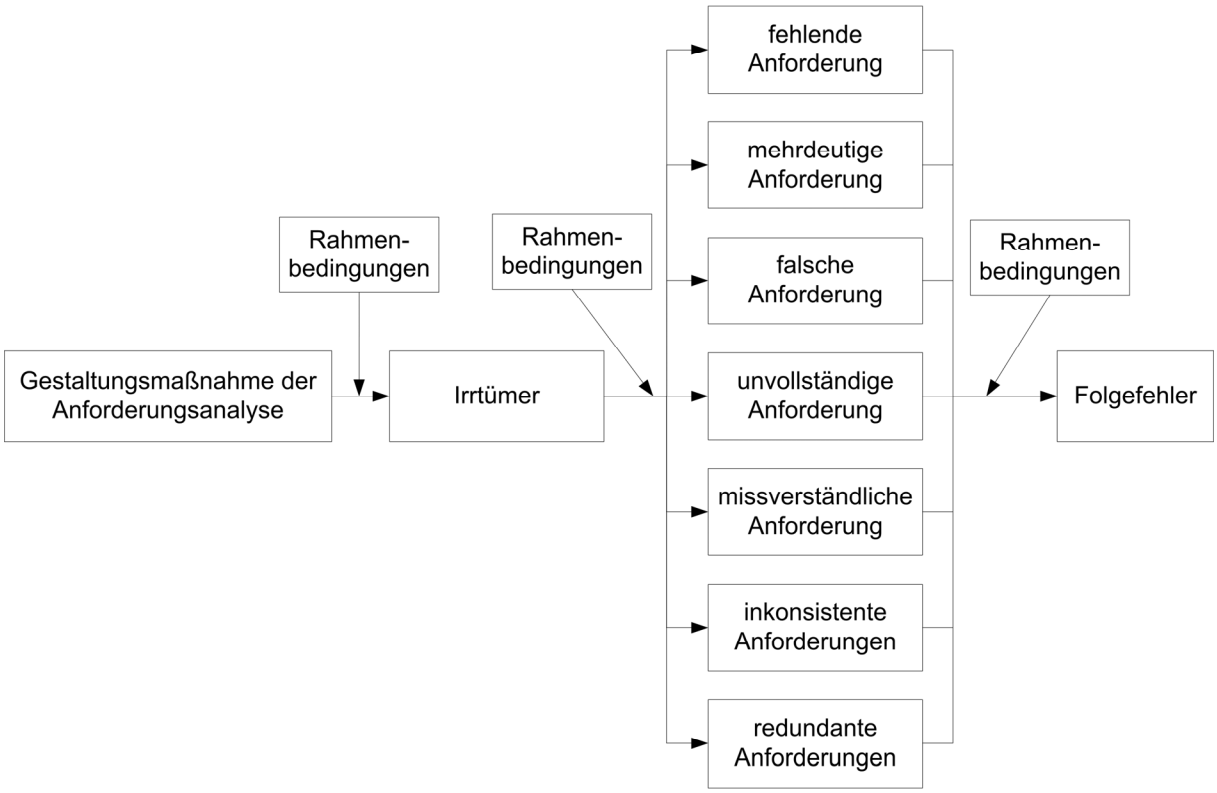


Abbildung 6-1: Theoretischer Ansatz des Erklärungsmodells für Anforderungsfehler und ihre Folgefehler

Ausgehend von dem Verständnis, dass ein Prozess ein Bündel von Gestaltungsmaßnahmen ist, ist ein Prozessmangel der Anforderungsanalyse eine Maßnahme, die *menschliche Irrtümer* begünstigt.[647] Ein Irrtum liegt vor, wenn eine geplante Abfolge von geistigen oder körperlichen Aktivitäten ohne Fremdeinwirkung nicht zum beabsichtigten Ergebnis führt. Voraussetzung für einen Irrtum ist die Absicht zur Handlung. Eine Absicht ist gegeben, wenn dem Individuum das erwartete Ergebnis und die Mittel bewusst sind, mit denen das Ergebnis erreicht werden soll.

Unterliegt ein Projektmitarbeiter einem Irrtum während er Teilaufgaben der Anforderungsanalyse ausführt, kann sein Irrtum dazuführen, dass Anforderungsfehler entstehen. Ein *Fehler* ist ein unzureichendes Merkmal oder ein erwartetes, jedoch fehlendes Merkmal eines Arbeitsergebnisses der Softwareentwicklung, sofern es eine Änderung in diesem Ergebnis notwendig macht. Fehler, die infolge eines Irrtums neu eingeführt werden, sind *originäre Fehler*.

Irrtümer in der Anforderungsanalyse verursachen systematisch bestimmte Typen von Anforderungsfehlern. Wenn sich ein Anforderungsfehler auf eine Anforderung bezieht, können insgesamt fünf Fälle voneinander abgegrenzt werden:

- Für ein bestehendes Kundenbedürfnis wurde eine Anforderung gänzlich übersehen (*fehlende Anforderung*).

[647] Vgl. zum Folgenden Kapitel 4.2.

- Die Beschreibung der Anforderung kann auf mehrere Arten gedeutet werden (*mehrdeutige Anforderung*).
- Die Anforderung erfüllt kein Kundenbedürfnis oder erfüllt sie nicht korrekt (*falsche Anforderung*).[648]
- Die Anforderung ist unvollständig beschrieben (*unvollständige Anforderung*).
- Die Anforderung ist zwar vollständig beschrieben, jedoch fehlen Kontextinformationen zu dieser Anforderung oder Verknüpfungen zu vorhandenen Kontextinformationen, um die Anforderung zu verstehen (*missverständliche Anforderung*). Kontextinformationen sind z. B.
  - Kundenbedürfnisse, die eine Anforderung begründen,
  - für die Anforderung relevante Kundengruppen oder
  - andere abhängige Anforderungen.

Ein Anforderungsfehler kann sich auch auf eine Beziehung zwischen zwei oder mehreren Anforderungen beziehen. Zwei Fälle können dabei unterschieden werden:

- Eine Anforderung steht im Widerspruch zu einer oder mehreren anderen Anforderungen (*inkonsistente Anforderungen*).
- Eine Anforderung ist redundant zu mindestens einer anderen Anforderung (*redundante Anforderungen*).

Die 7 Typen von Anforderungsfehlern können Folgefehler in Entwurf und Implementierung verursachen, wenn sie nicht frühzeitig entdeckt werden. Diese *Folgefehler* haben in Abgrenzung zu originären Fehlern ihre Ursache in einem Fehler einer vorgelagerten Entscheidung.

Die Wirkung einzelner Gestaltungsmaßnahmen ist abhängig von den Rahmenbedingungen, unter denen sie durchgeführt werden.[649] Folglich sind auch Prozessmängel situativ.[650]

6.2 Fehlerklassifikationsschema

6.2.1 Überblick

Für Anforderungsfehler wird nur der Typ des Fehlers erfasst (vgl. Tabelle 6-1). Hierbei wird auf die in Kapitel 6.2.2 im Rahmen einer Literaturanalyse hergeleitete Typologie zurückgegriffen.

---

[648] Vgl. hierzu Kapitel 2.2.1.

[649] Vgl. hierzu Abbildung 1-3.

[650] Vgl. hierzu die Fußnote 32 und die dort aufgeführten Literaturquellen.

Attribut	Bedeutung des Attributwertes	Attributwerte
Typ des Anforderungsfehlers	Was für ein Anforderungsfehler liegt vor?	▪ fehlende Anforderung ▪ mehrdeutige Anforderung ▪ falsche Anforderung ▪ unvollständige Anforderung ▪ missverständliche Anforderung ▪ inkonsistente Anforderungen ▪ redundante Anforderungen

Tabelle 6-1: Attribut für Anforderungsfehler

Das Klassifikationsschema für Entwurfs- und Implementierungsfehler ist weitgehend an dem ODC-Klassifikationsschema aus IBM Research /ODC/ angelehnt (vgl. Kapitel 6.2.3). Jedoch wurden den Attributen bei Bedarf prägnantere Namen gegeben (vgl. Tabelle 6-2) und die Erläuterungen zu den Attributwerten präzisiert. Bei dem Attribut „Gegenstand der Änderung“ (vormals Fehlertyp) wurde ein zusätzlicher Wert „Entwicklerdokumentation“ hinzugefügt (vgl. Tabelle 6-10), damit der gleiche Wert im Attribut QS-Kriterium (vormals Auslöser) eine Entsprechung hat.

Name des ODC-Attributs	Name geändert in
Aktivität (activity)	QS-Aktivität
Auslöser (trigger)	QS-Kriterium
Auswirkung (impact)	Auswirkung auf Kunden
Fehlertyp (defect type)	Gegenstand der Änderung
Kennzeichner (qualifier)	Art der Änderung
Ursprung (source)	Ursprung
Alter (age)	Vorgeschichte

Tabelle 6-2: Änderungen von Attributnamen

Zwei weitere Attribute werden hinzugefügt: Phase und Komponente. Das Attribut Phase erlaubt die Unterscheidung und damit die Auswertung von Entwicklungs- und Produktionsfehlern. Empirische Untersuchungen zu ODC, in denen sowohl Entwicklungs- und Produktionsfehler berücksichtigt werden, nutzen dieses Attribut, ohne es in der Regel explizit zu nennen.[651] Das Attribut Komponente ermöglicht es, spezifische Auswertungen zu einzelnen technischen Bestandteilen einer Anwendungssoftware durchzuführen. Dieses Attribut wird in zahlreichen empirischen Untersuchungen zu ODC bereits als zusätzliches Attribut aufgeführt.[652]

[651] Vgl. hierzu die empirischen Untersuchungen zu ODC in Anhang A.2.

[652] Vgl. z. B. Bhandari u. a. /Case study/ 1159, Bhandari u. a. /Improvement/ 185, Butcher, Munro, Kratschmer /Software testing/ 34, Chillarege, Prasad /ODC Triggers/ 676 und Christmansson, Santhanam /Error injection/ 179.

Werden entdeckte Fehler zudem fachlichen Architekturkomponenten zugeordnet, wird die Möglichkeit eröffnet, komponentenspezifische Maßnahmen zu ergreifen.[653] Einen Überblick über die neuen und bestehenden ODC-Attribute geben die Tabelle 6-3 (Attribute nach einer Fehlerentdeckung) und Tabelle 6-4 (Attribute nach einer Fehlerbehebung).

Attribut	Bedeutung des Attributwertes	Attributwerte
Phase	In welcher Phase wurde der Fehler gefunden?	▪ Entwicklung ▪ Produktion
QS-Aktivität	In welcher Prüf- oder Test-Aktivität wurde der Fehler gefunden?	*Projektspezifische Konfiguration erforderlich*. Z. B. Inspektion des Entwurfsdokuments, Codeinspektion, Modultest, Funktionstest, Systemtest.
QS-Kriterium	Nach welchem Kriterium wurde geprüft oder getestet?	▪ Entwurfskonformität ▪ Kontroll- und Datenfluss ▪ Abwärtskompatibilität ▪ Umfeldkompatibilität ▪ Steuerung gemeinsam genutzter Ressourcen ▪ Entwicklerdokumentation ▪ Besonderheiten der Entwicklungssprache ▪ Nebenwirkung ▪ seltene Situationen ▪ einfacher Pfad ▪ komplexer Pfad ▪ Funktionsabdeckung ▪ Funktionsvariation ▪ Funktionensequenz ▪ Funktioneninteraktion ▪ Leistungs- oder Stresstest ▪ Fehlerbehandlung / Ausnahmebehandlung ▪ Start / Neustart ▪ Hardwarekonfiguration ▪ Softwarekonfiguration ▪ blockierter Test
Auswirkung auf Kunden	Wie hätte sich oder hat sich der Fehler nach Auslieferung beim Kunden ausgewirkt?	▪ Installierbarkeit ▪ Analysierbarkeit von Fehlverhalten ▪ Standards für Softwareprodukte ▪ Integrität / Sicherheit ▪ Versionswechsel ▪ Zuverlässigkeit ▪ Zeitverhalten ▪ Benutzerdokumentation in der Software ▪ unbekannte Softwareanforderung ▪ Wartbarkeit ▪ Benutzbarkeit ▪ Barrierefreiheit ▪ bekannte Softwareanforderung

Tabelle 6-3: Attribute für Entwurfs- und Implementierungsfehler (nach einer Fehlerentdeckung)

[653] Vgl. Bhandari u. a. /Improvement/ 202-204.

Attribut	Bedeutung des Attributwertes	Attributwerte
Komponente	Welche Komponente des Arbeitsergebnisses enthielt den Fehler?	*Projektspezifische Konfiguration erforderlich*.
Gegenstand der Änderung	Was musste geändert werden, um den Fehler zu beheben?	▪ Zuweisung / Initialisierung ▪ Bedingung ▪ Algorithmus ▪ Funktion ▪ Timing / Serialisierung ▪ interne Schnittstellen ▪ Beziehung
Art der Änderung	Wie wurde der Fehler behoben?	▪ eingefügt ▪ modifiziert ▪ entfernt
Ursprung	Woher stammt der fehlerbehaftete Teil des Arbeitsergebnisses ursprünglich?	▪ Eigenentwicklung ▪ Wiederverwendung aus Standard-Bibliothek ▪ Entwicklung durch externen Auftragnehmer ▪ portiert
Vorgeschichte	Welche Vorgeschichte weist der Fehler auf?	▪ alter Fehler ▪ neuer Fehler ▪ Fehler infolge einer Überarbeitung ▪ Fehler infolge einer Fehlerkorrektur

Tabelle 6-4: Attribute für Entwurfs- und Implementierungsfehler (nach einer Fehlerbehebung)

6.2.2 Attribute und Attributwerte für Anforderungsfehler

Attributwert	Bedeutung des Attributwertes
fehlende Anforderung	Für ein bestehendes Kundenbedürfnis wurde eine Anforderung gänzlich übersehen.
mehrdeutige Anforderung	Die Beschreibung der Anforderung kann auf mehrere Arten gedeutet werden.
falsche Anforderung	Die Anforderung erfüllt kein Kundenbedürfnis oder erfüllt sie nicht korrekt.
unvollständige Anforderung	Die Anforderung ist unvollständig beschrieben.
missverständliche Anforderung	Die Anforderung ist zwar vollständig beschrieben, jedoch fehlen Kontextinformationen zu dieser Anforderung oder Verknüpfungen zu vorhandenen Kontextinformationen, um die Anforderung zu verstehen. Kontextinformationen sind z. B. ▪ Kundenbedürfnisse, die eine Anforderung begründen, ▪ für die Anforderung relevante Kundengruppen oder ▪ andere abhängige Anforderungen.
inkonsistente Anforderungen	Eine Anforderung steht im Widerspruch zu einer oder mehreren anderen Anforderungen.
redundante Anforderungen	Eine Anforderung ist redundant zu mindestens einer anderen Anforderung.

Tabelle 6-5: Werte des Attributs „Typ des Anforderungsfehlers“

6.2.3 Attribute und Attributwerte für Entwurfs- und Implementierungsfehler

Die Werte für die Attribute QS-Aktivität und Komponente sind für ein gegebenes Softwareentwicklungsprojekt zu definieren, weshalb sie hier nicht aufgeführt werden. Für die übrigen Attribute werden in jeweils einer Tabelle die Attributwerte einzeln erläutert.

Attributwert	Bedeutung des Attributwertes
Entwicklung	Der Fehler wurde vor der Freigabe der Software für die Nutzung seitens der Kunden gefunden.
Produktion	Der Fehler wurde nach der Freigabe der Software für die Nutzung seitens der Kunden gefunden.

Tabelle 6-6: Werte des Attributs „Phase“ und ihre Bedeutung

Attributwert	Bedeutung des Attributwertes
Entwurfskonformität	Wurde das Prüfobjekt dahingehend untersucht, ob es mit vorhergehenden Arbeitsergebnissen und für das Projekt gültigen Entwicklungsstandards konform ist?
Kontroll- und Datenfluss	Wurde das Prüfobjekt dahingehend untersucht, ob der Kontroll- und Datenfluss im Prüfobjekt korrekt und vollständig sind?
Abwärtskompatibilität	Wurde das Prüfobjekt dahingehend untersucht, ob die im Prüfobjekt beschriebenen Funktionen abwärtskompatibel mit früheren Versionen des Produkts sind?
Umfeldkompatibilität	Wurde das Prüfobjekt dahingehend untersucht, ob die beschriebenen Funktionen im Prüfobjekt kompatibel zu Softwareprodukten im Umfeld sind, mit denen es interagieren muss?
Steuerung gemeinsam genutzter Ressourcen	Wurde das Prüfobjekt dahingehend untersucht, ob die Steuerung gemeinsam genutzter Ressourcen abgestimmt ist?
Entwicklerdokumentation	Wurde das Prüfobjekt dahingehend untersucht, ob die Entwicklerdokumentation (z. B. Kommentare im Code) im oder zum Prüfobjekt eine falsche, widersprüchliche oder unvollständige Information enthält?
Besonderheiten der Entwicklungssprache	Wurde das Prüfobjekt dahingehend untersucht, ob Besonderheiten der Entwicklungssprache bei der Implementierung einer Komponente oder Funktion berücksichtigt sind?[654]
Nebenwirkung	Wurde das Prüfobjekt dahingehend untersucht, ob es bei einem gewöhnlichen Anwendungsfall unerwünschte Nebenwirkungen im Verhalten der Software verursacht, die außerhalb des Prüfobjekts liegen?
seltene Situation	Wurde das Prüfobjekt dahingehend untersucht, ob es bei einem nicht vorgesehenen Anwendungsfall unerwünschtes Verhalten der Software verursacht?
einfacher Pfad	Wurde das Testobjekt mit Testfällen getestet, um jeweils einen bestimmten Pfad im Kontrollfluss auszuführen?
komplexer Pfad	Wurde das Testobjekt mit Testfällen getestet, um mehrere Zweige unter verschiedenen Bedingungen auszuführen?
Funktionsabdeckung	Wurde die Software mit Testfällen getestet, um einzelne Funktionen jeweils ohne Eingabeparameter oder, falls erforderlich, mit genau einer Ausprägung von Eingabeparametern auszuführen?
Funktionsvariation	Wurde die Software mit Testfällen getestet, um einzelne Funktionen jeweils mit unterschiedlichen Ausprägungen von Eingabeparametern auszuführen?

Tabelle 6-7: Werte des Attributs „QS-Kriterium“ und ihre Bedeutung

[654] Eine Besonderheit einer Entwicklungssprache ist zum Beispiel, dass in C statt dem Vergleichsoperator == versehentlich der Zuweisungsoperator = genutzt wird.

Attributwert	Bedeutung des Attributwertes
Funktionensequenz	Wurde die Software mit Testfällen getestet, um mehrere Funktionen hintereinander in einer Sequenz auszuführen? Dieses QS-Kriterium darf nur ausgewählt werden, wenn die Funktionen einzeln erfolgreich ausgeführt werden können.
Funktioneninteraktion	Wurde die Software mit Testfällen getestet, um mehrere Funktionen auszuführen, die miteinander interagieren? Die Interaktion beschränkt sich nicht darauf, dass sie in einer Sequenz ausgeführt werden. Dieses QS-Kriterium darf nur ausgewählt werden, wenn die Funktionen einzeln erfolgreich ausgeführt werden können.
Leistungs- oder Stresstest	Wurde die Software mit Testfällen getestet, um das Verhalten der Software bei geringer und hoher Auslastung zu untersuchen?
Fehlerbehandlung / Ausnahmebehandlung	Wurde die Software mit Testfällen getestet, um das Verhalten der Software im Fall eines Fehlverhaltens oder einer Ausnahmesituation zu untersuchen?
Start / Neustart	Wurde die Software mit Testfällen getestet, um das Verhalten der Software beim Start oder Neustart zu untersuchen, nachdem sie ausgefallen oder beendet wurde?
Hardwarekonfiguration	Wurde die Software mit Testfällen getestet, um das Verhalten der Software unter verschiedenen Hardware-Konfigurationen zu untersuchen?
Softwarekonfiguration	Wurde die Software mit Testfällen getestet, um die Software unter verschiedenen Software-Konfigurationen zu untersuchen?
blockierter Test	Wurde die Durchführung eines Testfalls im Systemtest beabsichtigt, jedoch durch das Auftreten eines Fehlers verhindert, der außerhalb des Fokus des Testfalls liegt?

Tabelle 6-8: Werte des ODC-Attributs „QS-Kriterium“ und ihre Bedeutung (Fortsetzung)

Attributwert	Bedeutung des Attributwertes
Installierbarkeit	Die Kunden können die Software nicht oder nur eingeschränkt in einer festgelegten Umgebung installieren. (Aspekte der Benutzbarkeit werden hier nicht berücksichtigt.)
Integrität / Sicherheit	Ein unberechtigter Zugriff (versehentlich oder vorsätzlich) auf die Software und ihre Daten wird begünstigt.
Zeitverhalten	Die Software arbeitet in der Wahrnehmung der Kunden zu langsam (Antwort- und Verarbeitungszeit, Durchsatz bei der Funktionsausführung).
Wartbarkeit	Änderungen an der Software sind nur eingeschränkt oder nur mit hohem Aufwand möglich. Z. B. Fehlerbeseitigungen (Patch), Verbesserungen oder Anpassungen an Umgebungsänderungen (Update).
Analysierbarkeit von Fehlverhalten	Die Analyse von Fehlverhalten der Software wird erschwert.
Versionswechsel	Der Wechsel auf die neue Version der Software wird erschwert.
Benutzerdokumentation in der Software	Die in die Software integrierte Benutzerdokumentation (z. B. Hilfetexte, Hinweise, Bildschirmmeldungen) ist unvollständig oder falsch.
Benutzbarkeit	In der Wahrnehmung der Kunden gestaltet sich die Verständlichkeit, Erlernbarkeit oder Bedienbarkeit der Software als schwierig.
Standards für Softwareprodukte	Die Software genügt nicht verbreiteten und erwarteten (Quasi-)Standards bei der Bedienung und der Funkionalität.
Zuverlässigkeit	Die Software arbeitet für einen betrachteten Zeitrahmen nicht zuverlässig, weil ihre korrekte Ausführung durch unbeabsichtigte Ausfälle oder Abbrüche unterbrochen wird.
unbekannte Softwareanforderung	Der Kunde vermisst ein erwartetes Merkmal der Software, das nicht umgesetzt wurde, weil eine Softwareanforderung übersehen, nicht verstanden oder fälschlicherweise für die neue Version eine Software als nicht erforderlich bewertet wurde.
Barrierefreiheit	Kunden mit körperlichen Einschränkungen können die Software aufgrund bestimmter Merkmale nur erschwert nutzen.
bekannte Softwareanforderung	Die Umsetzung einer bekannten Softwareanforderung entspricht nicht der Erwartung der Kunden.

Tabelle 6-9: Werte des Attributs „Auswirkung auf Kunden“ und ihre Bedeutung

Attributwert	Bedeutung des Attributwertes
Entwicklerdokumentation	In dem Arbeitsergebnis integrierte technische Dokumentation (z. B. Codekommentar).
Zuweisung / Initialisierung	Wenige Codezeilen wurden geändert: eine Zuweisung von Werten, die Initialisierung von Kontrollstrukturen oder Datenstrukturen fehlte oder war falsch. Sofern mehrere zusammenhängende Änderungen dieser Art durchgeführt werden mussten, liegt ein Algorithmusfehler vor.
Bedingung	Die Überprüfung von Parametern oder Daten in Bedingungen fehlte oder war falsch. Änderungen müssen sich nicht nur auf eine Bedingung beschränken, sondern können auch zur Folge haben, dass ein Block von Anweisungen für einen neuen Bedingungszweig eingefügt wird.
Algorithmus	Aufgrund von Problemen bzgl. der Effizienz oder Korrektheit wurde ein Algorithmus oder eine lokale Datenstruktur geändert, die nicht die Änderung einer Entscheidung des Grobentwurfs erfordert.
Funktion	Ein Funktionsfehler beeinträchtigt eine vom Kunden erwartete Funktion, Benutzerschnittstellen, externe Programmierschnittstellen der Software (API), Schnittstellen mit der Hardwarearchitektur oder globale Datenstrukturen.
Timing / Serialisierung	Die Abstimmung des Zugriffs auf eine gemeinsam genutzte Ressource fehlt oder ist fehlerhaft.
interne Schnittstellen	Änderungen sind erforderlich aufgrund von Problemen bei der Kommunikation zwischen ▪ Modulen, ▪ Komponenten, ▪ Gerätetreibern, ▪ Objekten, ▪ Prozeduren mittels ▪ Makros, ▪ Prozeduraufrufen, ▪ Kontrollblöcken, ▪ Parameterlisten.
Beziehung	Änderungen sind erforderlich aufgrund von Problemen in Beziehungen zwischen Prozeduren, Datenstrukturen und Objekten.

Tabelle 6-10: Werte des Attributs „Gegenstand der Änderung“ und ihre Bedeutung

Attributwert	Bedeutung des Attributwertes
eingefügt	Etwas Fehlendes wurde eingefügt.
modifiziert	Etwas Bestehendes war falsch und wurde korrigiert.
entfernt	Etwas Bestehendes war irrelevant und wurde entfernt.

Tabelle 6-11: Werte des Attributs „Art der Änderung“ und ihre Bedeutung

Attributwert	Bedeutung des Attributwertes
Eigenentwicklung	Der fehlerbehaftete Teil des Arbeitsergebnisses wurde durch das Entwicklungsteam der eigenen Organisation erstellt.
Wiederverwendung aus Standard-Bibliothek	Der fehlerbehaftete Teil des Arbeitsergebnisses wurde aus einer Standard-Bibliothek wieder verwendet.
Entwicklung durch externen Auftragnehmer	Der fehlerbehaftete Teil des Arbeitsergebnisses wurde außerhalb der eigenen Organisation durch einen externen Anbieter erstellt.
portiert	Der fehlerbehaftete Teil des Arbeitsergebnisses wurde von einer anderen Plattform portiert.

Tabelle 6-12: Werte des Attributs „Ursprung“ und ihre Bedeutung

Attributwert	Bedeutung des Attributwertes
alter Fehler	Der Fehler ist Teil einer Funktion, die weder im aktuellen Projekt modifiziert wurde noch aus einer Standard-Bibliothek wieder verwendet wurde. Folglich handelt es sich um einen alten Fehler, der übersehen und nicht durch das aktuelle Projekt eingeführt worden ist.
neuer Fehler	Der Fehler ist in einer Funktion, die für aktuelle Version der Software neu entwickelt wurde.
Fehler infolge einer Überarbeitung	Der Fehler wurde eingeführt, nachdem eine alte Funktion überarbeitet wurde.
Fehler infolge einer Fehlerkorrektur	Der Fehler wurde im Rahmen der Korrektur eines anderen Fehlers eingeführt.

Tabelle 6-13: Werte des Attributs „Vorgeschichte“ und ihre Bedeutung

6.3 Konfiguration

Konfiguration des Verfahrens für Anforderungsfehler

Die Typen eines Anforderungsfehlers (vgl. Tabelle 6-5) können analog zum ODC-Attribut „Fehlertyp“ mit Teilprozessen der Anforderungsanalyse projektspezifisch verknüpft werden. Dies erlaubt, Rückschlüsse über den Prozess einzelner Teilaufgaben der Anforderungsanalyse zu ziehen. In Kapitel 2.3.2 wurden vier Teilaufgaben unterschieden: Anforderungen erheben, dokumentieren, validieren und verwalten. Die Tabelle 6-14 zeigt eine examplarische Verknüpfung.

	Teilprozesse der Anforderungsanalyse			
Attributwert	Anforderungen erheben	Anforderungen dokumentieren	Anforderungen validieren	Anforderungen verwalten
fehlende Anforderung	x			
mehrdeutige Anforderung		x		
falsche Anforderung	x			
unvollständige Anforderung		x		
missverständliche Anforderung		x		
inkonsistente Anforderungen			x	x
redundante Anforderungen				x

Tabelle 6-14: Verknüpfungen von Anforderungsfehlertypen und Teilprozessen der Anforderungsanalyse (Beispiel)

Diese Verknüpfungstabelle ist für die Auswertung klassifizierter Anforderungsfehler bedeutsam und hat keinen Einfluss auf die Fehlerklassifikation.

Konfiguration des Verfahrens für Entwurfs- und Implementierungsfehler

Folgende Attribute müssen projektspezifisch konfiguriert werden:

- QS-Aktivität

  Für das Attribut QS-Aktivität muss vor der Fehlerklassifikation zunächst der projektspezifische Wertebereich bestimmt werden.[655] Folglich ist zu klären, welche QS-Aktivitäten geplant sind.

- QS-Kriterium (vormals Auslöser)

  Die definierten Werte des Attributs QS-Aktivität sind mit den Werten des Attributs QS-Kriterium zu verknüpfen.[656] Die zu Grunde liegende Fragestellung lautet: Welche Prüf- oder Testkriterien kommen in einem projektspezifischen QS-Prozess zum Einsatz? Liegt eine Verknüpfungstabelle vor, kann dies die Klassifikation von Fehlern erleichtern. Ein Tester, der in einem bestimmten QS-Prozess einen Fehler findet, muss nicht aus allen möglichen Werten des Attributs QS-Kriterium auswählen, sondern kann sich auf Werte beschränkten, die mit dem jeweiligen QS-Prozess verknüpft sind.

- Phase

  Dieses Attribut dient zur Unterscheidung von Entwicklungs- und Produktionsfehlern. Der Wertebereich ist bereits vorgegeben. Um jedoch für einen gefundenen Fehler den korrekten Wert auszuwählen, muss bestimmt werden, wann ein Softwareentwicklungsprojekt begonnen hat und wann ein Softwarerelease für die produktive Nutzung freigegeben wurde.

- Gegenstand der Änderung (vormals Fehlertyp)

  Für das Attribut „Gegenstand der Änderung“ müssen zwei Prozessverknüpfungstabellen angelegt werden.[657] Zum einen sind die Werte dieses Attributs mit Prozessen der Entwicklungsaufgaben zu verknüpfen. Eine Verknüpfung beschreibt, welcher Fehlertyp in welchen projektspezifischen Prozessen für Entwicklungsaufgaben entstehen kann. Zum anderen sind die Werte dieses Attributs mit den Werten des Attributs QS-Aktivität und damit mit Prozessen der Qualitätssicherung zu verknüpfen. In diesen Fällen beschreibt eine Verknüpfung, welcher Fehlertyp in welchen projektspezifischen QS-Prozessen gefunden werden soll.

---

[655] Vgl. zu diesem Absatz IBM Research /ODC/ o. S.: “One of the first things an organization must do once they have decided to implement ODC is to define the activities they perform and map the triggers to those activities. Note that although the organization defines their activities, they do not define or redefine the triggers.”

[656] Für eine detaillierte Darstellung siehe Kapitel 5.1.3.2.

[657] Für eine detaillierte Darstellung siehe Kapitel 5.1.3.2.

Es sei darauf hingewiesen, dass die Konfiguration des Attributs „Gegenstand der Änderung“ erst für die Auswertung klassifizierter Fehler und die Ursachenanalyse bedeutsam ist und keinen Einfluss auf die Fehlerklassifikation hat. Folglich kann die Konfiguration dieses Attributs auch zeitnah vor der Diskussion der Ergebnisse einer Fehlerauswertung erfolgen.[658]

- Komponente

  Jede untersuchte Software kann in Architekturkomponenten zerlegt werden. Für die Zerlegung ist entscheidend, dass Fehler eindeutig genau einer Komponente zugeordnet werden können. In welche Komponenten eine Software zerlegt wird, ist insbesondere davon abhängig, zu welchen Bestandteilen zu einer Software separate Fehlerauswertungen gewünscht werden.

6.4 Fehler klassifizieren

Anforderungsfehler sind möglichst zeitnah nach ihrer Entdeckung zu klassifizieren. Sie können entdeckt werden, wenn ein Anforderungsdokument geprüft wird oder Entwurfs- und Implementierungsfehler auf einen Anforderungsfehler als Ursache schließen lassen.
Für die Klassifikation von Entwurfs- und Implementierungsfehlern gelten die Ausführungen zu ODC in Kapitel 5.1.3.3.

6.5 Klassifizierte Fehler auswerten

Die Auswertung klassifizierter Fehler lehnt sich an ODC an (vgl. Kapitel 5.1.3.4). Attribute Focusing wird eingesetzt, um potenzielle Fehlermuster automatisch aufzuzeigen. Fehlermuster sind Teilmengen von Fehlern, die hinsichtlich ausgewählter Attribute die gleichen Ausprägungen haben.[659] Diese Fehlermuster können durch zusätzliche manuelle Auswertungen bei Bedarf spezifischer charakterisiert werden. Ebenso können manuelle Auswertungen die gezielte Suche nach Prozessmängeln in der Anforderungsanalyse unterstützen.
Attribute Focusing wird auf alle zum Auswertungszeitpunkt vorliegenden klassifizierten Fehler angewandt. Diese können unter Umständen sowohl Entwicklungs- als auch Produktionsfehler sein.

---

[658] Vgl. hierzu Kapitel 5.1.3.5

[659] In Anlehnung an Dalal u. a. /Defect patterns/ 3 f.

Schritt 1: Ein- und zweidimensionale Auswertungen durchführen

Für jedes Attribut aus dem Klassifikationsschema für Entwurfs- und Implementierungsfehler (vgl. Kapitel 6.2.3) sind eindimensionale Auswertungen gemäß Kapitel 5.1.3.4 durchzuführen. Diese Attribute sind:

- Gegenstand der Änderung
- Art der Änderung
- QS-Kriterium
- Komponente
- Phase
- Auswirkung
- QS-Aktivität
- Vorgeschichte

Anschließend erfolgen für alle Attributkombinationen zweidimensionale Auswertungen gemäß Kapitel 5.1.3.4. Folgende zweidimensionalen sind demnach zu berücksichtigen:

- Gegenstand der Änderung x Art der Änderung
- Gegenstand der Änderung x QS-Kriterium
- Gegenstand der Änderung x Komponente
- Gegenstand der Änderung x Phase
- Gegenstand der Änderung x Auswirkung
- Gegenstand der Änderung x QS-Aktivität
- Gegenstand der Änderung x Vorgeschichte
- Art der Änderung x QS-Kriterium
- Art der Änderung x Komponente
- Art der Änderung x Phase
- Art der Änderung x Auswirkung
- Art der Änderung x QS-Aktivität
- Art der Änderung x Vorgeschichte
- QS-Kriterium x Komponente
- QS-Kriterium x Phase
- QS-Kriterium x Auswirkung
- QS-Kriterium x QS-Aktivität
- QS-Kriterium x Vorgeschichte
- Komponente x Phase
- Komponente x Auswirkung

- Komponente x QS-Aktivität
- Komponente x Vorgeschichte
- Phase x Auswirkung
- Phase x QS-Aktivität
- Phase x Vorgeschichte
- Auswirkung x QS-Aktivität
- Auswirkung x Vorgeschichte
- QS-Aktivität x Vorgeschichte

Schritt 2: Reihenfolge der Auswertungen ermitteln

Mit Abschluss der ein- und zweidimensionalen Auswertungen liegt jeweils die potenzielle Relevanz einer Auswertung vor. Die potenzielle Relevanz ist der absolute Betrag der Differenz zwischen beobachteter und erwarteter relativer Häufigkeit eines Attributwerts im Fall eindimensionaler Auswertungen oder eines Attributwertepaares im Fall zweidimensionaler Auswertungen (vgl. Kapitel 5.1.3.4). Mittels der potenziellen Relevanz wird die Reihenfolge der Auswertungstypen bestimmt. Hierzu sind alle ein- und zweidimensionalen Auswertungen in einer Tabelle zu vereinen und absteigend nach der potenziellen Relevanz zu sortieren. In Anlehnung an Bhandari wird auf der Grundlage der Tabelle bestimmt, welche ein- und zweidimensionalen Auswertungstypen wann erstmalig auftreten, und die ersten 20 Auswertungstypen in der Reihenfolge der Tabelle übernommen.

Zu den 20 Auswertungstypen werden dann jeweils Tabellen mit entsprechenden Auswertungen generiert, wobei die Auswertungen wiederum absteigend nach potenzieller Relevanz sortiert sind.[660] Zugleich sind maximal 7 Auswertungen pro Tabelle aufzuführen.

Schritt 3: Auffällige Zusammenhänge in den Fehlerdaten bestimmen

Um Erkenntnisse aus den Tabellen abzuleiten, die mit Attribute Focusing generiert worden sind, sind die Auswertungen wie folgt mit Schlüsselpersonen aus dem Softwareentwicklungsprojekt zu interpretieren:[661]

- Für eindimensionale Auswertungen zu einem Attribut q:
    - Für eine positive Differenz zwischen beobachteter und erwarteter relativer Häufigkeit: Welcher Attributwert von q tritt häufiger auf als erwartet? Ist dies erstrebenswert?

[660] Vgl zur potenziellen Relevanz von Auswertungen Kapitel 5.1.3.4.

[661] In Anlehnung an Bhandari /Attribute Focusing/ 282 f. und Bhandari u. a. /Improvement/ 188-190.

- Für eine negative Differenz zwischen beobachteter und erwarteter relativer Häufigkeit: Welcher Attributwert von q tritt seltener auf als erwartet? Ist dies erstrebenswert?

- Für zweidimensionale Auswertungen zu den Attributen q_1 und q_2:
  - Für eine positive Differenz zwischen beobachteter und erwarteter relativer Häufigkeit: Welcher Wert v des Attributs q_1 trifft häufiger als erwartet mit dem Wert u des Attributs q_2 auf? Ist dies erstrebenswert?
  - Für eine negative Differenz zwischen beobachteter und erwarteter relativer Häufigkeit: Welcher Wert v des Attributs q_1 trifft seltener als erwartet mit dem Wert u des Attributs q_2 auf? Ist dies erstrebenswert?

Schritt 4: Auffällige Zusammenhänge zu Fehlermustern mit potenziellem Handlungsbedarf verdichten

Als nächstes sind die gewonnenen Erkenntnisse der Tabellen soweit wie möglich zu potenziellen Fehlermustern zu verdichten.

Die Verdichtung der Erkenntnisse zu Fehlermustern kann wie folgt erfolgen:

1. Zunächst ist eine Erkenntnis aus dem dritten Schritt der Fehlerauswertung zu wählen, der einen Wert des Attributs „Gegenstand der Änderung“ beinhaltet.
2. Füge eine weitere Erkenntnis hinzu, die sich inhaltlich an die bisher ausgewählten Erkenntnisse anknüpfen lässt und zu keiner im Widerspruch steht. Wiederhole diesen Schritt so häufig wie möglich.

Beispielsweise kann Attribute Focusing folgende Auffälligkeiten in Fehlerdaten aufzeigen:

- Interne Schnittstellenfehler (Gegenstand der Änderung) werden verhältnismäßig häufig korrigiert, indem etwas eingefügt wird (Art der Änderung).
- Komponente A (Komponente) hat verhältnismäßig viele interne Schnittstellenfehler (Gegenstand der Änderung).

Eine mögliche Verdichtung zu einem potenziellen Fehlermuster lann wie folgt aussehen: Komponente A hat verhältnismäßig viele interne Schnittstellenfehler, die häufig korrigiert werden, indem etwas eingefügt wird.

Schritt 5: Konsolidierung der Fehlermuster durch manuelle Auswertungen

Durch zusätzliche manuelle Auswertungen kann nun untersucht werden, ob die verdichteten Erkenntnisse tatsächlich Fehlermuster sind. Ausgehend von den ein- und zweidimensionalen Auswertungen mit Attribute Focusing wird durch zusätzliche manuelle Auswertungen versucht, Teilmengen von Fehlern mit möglichst vielen Gemeinsamkeiten zu bestimmten. Eine

Gemeinsamkeit liegt vor, wenn alle Fehler der betrachteten Teilmenge für ein Attribut den gleichen Attributwert besitzen. Bei der Suche nach Fehlermustern ist nicht aus den Augen zu verlieren, dass mit der präzisieren Charaktisierung eines Fehlermusters einhergeht, dass die zu Grunde liegende Teilmenge an Fehlern tendenziell kleiner wird.

Schritt 6: Fehlermuster mit Handlungsbedarf bestimmen

In einer gemeinsamen Arbeitssitzung mit Schlüsselpersonen aus dem Softwareentwicklungsprojekt sind die potenziellen Fehlermuster vorzustellen. Die Schlüsselpersonen bestimmen mit ihrem Domänenwissen über den Softwareentwicklungsprozess, welche Fehlermuster Handlungsbedarf aufweisen.

6.6 Ursachen analysieren

Die Identifikation der Ursachen für ausgewählte Fehlermuster erfolgt im Rahmen einer Arbeitssitzung mit Schlüsselpersonen des Softwareentwicklungsprojekts. Diese Arbeitssitzung muss durch einen Moderator geführt werden, der neutral ist und dessen Neutralität akzeptiert ist.

Für jedes Fehlermuster mit Handlungsbedarf erfolgt die Ursachenanalyse wie folgt:

- Die Merkmale des ausgewählten Fehlermusters werden anhand von Auswertungen in Form von Abbildungen und / oder Tabellen in die Erinnerung der Teilnehmer gerufen.
- Ausgehend von der Konfiguration des ODC-Attributs Gegenstand der Änderung können Schlussfolgerungen über die Prozesse der Entwicklungsaufgaben und der Qualitätssicherung gezogen werden.[662]
- Zur Konkretisierung der Diskussion werden exemplarisch die Beschreibung weniger Fehler aus dem Fehlermuster diskutiert. Die Auswahl der Fehler erfolgt durch die Teilnehmer.
- Es wird in der Gruppe diskutiert, ob die Fehler des ausgewählten Fehlermusters in der Regel originäre Fehler oder Folgefehler sind.[663] Sofern Folgefehler vorliegen, ist zu diskutieren, welche originären Fehler zu Grunde liegen. Sofern die Folgefehler auf einen Anforderungsfehler zurückzuführen sind, ist der Typ des Anforderungsfehlers zu bestimmen.[664]

[662] Vgl. hierzu Kapitel 5.1.3.5.
[663] Vgl. hierzu Kapitel 4.1.
[664] Vgl. hierzu Kapitel 6.2.2.

- Die Teilnehmer nennen Ursachen für das Fehlermuster. Diese Ursachen werden während der Arbeitssitzung in einem Ursache-Wirkungsdiagramm nach Ishikawa[665] auf einer Pinnwand erfasst.

[665] Vgl. Ishikawa /Quality Control/ 18-29.

7 Anwendung des entwickelten Verfahrens in einer Fallstudie

7.1 Ziel der explorativen Fallstudie

Die *empirische Forschungsstrategie* bzw. kurz *Empirie* ist eine methodengestützte Vorgehensweise, bei der gedankliche Konzepte aus einer Beobachtung der Realität gewonnen und die in diesen Konzepten enthaltenen Aussagen an der Realität überprüft werden.[666] Sie wird in dieser Arbeit wie die sachlich-analytische Forschungsstrategie eingesetzt, um das erstmalige Erkennen von Zusammenhängen zu ermöglichen. Im Gegensatz zur sachlich-analytischen Forschungsstrategie erfolgt die Exploration in der empirischen Forschungsstrategie jedoch durch die Konfrontation mit der Realität.[667]

Als Forschungsmethode innerhalb der empirischen Forschungsstrategie wird die Fallstudienmethode gewählt. Die *Fallstudienmethode* untersucht ein aktuelles Phänomen innerhalb seines realen Kontextes.[668] Sie eignet sich insbesondere dann, wenn die Grenzen zwischen dem Phänomen und seinem Kontext nicht direkt einsichtig sind. In dieser Arbeit wird die Fallstudienmethode mit einer explorativen Zielsetzung eingesetzt. Der Umstand, dass eine Untersuchungseinheit eingehend in ihrem realen Kontext analysiert wird, erfordert einen längeren Aufenthalt des Forschers am Ort der Untersuchung. Dies ermöglicht aber zugleich das Erhebungsprogramm kurzfristig bei Bedarf zu ändern. Hieraus ergibt sich das explorative Potenzial der Fallstudienmethode.[669]

Ziel der explorativen Fallstudie ist die Durchführbarkeit des hergeleiteten Fehleranalyseverfahrens zu untersuchen:[670]

> *Welche Erfahrungen werden gemacht, wenn das entwickelte Verfahren praktisch eingesetzt wird?*

Diese Zielfrage kann weiter konkretisiert werden, indem sie in Teilfragen zerlegt wird:

> *Welche Erfahrungen werden bei*
>
> - *der Konfiguration des Verfahrens,*

---

[666] Vgl. zu diesem Absatz Kapitel 1.3.4.2.
[667] Vgl. hierzu Kapitel 1.3.4.2.
[668] Vgl. hierzu und zum folgenden Satz Yin /Case study research/ 13 f. und Yin /Case study crisis/ 58.
[669] Vgl. Kubicek /Organisationsforschung/ 59 f.
[670] Vgl. Kapitel 1.2.

- *der Klassifikation von Fehlern,*
- *der Auswertung klassifizierter Fehlern und*
- *der Ursachenanalyse*

gemacht?

Die *Untersuchungseinheit der Fallstudie* ist die Anwendung des Fehleranalyseverfahrens aus Kapitel 6 in einem realen Softwareentwicklungsvorhaben. Es werden reale Fehler dieser Softwareentwicklung erfasst und analysiert, um Mängel im Prozess der Anforderungsanalyse dieses Entwicklungsvorhabens aufzudecken.

7.2 Kontext der Fallstudie

Die detaillierte Beschreibung des Kontexts der Fallstudie ermöglicht die Ergebnisse der Untersuchung besser zu verstehen und adäquat einzuordnen.[671]

7.2.1 Auswahl des Untersuchungsobjekts

Im dritten und vierten Quartal des Jahres 2005 wurden 25 Unternehmen kontaktiert, in denen Anwendungssoftware entwickelt wird. Drei Unternehmen haben Interesse daran bekundet an der Fallstudie mitzuwirken und ein Softwareentwicklungsvorhaben als Untersuchungsobjekt[672] zur Verfügung zu stellen. Mit einem dieser drei Unternehmen wurden die Gespräche weiter vertieft, da inhaltliche und zeitliche Kriterien[673] für eine Zusammenarbeit zweckmäßig waren. Dieses Unternehmen sowie dessen Mitarbeiter, Produkte, Kunden und Mitarbeiter der Kunden werden im Weiteren anonymisiert.[674]

Das Fallstudien-Unternehmen, nachfolgend als *IT-Power* bezeichnet, gehört zum Unternehmensverbund einer Versicherung und ist der IT-Dienstleister des Konzerns.[675] Die Versicherung gehört zu den großen deutschen Versicherungskonzernen und wird nachfolgend kurz als

---

[671] Vgl. Dubé, Paré /Case research/ 610 f.

[672] Das Untersuchungsobjekt ist nicht zu verwechseln mit der Untersuchungseinheit. Die Untersuchungseinheit ist die Anwendung des Fehleranalyseverfahrens in einem realen Softwareentwicklungsvorhaben. Das Untersuchungsobjekt ist ein konkretes Softwareentwicklungsvorhaben, in dem das Verfahren angewandt wird. Wird eine Untersuchung wiederholt, bleibt die Untersuchungseinheit bestehen, während Subjekte und Objekte der Untersuchung ersetzt werden können. Vgl. hierzu auch Kapitel 4.3.3.2.

[673] Inhaltliche Kriterien waren z. B., dass eine Software entwickelt und nicht nur angepasst wird und dass das Softwareentwicklungsprojekt alle Teilaufgaben gemäß Kapitel 2.2.3 umfasst. Ein zeitliches Kriterium war, dass die Fallstudie spätestens im zweiten Quartal 2005 beginnen kann.

[674] Vgl. Fallstudiendokument Nr. 2 im Anhang C.1.

[675] Vgl. Interviews vom 01.06.05 und 10.06.05 (Ereignisse 1 und 3) mit dem internem Berater (IT-Power) sowie E-Mail-Antwort des Bereichsleiters Anwendungsentwicklung (IT-Power) vom 30.08.05 (Ereignis 20) in Anhang C.3.

GDV bezeichnet. GDV bietet Produkte für Versicherungsschutz, Vorsorgestrategien und Vermögensberatung an. IT-Power bietet folgende Dienstleistungen an:

- IT-Strategie- und Architektur-Beratung
- Anwendungsentwicklung
- Betrieb und Verwaltung von Hardware-Plattformen, Standardapplikationen und Netzwerken
- Konzeption und Realisierung von Daten- und Sprachnetzen
- Benutzerunterstützung für Client-/Server-Infrastrukturen

Der Kontakt zu IT-Power wurde über den *Bereichsleiter der Anwendungsentwicklung*[676] aufgebaut. Die weitere Zusammenarbeit wurde auf Seiten von IT-Power von einem *internen Berater*[677] der Stabsstelle für Qualitätsmanagement betreut. Die erste Kontaktaufnahme mit IT-Power erfolgte am 15.11.04. Nach einigen vorbereitenden Gesprächen begann die Fallstudie am 01.06.05 und endete am 20.03.06.[678] Sie dauerte folglich ca. 10 Monate.

Um die Arbeit in einem Softwareentwicklungsprojekt nur minimal durch zusätzlichen Aufwand für die Fallstudie zu beeinträchtigen, wurde beschlossen, Fehler auf der Grundlage von erfassten Informationen in einer Fehlerverfolgungssoftware nachträglich zu klassifizieren.

Zum Zeitpunkt der Anfrage liefen ca. 30 Softwareentwicklungsprojekte unterschiedlicher Größe bei IT-Power.[679] Aus diesen wurde durch den internen Berater (IT-Power) eines von zweien zur Auswahl gestellt. In den Projekten wurden verschiedene Softwareprodukte zur Fehlerverfolgung eingesetzt, in dem einen eine kommerzielle Software und in dem anderen eine Eigenentwicklung in IBM Lotus Notes. Da Fehler nachträglich klassifiziert werden sollten, waren der Umfang, die Art und die Qualität der erfassten Informationen zu den Fehlern von erheblicher Bedeutung. Um eine Auswahl unter diesen Kriterien zu bewerkstelligen, wurde jeweils eine vollständige exemplarische Beschreibung eines Fehlers aus den zwei Projekten in Form von Bildschirmfotos (Screenshots) bereitgestellt.[680] Die Entscheidung fiel auf das Projekt, in dem die eigenentwickelte Software zur Fehlerverfolgung eingesetzt wurde. Maßgeblich für diese Entscheidung waren zum einen der Umfang der Informationen, die zu einem Fehler erfasst wurden, sowie die Art der erfassten Informationen. Das Untersuchungsobjekt wird nachfolgend in Kapitel 7.2.2 ausführlich dargestellt.

---

[676] Vgl. zum Profil des Bereichsleiters für Anwendungsentwicklung Anhang C.4.
[677] Vgl. zum Profil des internen Beraters der Stabstelle Qualitätsmanagement Anhang C.4.
[678] Vgl. hierzu Anhang C.3.
[679] Vgl. E-Mail-Antwort vom Bereichsleiter vom 30.08.05 (Ereignis 20) in Anhang C.3.
[680] Vgl. Fallstudiendokument 1 in Anhang C.1.

7.2.2 Darstellung des Untersuchungsobjekts

7.2.2.1 Softwareprodukt LV-Neu

Das für die Fallstudie ausgewählte Softwareentwicklungsvorhaben hat die Wartung[681] einer Anwendungssoftware für Lebensversicherungen zum Gegenstand. Regelmäßig werden neue Releases dieser Anwendungssoftware für die produktive Nutzung freigegeben. Die Anwendungssoftware wird nachfolgend als *LV-Neu* bezeichnet.

Ziel und grundlegende Funktionalität von LV-Neu[682]

LV-Neu ist ein Verwaltungssystem für Lebensversicherungsverträge. Die Sachbearbeitung wird durch geschäftsprozessorientierte Bearbeitungen für den gesamten Lebenszyklus eines Vertrages unterstützt.

Die Dialogmasken einer Bearbeitung werden in einer standardisierten Reihenfolge durchlaufen, wobei benutzerdefinierte Sprünge möglich sind. Die einzelnen Masken haben einen vom Versicherungsprodukt abhängigen Aufbau. Die Bearbeitung kann auf jeder Maske durch Freigabe (Änderungen werden in die Datenbank geschrieben), Abbruch (eingegebene Daten gehen verloren) oder Schwebe (vorhandene Daten werden als Arbeitsversion zwischengespeichert) beendet werden.

Die Unterbrechung einer erfassungsintensiven Bearbeitung (z. B. Antragsbearbeitung) ist durch die Implementierung eines Schwebekonzeptes möglich. Der Sachbearbeiter kann Arbeitsversionen des Vertrages abspeichern, die er zu einem späteren Zeitpunkt weiterbearbeiten und freigeben kann. Je Vertrag existiert höchstens ein schwebender Vorgang, der nicht historisiert wird. Schwebende Vorgänge blockieren nicht die Gesamtbestandsbearbeitungen.

Differenzierte Plausibilitätsprüfungen stellen die Konsistenz der Daten sicher. Es gibt eine Plausibilisierung

- auf der Ebene einzelner Maskenfelder (Steuerung abhängig vom Versicherungsprodukt, Verlassen des Maskenfeldes nur bei korrektem Inhalt möglich),
- auf der Ebene einzelner Masken (Konsistenzprüfung aller Felder einer Maske, durchgeführte Prüfungen und Grenzwerte sind abhängig vom Versicherungsprodukt) und
- auf der Ebene des Gesamtvertrages (Konsistenzprüfung aller Daten vor und nach Berechnung, Freigabe nur bei konsistentem Vertragsstand möglich).

Durch die Implementierung einer Kontenführung, mit der alle Bewegungsdaten kontenmäßig geführt werden, wird die vollständige Transparenz aller Geschäftsvorfälle sichergestellt. Die

---

[681] Vgl. zum Begriff der Wartung Kapitel 2.2.4.

[682] Vgl. zum Ziel und der grundlegenden Funktionalität von LV-Neu Fallstudiendokument 6 in Anhang C.1.

Dokumentation der verschiedenen Vertragsstände im Lebenszyklus des Vertrages erfolgt unter anderem über folgende Konstrukte:

- Bearbeitungsnachweis: Information darüber, welcher Sachbearbeiter oder welches Batchprogramm, was (Art der Bearbeitung) wann (Bearbeitungstermin) womit (Versicherungsvertrag) mit welcher Gültigkeit (Wirksamkeitstermin) gemacht hat.
- Historienführung: chronologische und lückenlose Dokumentation aller Vertragsstände und der Bearbeitungen, aus denen die Vertragsstände hervorgegangen sind.

Historie und Bedeutung von LV-Neu

LV-Neu soll eine Anwendungssoftware *LV-Alt* ersetzen, mit der zuvor Lebensversicherungsverträge verwaltet wurden.[683] Monatlich werden Verträge aus LV-Alt nach LV-Neu migriert. LV-Neu bietet noch nicht alle Funktionen, die LV-Alt zur Verfügung stellt. Während der Übergangsphase müssen daher beide Softwaresysteme parallel benutzt werden. Ziel ist es LV-Alt irgendwann komplett abzulösen.

LV-Neu ersetzt zwar eine ältere Anwendungssoftware, hat aber selbst bereits Merkmale die für Altsysteme (legacy information systems) charakteristisch sind.[684] Hierzu zählen vor allem:[685]

- Sie wird seit über 10 Jahren produktiv eingesetzt.
- Sie ist für die GDV geschäftskritisch und muss stets produktiv einsatzbereit sein.
- Sie hat mit über 1,5 Millionen Codeanweisungen einen großen Umfang.[686]
- Sie hat eine zeichenbasierte Benutzerschnittstelle und wird über eine Terminalemulation genutzt, die eine Verbindung zu einem Großrechner aufbaut.

Architektur von LV-Neu[687]

Die Anwendungssoftware LV-Neu ist eine Großrechneranwendung und wird in der Programmiersprache C entwickelt. Als Datenbankmanagementsystem kommt Informix von IBM zum Einsatz.

Die Abbildung 7-1 veranschaulicht die Komponenten der Architektur von LV-Neu.[688]

---

[683] Vgl. zu diesem Absatz Telefongespräch mit Testmanager 1 (GDV) vom 26.07.05 (Ereignis 13) in Anhang C.3.

[684] Vgl. zu den Merkmalen für Altsysteme Brodie, Stonebraker /Legacy systems/ 3 f.

[685] Vgl. zum Folgenden E-Mail-Antwort vom Bereichsleiter Anwendungsentwicklung (IT-Power) vom 30.08.05 (Ereignis 20), Interview mit Leiter Qualitätsmanagement (IT-Power) vom 08.09.05 (Ereignis 24), Interview mit Produktdatenverantwortlichen (GDV) vom 02.11.05 (Ereignis 27).

[686] Vgl. hierzu Anhang C.5.

[687] Vgl. zu diesem Abschnitt Fallstudiendokumente 6 und 11 aus Anhang C.1 sowie Interviews mit Testmanager 1 (GDV) am 08.07.05 (Ereignis 8), mit Produktdatenverantwortlichen (GDV) am 02.11.05 (Ereignis 27), mit Leiter Qualitätsmanagement (IT-Power) am 08.09.05 (Ereignis 24) im Anhang C.3.

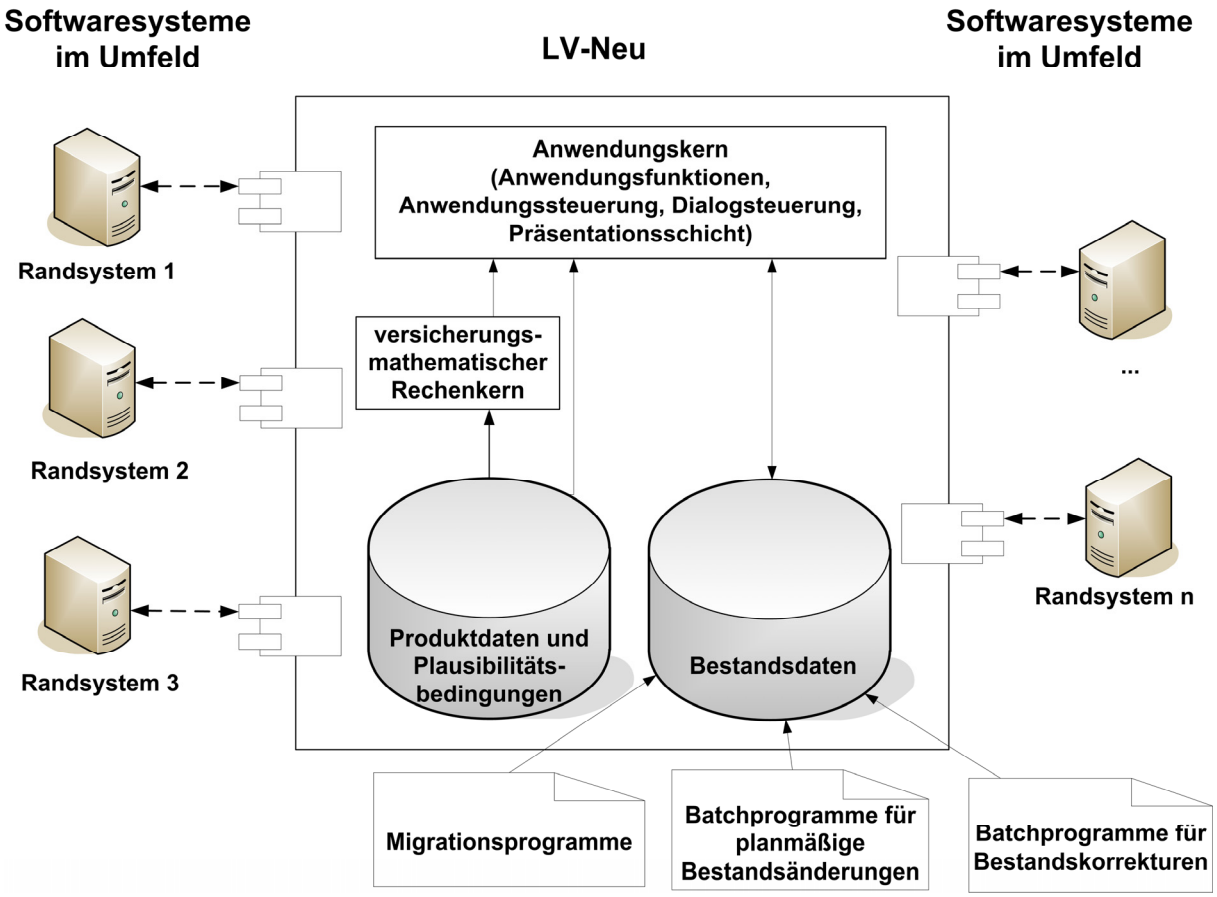


Abbildung 7-1: Architektur von LV-Neu

LV-Neu besteht aus folgenden Komponenten:

- **Anwendungskern**

  Der Anwendungskern besteht aus Anwendungsfunktionen, der Anwendungssteuerung, der Dialogsteuerung sowie der Präsentationsschicht. Die *Anwendungsfunktionen* bilden den fachlichen Kern von LV-Neu. Hierzu zählen z. B. Funktionen zur Kontoführung und zur Plausibilitätsprüfung. Letztere greift entweder auf Plausibilitätsbedingungen in den Produktdaten zurück oder auf im Anwendungskern fest verankerte Systemplausibilitätsbedingungen.

  Die *Anwendungssteuerung* steuert die Aufrufreihenfolge der Anwendungsfunktionen. Sie wird von der Dialogsteuerung beim Start einer Anwendung aus einem Menü und nach Eingabe von Daten in einer Maske aktiviert. In Batchanwendungen wird die Anwendungssteuerung von der Batchsteuerung aufgerufen.

[688] Es handelt sich um eine stark vereinfachte Darstellung der Architektur. Diese ist jedoch für die vorliegende Arbeit ausreichend und zweckmäßig.

In der *Dialogsteuerung* wird der Benutzer durch die Menüs und Masken geführt und seine Eingaben werden feld- und maskenspezifisch auf Korrektheit und Konsistenz geprüft.

Die *Präsentationsschicht* umfasst die Ausgabe der Masken, die Anzeige der Feldinhalte und die Verarbeitung der Tastatureingaben mit der Zuordnung zu Maskenfeldern. Sie ist mit der Dialogsteuerung verflochten, da Präsentationsschicht und Dialogsteuerung gegenseitig Dienste voneinander aufrufen.

- **Versicherungsmathematischer Rechenkern**

  Der versicherungsmathematische Rechenkern führt alle versicherungsmathematischen Berechnungen durch. Er ist sowohl Bestandteil von LV-Neu als auch in funktional reduzierter Form von den Außendienstsystemen der GDV.

- **Produktdaten und Plausibilitätsbedingungen**

  Produktdaten beschreiben die versicherungstechnischen Produkte, die in LV-Neu zur Verfügung stehen. Mit den Produktdaten sind Plausibilitätsbedingungen verknüpft. Es gibt zum einen Plausibilitätsbedingungen, die über die Produktdaten gesteuert werden und zum anderen solche, die im Code fest programmiert und damit integraler Bestandteil des Anwendungskerns sind. Ein Sachbearbeiter hat nur lesenden Zugriff auf die Produktdaten und Plausibilitätsbedingungen.

- **Bestandsdaten**

  In den Bestandsdaten sind alle Daten über Anträge und Verträge mit ihrer Historie, Kontenständen und Bewegungen enthalten. Die Bestandsdaten werden in einer Informix-Datenbank zentral verwaltet.

- **Schnittstellen zu Randsystemen**

  LV-Neu muss mit weiteren Softwaresystemen der GDV wie z. B. einer Massenbriefschreibung interagieren. Diese werden im Folgenden als Randsysteme im Umfeld von LV-Neu bezeichnet. Eine Komponente von LV-Neu umfasst die Schnittstellen zu diesen Randsystemen und ist für die Kommunikation und den Zugriff auf deren Datenbestände zuständig.

Neben den Randsystemen von LV-Neu existieren drei Softwaresysteme, die parallel zu LV-Neu weiterentwickelt werden. Sie sind nicht Bestandteil von LV-Neu, greifen aber direkt auf die Bestandsdaten zu. Hierzu zählen:

- Migrationsprogrammen, mit denen monatlich Verträge aus LV-Alt nach LV-Neu migriert werden.

- Batchprogramme für planmäßige Bestandsänderungen.
- Batchprogramme für außerplanmäßige Bestandskorrekturen.

7.2.2.2 Softwareentwicklungsvorhaben LV-Neu

Merkmale des Softwareentwicklungsvorhabens LV-Neu

Das Softwareentwicklungsvorhaben LV-Neu gehört hinsichtlich Aufwand und der Personalanzahl zu den größten Vorhaben, die IT-Power betreut.[689] Organisationsübergreifend wirken ca. 80 Personen daran LV-Neu weiterzuentwickeln. Darunter sind 55 bis 60 Personen auf der Seite von IT-Power beschäftigt, unter anderem 12 bis 15 Programmierer, 9 Tester, 8 Projektleiter von verschiedenen Teilprojekten (wie z. B. der Harmonisierung mit Randsystemen oder die Migration) sowie ein Fehlermanager.

Eine Besonderheit dieses Softwareentwicklungsvorhabens ist, dass die GDV nicht nur Kunde der Anwendungssoftware ist.[690] Vielmehr entwickelt sie mit dem versicherungsmathematischen Rechenkern sowie den Produktdaten und Plausibilitätsbedingungen zwei wichtige Komponenten im eigenen Hause und tritt folglich zugleich gegenüber IT-Power als Zulieferer auf. Ebenso ist GDV als Kunde nicht nur am Abnahmetest beteiligt, sondern trägt die Verantwortung für weitere zentrale Teststufen. Eine klare Trennung zwischen Auftraggeber und -nehmer liegt in diesem Softwareentwicklungsvorhaben folglich nicht vor.

Softwareentwicklungsprojekte LV-Neu 4.2 und LV-Neu 4.3

Die Wartung der Anwendungssoftware LV-Neu wird nachfolgend als Softwareentwicklungsvorhaben bezeichnet. Sofern auf die Entwicklung eines bestimmten Releases Bezug genommen werden soll, wird im Folgenden der Begriff des Softwareentwicklungsprojekts benutzt.

Da erfasste Fehler nachträglich klassifiziert werden sollten, wurde beschlossen, diese Fehlerklassifikation möglichst zeitnah durchzuführen, um bei Bedarf Projektbeteiligte fragen zu können.[691] Zeitnah zum Untersuchungszeitraum vom 01.06.05 bis 20.03.06 fand die Freigabe zweier Releases von LV-Neu statt: das Release LV-Neu 4.2 wurde am 06.12.2004 für die produktive Nutzung freigegeben und Release LV-Neu 4.3 am 02.05.05. Beide Softwareentwicklungsprojekte wurden im Rahmen der Fallstudie berücksichtigt, da sowohl während der Ent-

---

[689] Vgl. zu diesem Absatz Interviews mit internem Berater (IT-Power) vom 10.06.05 (Ereignis 3), mit Testmanager 2 (IT-Power) vom 08.09.05 (Ereignis 23), mit Leiter Qualitätsmanagement (IT-Power) vom 08.09.05 (Ereignis 24) sowie der E-Mail-Antwort des Bereichsleiter Anwendungsentwicklung (IT-Power) vom 30.08.05 (Ereignis 20) im Anhang C.3.

[690] Vgl. zu diesem Absatz Interview mit Leiter Qualitätsmanagement (IT-Power) vom 08.09.05 (Ereignis 24) in Anhang C.3.

[691] Vgl zu diesem Absatz Interview mit Testmanager 1 (GDV) vom 03.06.05 (Ereignis 2) in Anhang C.3.

wicklung als auch nach Auslieferung des jeweiligen Releases entdeckte Fehler untersucht werden konnten.

Die Entwicklung eines Releases von LV-Neu dauert in der Regel ca. 6 bis 7 Monate, so dass jährlich zwei Releases freigegeben werden.[692]

Prozesse der Entwicklungsaufgaben und der Qualitätssicherung

Die Softwareentwicklungsprojekte für LV-Neu 4.2 und 4.3 weisen die gleichen Prozesse der Entwicklungsaufgaben auf. IT-Power verfolgt in diesem Zusammenhang einen sequenziellen Entwicklungsansatz (vgl. Abbildung 7-2).[693]

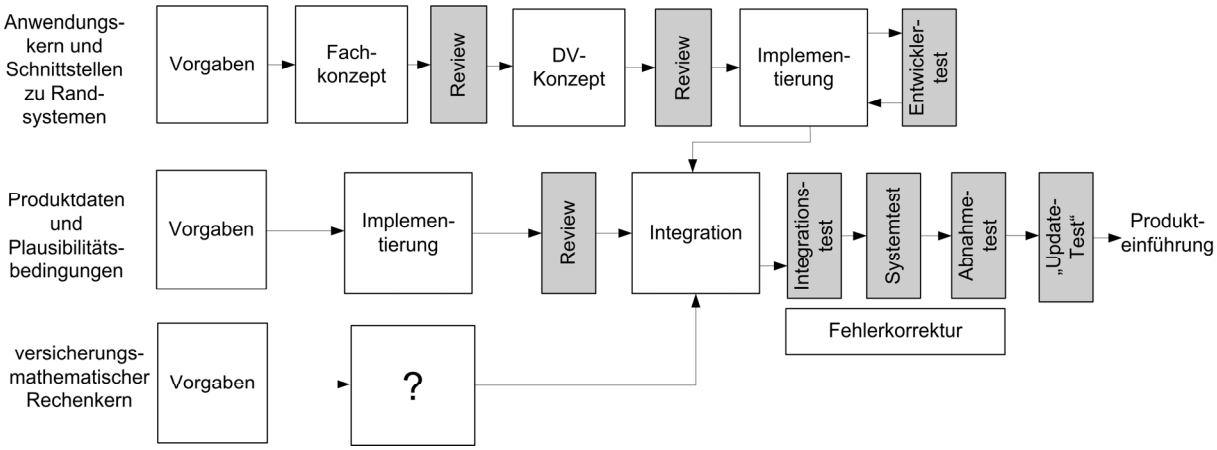


Abbildung 7-2: Sequenzieller Entwicklungsansatz für LV-Neu

Am Prozess der Entwicklungsaufgaben und der Qualitätssicherung sind drei Organisationseinheiten bei IT-Power und der GDV beteiligt.[694] Auf der Grundlage der Vorgaben der GDV erstellt IT-Power ein Fachkonzept, das sowohl fachliche als auch technische Entscheidungen enthält. Das Fachkonzept ist folglich eine Mischung aus Anforderungs- und Feinentwurfsdokument.[695] Dieses Fachkonzept wird einem Review unterzogen, an dem Entwickler und Vertreter der GDV teilnehmen.

Bei Bedarf wird anschließend das DV-Konzept angepasst. Das DV-Konzept umfasst Entscheidungen des Grobentwurfs (z. B. Softwarearchitektur) und des Feinentwurfs (z. B. Algo-

[692] Vgl. Interview mit internem Berater (IT-Power) vom 01.06.05 (Ereignis 1) in Anhang C.3.

[693] Vgl. Interview mit Testmanager 1 (GDV) vom 08.07.05 (Ereignis 8) in Anhang C.3.

[694] Diese drei Organisationseinheiten können in weitere organisatorisch unabhängige Einheiten zerlegt werden. Vgl. hierzu Fallstudiendokument 5 in Anhang C.1.

[695] Vgl. Interviews mit internem Berater (IT-Power) vom 20.07.05 (Ereignis 11) und Testmanager 2 (IT-Power) vom 08.09.05 (Ereignis 23) in Anhang C.3. Ein Beispiel für ein Fachkonzept ist das Fallstudiendokument 7 in Anhang C.1.

rithmen, Datenstrukturen, Schnittstellen zu Randsystemen).[696] Änderungen an der Softwarearchitektur treten nur selten auf. Änderungen am DV-Konzept, insbesondere wenn sie Schnittstellen zu Randsystemen betreffen, werden mittels informalen Reviews geprüft. Danach erfolgt die Implementierung, welche durch Entwicklertests begleitet wird.

Die Komponente Produktdaten und Plausibilitätsbedingungen wird durch eine organisatorische Einheit der GDV betreut.[697] Die Produktdaten und Plausibilitätsbedingungen werden in einer relationen Datenbank mit Microsoft Access verwaltet. Für den versicherungsmathematischen Rechenkern wird später automatisch C-Code generiert (nur mit Produktdaten und ohne Plausibilitätsbedingungen), während für den Anwendungskern komprimierte Binärdateien erzeugt werden. Änderungen in den Produktdaten und Plausibilitätsbedingungen werden in der organisatorischen Einheit nach dem Vier-Augen-Prinzip geprüft.

Der versicherungsmathematische Rechenkern wird ebenfalls in der GDV, jedoch durch eine andere organisatorische Einheit betreut.[698] Diese Komponente wird insbesondere, aber nicht ausschließlich, für LV-Neu entwickelt. Aus Sicht von IT-Power und des Entwicklungsvorhaben LV-Neu ist diese Komponente eine externe Software-Bibliothek, die eingebunden wird, deren Entwicklungsprozess jedoch intransparent ist.

Nach der Integration des Anwendungskerns (inklusive Schnittstellen zu Randsystemen), der Produktdaten und Plausibilitätsbedingungen sowie des versicherungsmathematischen Rechenkerns folgen die Teststufen Integrationstest, Systemtest, Abnahmetest und der Update-Test.[699] Hierbei handelt es sich um funktionale Teststufen, die abgesehen vom Integrationstest alle in der Verantwortung der GDV liegen. In diesen Teststufen kommt ein Testautomatisierungswerkzeug zum Einsatz, das zuvor definierte Testfälle automatisch ausführt und etwaige Abweichungen protokolliert. Zeitgleich zu diesen automatischen Tests testen während des Abnahme- und Update-Tests erfahrene Sachbearbeiter aus den Fachbereichen der GDV ein Release von LV-Neu. Fehler, die während des Integrations-, System- oder Abnahmetests entdeckt werden und deren Korrektur dringlich ist, werden parallel zu diesen drei Stufen behoben. Die korrigierte Fassung des Releases wird mit dem Zusatz „Update“ versehen. Im Rahmen des Update-Tests werden alle Testfälle aus dem Integrations-, System- und Abnahmetest

---

[696] Vgl. zu diesem Absatz die Interviews mit internem Berater (IT-Power) vom 20.07.05 (Ereignis 11) und mit Testmanager 1 (GDV) vom 08.07.05 (Ereignis 8) in Anhang C.3. Beispiele für das DV-Konzept sind die Fallstudiendokumente 8 bis 11.

[697] Vgl. zu diesem Absatz das Interview mit dem Produktdatenverantwortlichen (GDV) vom 02.11.05 (Ereignis 27) in Anhang C.3.

[698] Vgl. zu diesem Absatz das Interview mit Testmanager 1 (GDV) vom 19.07.05 (Ereignis 10).

[699] Vgl. zu diesem Absatz Fallstudiendokument 15 in Anhang C.1 sowie die Interviews mit Tester (GDV) am 02.09.05 (Ereignis 21) und 05.09.05 (Ereignis 22), mit Testmanager 2 (IT-Power) am 08.09.05 (Ereignis 23) und mit Testmanager 1 (GDV) am 08.08.05 (Ereignis 16) in Anhang C.3.

nochmals ausgeführt, um u. a. zu überprüfen, ob die Fehlerkorrekturen erfolgreich waren. Letztlich ist der Update-Test ein Abnahmetest im eigentlichen Sinn.

Fehlerverfolgung im Softwareentwicklungsvorhaben LV-Neu[700]

Die Fehlerverfolgung im Softwareentwicklungsvorhaben LV-Neu wird durch ein Softwarewerkzeug unterstützt, das eine Eigenentwicklung in IBM Lotus Notes ist. Dieses Softwarewerkzeug wird nachfolgend als Problemdatenbank bezeichnet. Der Autor der vorliegenden Arbeit hatte vollen Zugriff auf die Problemdatenbank und hat im Verlauf der Fallstudie jeweils einen aktualisierten Stand der Problemdatenbank am 04.05.05, 01.08.05 und am 23.09.05 erhalten.

Die Problemdatenbank verwaltet Problemberichte zur Anwendungssoftware LV-Neu sowie zu den Migrations- und Batchprogrammen.[701] Ein Problem kann ein wahrgenommenes Fehlverhalten einer Software oder eine fehlende bzw. zu ändernde Funktion der Software sein. Die Analyse eines wahrgenommenen Fehlverhaltens kann zu unterschiedlichen Ergebnissen führen. Ergebnisse können z. B. sein:

- Es liegt ein Implementierungsfehler in einem Randsystem vor, der ein Fehlverhalten von LV-Neu verursacht.
- Inkonsistente Datensätze in den Bestandsdaten verursachen ein Fehlverhalten von LV-Neu. Diese Inkonsistenz wurde durch ein fehlerhaftes Batchprogramm eingeführt.
- Fehlerhafte Testdaten führen zu einem Fehlverhalten von LV-Neu.
- Ein Implementierungsfehler in LV-Neu liegt vor.
- Das beobachte Ist-Verhalten der Software ist korrekt. Es liegt kein Fehlverhalten vor.

Es werden Problemberichte zu jedem Fehlverhalten der Software erfasst, sobald GDV als Kunde Kontakt zur Software bekommt. Das heißt insbesondere, dass alle wahrgenommenen Abweichungen der Software in den von der GDV betreuten Teststufen Systemtest, Abnahmetest und dem Update-Test festgehalten werden. Im Integrationstest festgestellte Abweichungen werden nur dann erfasst, sofern sie nicht bis zum Beginn der nachfolgenden Teststufe, dem Systemtest, geklärt werden können. Neben potenziellen Entwicklungsfehlern werden alle potenziellen Produktionsfehler, d. h. alle nach Auslieferung der Software wahrgenommenen Abweichungen dokumentiert.

---

[700] Vgl. zu diesem Abschnitt die E-Mail-Antworten vom Testmanager 1 (GDV) am 14.06.05 (Ereignis 5), das Interview mit Testmanager 1 (GDV) am 19.07.05 (Ereignis 10) in Anhang C.3 sowie das Fallstudiendokument 13 in Anhang C.1.

[701] Vgl. zu den Migrations- und Batchprogrammen Kapitel 7.2.2.1.

Neben einem wahrgenommenen Fehlverhalten einer Software kann ein Problembericht ebenso eine fehlende bzw. zu ändernde Funktion der Software beschreiben. In diesem Fall ist ein Problembericht als Änderungsantrag zu verstehen, der eine gewünschte nachträgliche Änderung eines abgenommenen Fachkonzepts für ein Release beschreibt.
Die Problemdatenbank wird im Softwareentwicklungsvorhaben für folgende Zwecke genutzt:[702]

- Dokumentation von wahrgenommenen Problemen der Software und schnelle Lokalisierung etwaiger Implementierungsfehler.
- Aufwandsschätzung und Planung des fachlichen Umfangs nachfolgender Releases.
- Verfolgung des Fortschritts der Fehlerkorrektur.
- Organisationsübergreifende Kommunikationsplattform zur Klärung wahrgenommener Probleme mit der Software.
- Dokumentation getroffener Entscheidungen im Kontext von festgestellten Problemen mit der Software.

7.3 Forschungsmethodik im Rahmen der Fallstudie

Allgemeine Empfehlungen zur Fallstudienmethode im positivistischen Kontext finden sich insbesondere in den Arbeiten von Yin[703], Dubé und Paré[704] sowie Benbasat, Goldstein und Mead[705]. Auf die explorative Anwendung der Fallstudienmethode im Besonderen, ebenfalls im positivistischen Kontext, geht eine Arbeit von Paré[706] ein. Die vorliegende Arbeit orientiert sich an diesen Empfehlungen.

7.3.1 Datenerhebung

In der Fallstudie wurden semistrukturierte Interviews, Dokumentenanalysen und Fragebogenerhebung zur Datenerhebung eingesetzt.

Semistrukturierte Interviews

In der Fallstudie wurden Daten von Schlüsselpersonen mittels Interviews erhoben. Diese Interviews hatten drei Ziele:[707]

---

[702] Vgl. zum Folgenden Fallstudiendokumente 3, 23 und 27 in Anhang C.1 sowie die E-Mail-Antwort von Testmanager 1 (GDV) am 14.06.05 (Ereignis 5) in Anhang C.3.

[703] Vgl. Yin /Case study research/.

[704] Vgl. Dubé, Paré /Case research/.

[705] Vgl. Benbasat, Goldstein, Mead /Case research strategy/.

[706] Vgl. Paré /Case study research/.

[707] Vgl. zum Folgenden Seaman /Methods/ 562 f.

1. Abruf historischer Informationen zu den Softwareentwicklungsprojekten LV-Neu 4.2 und 4.3 aus dem Erinnerungsvermögen der Interviewpartner.
2. Sammeln von Meinungen und Einschätzungen zu ausgewählten Sachverhalten des Softwareentwicklungsvorhabens LV-Neu und des Fehleranalyseverfahrens.
3. Identifikation von für die Fallstudie relevanten Begriffen und ihrer Bedeutung im Kontext des Softwareentwicklungsvorhabens LV-Neu.

Vor Beginn der Interviews wurde jeweils ein Interview-Leitfaden mit offenen Fragen vorbereitet.[708] Sofern sich in einem laufenden Interview neue relevante Erkenntnisse ergaben, wurden diese durch zusätzliche Fragen berücksichtigt. Insofern wurden nicht nur erwartete, sondern insbesondere auch unerwartete Informationen erfasst.

Im Rahmen der Fallstudie wurden sowohl persönliche Interviews als auch Telefoninterviews eingesetzt.

Zu allen Interviews wurden Gesprächsprotokolle erstellt und den Interviewteilnehmern zeitnah zur Verfügung gestellt. Bei einigen persönlichen Interviews stand eine studentische Hilfskraft zur Verfügung, die während des Interviews ein vorläufiges handschriftliches Gesprächsprotokoll erstellt hat, welches im Anschluss mit dem Autor der vorliegenden Arbeit konsolidiert wurde. Sofern keine studentische Hilfskraft zur Verfügung stand wurde direkt im Anschluss an das Interview das Gesprächsprotokoll auf der Grundlage persönlicher Notizen niedergeschrieben. Die durchgeführten Interviews werden im Anhang C.3 aufgelistet. Profile zu den befragten Schlüsselpersonen sind im Anhang C.4 aufgeführt.

Alle Interviews wurden vom Autor der vorliegenden Arbeit geleitet.

Dokumentenanalyse

Zur Konsolidierung und Vertiefung der durch die Interviews gewonnenen Erkenntnisse wurden gezielt Dokumente durch den Autor angefordert und durch IT-Power zur Verfügung gestellt. Diese sind im Anhang C.1 aufgelistet.[709] Das wichtigste Fallstudiendokument ist die Problemdatenbank.

Fragebogenerhebung

Ebenfalls wurde zur Konsolidierung und Vertiefung von Erkenntnissen aus den Interviews bei Bedarf ausgewählten Schlüsselpersonen jeweils ein Fragebogen mit offenen Fragen per E-Mail gesandt. Die Antworten per E-Mail sind ebenfalls im Anhang C.3 aufgelistet.

---

[708] Vgl. zu diesem Absatz Seaman /Methods/ 562.

[709] Vgl. zur Dokumentenanalyse Yin /Case study research/ 85-88.

Maßnahmen zur Steigerung der Qualität der erhobenen Daten

Zur Steigerung der Qualität der in der Fallstudie erhobenen Daten wurden folgende Schritte unternommen: [710]

- In der Fallstudie wurden mit semistrukturierten Interviews, Dokumentenanalysen und Fragebogenerhebung drei unterschiedliche Methoden zur Datenerhebung eingesetzt.
- Alle Interviewprotokolle wurden den Interviewteilnehmern zur Verfügung gestellt, um bei Bedarf Missverständnisse zu klären.
- Neben Interviewprotokollen wurden direkt zeitnah nach den Interviews weitere Informationen festgehalten (so genannte field notes oder field memos).[711] Diese Informationen bezogen sich entweder auf Beobachtungen des Autors während eines Interviews (z. B. bzgl. einer non-verbaler Kommunikation von Interviewteilnehmern) oder auf Sachverhalte, die sich in beiläufigen Gesprächen mit den Interviewteilnehmern nach Abschluss der Interviews ergeben haben.
- Neben qualitativen Daten werden mit der Klassifikation von Implementierungsfehlern auch auf quantitative Daten in der Fallstudie zurückgegriffen.
- Im Rahmen der Fallstudie gewonnene Erkenntnisse beruhen weitgehend mindestens auf zwei unterschiedlichen Quellen.[712] Dies kann sich darin äußern, dass eine Aussage von zwei befragten Personen unabhängig voneinander bestätigt wird oder die Aussage einer Person durch die Projektdokumentation gestützt wird.
- Es wurde eine Fallstudiendatenbank[713] angelegt, welche unter anderem folgende Elemente umfasst:
    - alle Interviewleitfäden,
    - die Gesprächsprotokolle aller Interviews (siehe Anhang C.3),
    - alle Antworten auf die E-Mail-Fragebögen (siehe Anhang C.3),
    - alle von IT-Power zur Verfügung gestellen Dokumente (siehe Anhang C.1),
    - alle durch den Autor erstellten Arbeitsergebnisse (z. B. klassifizierte Fehler, Folien etc.) (siehe Anhang C.2) und
    - eingesetzte Softwareprodukte (insbesondere IBM Lotus Notes Client für die Problemdatenbank und Softwarewerkzeuge zur Messung des Codeumfangs).

[710] Vgl. zu Folgendem Dubé, Paré /Case research/ 612-615.

[711] Vgl. Dubé, Paré /Case research/ 616 und Seaman /Methods/ 567.

[712] Diese Vorgehensweise wird als Datentriangulation bezeichnet. Vgl. Dubé, Paré /Case research/ 615.

[713] Vgl. Dubé, Paré /Case research/ 616.

7.3.2 Datenanalyse

Constant Comparison Method

Die Datenerhebung wurde auf der Grundlage der für die Fallstudie formulierten Ziele in Kapitel 7.1 gesteuert. Um die Vielzahl der erhobenen Daten systematisch zu analysieren wurde eine Vorgehensweise in Anlehnung an die Methode Constant Comparison Method[714] angewandt. Hierbei wurden zunächst alle Einzelaussagen aus den Interviews und den E-Mail-Antworten extrahiert. Allen Einzelaussagen wurden Kategorien und Unterkategorien zugeordnet, die aus den Zielen und dem Kontext der Fallstudie abgeleitet worden sind. Die Tabelle 7-1 listet examplarisch einige Kategorien mit Unterkategorien auf.

Kategorie	Unterkategorie
Prozesse der Qualitätssicherung	Review Fachkonzept, Review DV-Konzept, Review Produktdaten und Plausibilitätsbedingungen, Entwicklertest, Integrationstest, Systemtest, Abnahmetest, Update-Test
Prozesse der Entwicklungsaufgaben	Anforderungsanalyse, Grobentwurf, Feinentwurf, Implementierung
Anwendungssoftware LV-Neu	Release 4.2, Release 4.3
Fehlerverfolgung	Problemdatenbank, Prozess der Fehlerverfolgung
Komponenten von LV-Neu	Anwendungskern, Schnittstellen zu Randsystemen, Produktdaten und Plausibilitätsbedingungen, versicherungsmathematischer Rechenkern
Schritt des Fehleranalyseverfahrens	Konfiguration, Fehlerklassifikation, Fehlerauswertung, Ursachenanalyse

Tabelle 7-1: Beispiele für in der Datenauswertung eingesetzte Kategorien und Unterkategorien

Die Kategorisierung der Einzelaussagen ermöglicht es, Einzelaussagen nach Kategorien oder Unterkategorien gruppieren. Aus den entstehenden inhaltlichen Mustern wurden Thesen geschlussfolgert.

Abnahme aus der Datenanalyse gewonnener Erkenntnisse

Für die Fallstudie wichtige Erkenntnisse aus der Datenanalyse wurden durch den Autor dokumentiert und zur Abnahme durch Schlüsselpersonen der Fallstudie vorgelegt.[715] Diese Dokumente sind in Anhang C.2 aufgeführt.

[714] Vgl. Seaman /Methods/ 566 f.

[715] Vgl. Dubé, Paré /Case research/ 620.

Flexibler Prozess der Datenerhebung und –analyse

Die Datenerhebung und –analyse waren in der Fallstudie keine sequenziellen Schritte.[716] Vielmehr wurden bei Bedarf aus Erkenntnissen der Datenanalyse zielgerichtet zusätzliche Daten erhoben. Diese Vorgehensweise nutzt das explorative Potenzial der Fallstudienmethode.

7.4 Ergebnisse der Fallstudie

7.4.1 Konfiguration des Verfahrens

7.4.1.1 Ablauf der Konfiguration

Die für die Konfiguration erforderlichen Informationen wurden in mehreren Schritten mit allen in Kapitel 7.3.1 dargestellten Methoden (Interview, Fragebogen, Dokumentenanalyse) erhoben.

Für die Konfiguration der Attributs QS-Aktivität mussten zunächst alle QS-Prozesse in den Softwareentwicklungsprojekten LV-Neu 4.2 und 4.3 bestimmt werden. In beiden Projekten wurden die gleichen QS-Prozesse durchgeführt (vgl. Abbildung 7-3):[717]

- Review des Fachkonzepts
- Review des DV-Konzepts
- Entwicklertest
- Review Produktdaten und Plausibilitätsbedingungen
- Integrationstest
- Systemtest
- Abnahmetest
- Update-Test

[716] Vgl. zu diesem Absatz Dubé, Paré /Case research/ 618.

[717] Vgl. Fallstudiendokument 15 und 26 in Anhang C.1, die Interviews mit Testmanager 1 (GDV) vom 08.07.05 (Ereignis 8) und 19.07.05 (Ereignis 10) sowie mit Testmanager 2 (IT-Power) vom 08.09.05 (Ereignis 23) und seiner E-Mail-Antwort vom 14.09.05 (Ereignis 25) in Anhang C.3. Siehe hierzu auch die Erläuterungen in Kapitel 7.2.2.2.

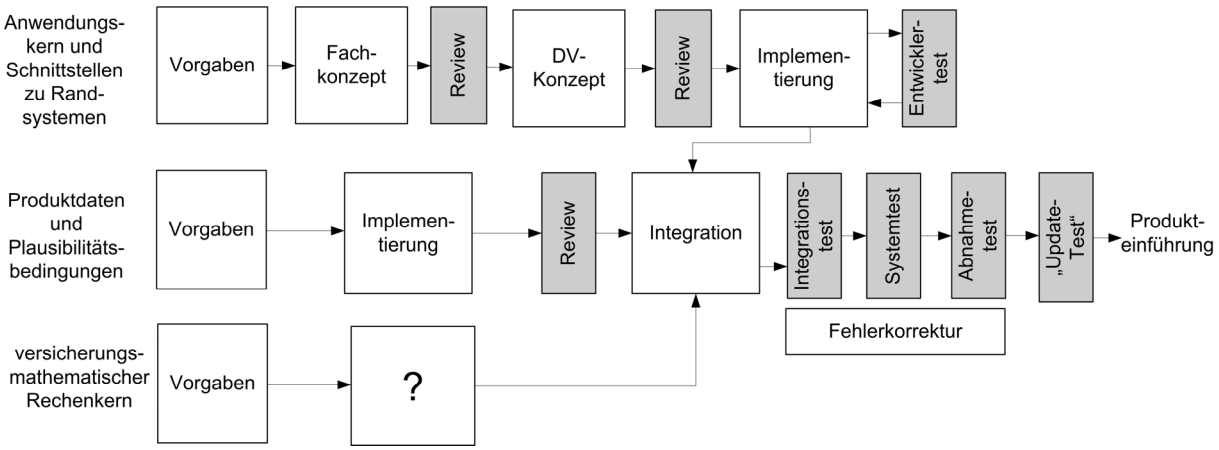


Abbildung 7-3: Sequenzieller Entwicklungsansatz für LV-Neu

Sofern Fehler im Rahmen von Reviews entdeckt werden, werden sie nicht systematisch in einem Softwarewerkzeug erfasst. Die Ergebnisprotokolle der Review-Sitzungen ermöglichen aufgrund fehlender Informationen keine nachträgliche Klassifikation von Fehlern.[718] Dies bedeutet insbesondere, dass keine Anforderungsfehler im Rahmen dieser Fallstudie klassifiziert werden können. Zudem werden erst Problemberichte zu allen Abweichungen erfasst, sobald die GDV als Kunde Kontakt zur Software bekommt. Folglich werden nur alle in den Teststufen Systemtest, Abnahmetest und dem Update-Test entdeckten Implementierungsfehler dokumentiert, da diese von der GDV verantwortet werden. Im Integrationstest festgestellte Abweichungen und damit potenzielle Fehler werden nur erfasst, sofern sie nicht bis zum Beginn des Systemtests geklärt werden können. Alle übrigen werden ebenfalls nicht dokumentiert. Damit ist der Wertebereich des Attributs QS-Aktivität definiert: Integrationstest, Systemtest, Abnahmetest und Update-Test.

Da die untersuchten QS-Prozesse von der GDV-Seite betreut wurden, erfolgte die Verknüpfung der QS-Prozesse mit den QS-Kriterien maßgeblich mit dem Testmanager 1 (GDV) und dem Tester (GDV). Es stellte sich heraus, dass wenige QS-Kriterien zum Einsatz kommen. Die Wahl der QS-Kriterien wird maßgeblich durch die Rolle (Tester, erfahrener Benutzer) bestimmt, die eine Person wahrnahm. Um die Teststufen trotzdem besser voneinander abgrenzen zu können, wurden zusätzliche Informationen zu den Teststufen erfasst (Anzahl Testfälle, manuelle oder automatische Tests, Testobjekte).[719]

[718] Vgl. z. B. Fallstudiendokumente 19-22 in Anhang C.1.
[719] Vgl. Tabelle 7-3.

Für die Konfiguration des Attributs Phase wurden Projektpläne zu LV-Neu 4.2 und 4.3 ausgewertet,[720] um für ein Release den Zeitraum des Softwareentwicklungsprojekts vom Zeitraum der produktiven Nutzungsphase abzugrenzen.

Das Attribut „Gegenstand der Änderung“ wurde *nach* der Fehlerauswertung, jedoch vor der Präsentation der Ergebnisse der Fehlerauswertung gemeinsam mit Schlüsselpersonen konfiguriert.[721] Die Fehlerauswertung verdeutlichte, dass aufgrund ihrer Häufigkeit folgende Werte dieses Attributes im weiteren Verlauf relevant sein werden: Zuweisung/Initialisierung, Bedingung und Algorithmus. Deshalb wurden bei der Konfiguration des Attributs nur diese ausgewählten Werte berücksichtigt. Das heißt, die Verknüpfung mit Prozessen der Entwicklungsaufgaben und der Qualitätssicherung beschränkte sich auf diese Werte.
Bei der Konfiguration des Attributs „Gegenstand der Änderung“ erwies es sich als zweckmäßig, nach den Komponenten Anwendungskern und Schnittstellen zum einen und der Komponente Produktdaten und Plausibilitätsbedingungen zum anderen zu unterscheiden. Dies begründete sich dadurch, dass unterschiedliche organisatorische Einheiten für die Entwicklung verantwortlich waren und die Prozesse der Entwicklungsaufgaben und der Qualitätssicherung bis zum Integrationstest unterschiedlich waren (vgl. Abbildung 7-3).

Der Wertebereich für das Attribut Komponente konkretisierte sich im Verlauf der Fehlererfassung. Nähere Erläuterungen diesbezüglich finden sich deshalb in Kapitel 7.4.2.1. Für den Fall, dass eine eindeutige Zuordnung zu einer Komponente nicht möglich war, wurde der Attributwert „unbekannt“ vorgesehen.

7.4.1.2 Ergebnis der Konfiguration

Die Konfiguration des Fehleranalyseverfahrens wurde in Kapitel 6.3 beschrieben. Nachfolgend wird das Ergebnis der Konfiguration in der Fallstudie dargestellt. Da zwei Softwareentwicklungsprojekte untersucht werden, wird zwischen LV-Neu 4.2 und 4.3 unterschieden, sofern Unterschiede in der Konfiguration vorliegen. Anderfalls gelten die Ausführungen für beide Softwareentwicklungsprojekte.

---

[720] Vgl. Fallstudiendokumente 12, 14, 16, 24 und 25 in Anhang C.1.

[721] Vgl. Arbeitssitzung mit Testmanager 1 (GDV), Testmanager 2 (IT-Power), Fehlermanager (IT-Power), interner Berater (IT-Power), Student (Lehrstuhl) und Autor (Lehrstuhl) vom 23.11.05 (Ereignis 29) in Anhang C.3.

Konfiguration des Attributs QS-Aktivität

Für die beiden Softwareentwicklungsprojekte LV-Neu 4.2 und 4.3 hat das Attribut QS-Aktivität den in Tabelle 7-2 dargestellten Wertebereich.

Attribut	Bedeutung des Attributwertes	Attributwerte
QS-Aktivität	In welcher Prüf- oder Test-Aktivität wurde der Fehler gefunden?	▪ Integrationstest ▪ Systemtest ▪ Abnahmetest ▪ Update-Test

Tabelle 7-2: Konfiguration des Attributs QS-Aktivität

Verknüpfung der Werte der Attribute QS-Aktivität und QS-Kriterium

Wie in Kapitel 7.4.1.1 ausgeführt, werden in den QS-Prozessen weitgehend die gleichen QS-Kriterien eingesetzt (vgl. Tabelle 7-3). Um trotzden die QS-Prozesse voneinander zu unterscheiden, wurden zusätzliche Information erhoben. Neben der Verküpfung mit QS-Kriterien wurde mit dem Zusätz primär oder sekundär festgehalten, in welchem Umfang ein QS-Kriterium in einem QS-Prozess angewandt wurde.

	QS-Aktivität			
	Integrationstest	Systemtest	Abnahmetest	Update-Test
QS-Kriterien	▪ Abdeckung von einzelnen Funktionen (primär). ▪ Sequenz von mehreren Funktionen (sekundär).	▪ Abdeckung von einzelnen Funktionen (primär). ▪ Sequenz von mehreren Funktionen (sekundär).	▪ Abdeckung von einzelnen Funktionen (primär). ▪ Variation von einzelnen Funktionen (primär). ▪ Sequenz von mehreren Funktionen (sekundär).	▪ Abdeckung von einzelnen Funktionen (primär). ▪ Variation von einzelnen Funktionen (primär). ▪ Sequenz von mehreren Funktionen (sekundär).
Zusätzliche Informationen	▪ mit ca. 300 (meist neuen) Testfällen werden neue Funktionen und neue Versicherungsprodukte getestet, die Aufschluss über die korrekte Integration der Komponenten geben können. ▪ Ausschließlich automatisierter Funktionstest.	▪ 7000 Testfälle (einschließlich der 300 Testfälle aus dem Integrationstest). ▪ Deutlich mehr Testfälle als im Integrationstest, von denen 50% neu sind. ▪ Ausweitung auf mehr Funktionen. ▪ Alte und neue Funktionen werden getestet. ▪ Ausschließlich automatischer Funktionstest.	▪ automatischer Funktionstest mit 3000 Testfällen: Schwerpunkt sind Funktionen, die häufig genutzt werden. ▪ Parallel zum automatischen Funktionstest führen Fachbereiche mit erfahrenen Benutzern manuelle Tests durch, die einzelne Funktionen gemäß ihrem Aufgabenschwerpunkt testen.	▪ automatischer Funktionstest: alle Testfälle aus dem Integrations-, System- und Abnahmetest (10000 Testfälle) werden nochmals durchgeführt. ▪ Fachbereichstest: manuelle Tests zuvor fehlerhafter und korrigierter Funktionen durch erfahrene Benutzer.

Tabelle 7-3: Verknüpfung der Werte der Attribute QS-Aktivität und QS-Kriterium

Konfiguration des Attributs Phase

Auf der Grundlage der Projektpläne zu LV-Neu 4.2 und 4.3 ergaben sich für das Attribut Phase jeweils unterschiedliche Wertebereiche für die zwei Softwareentwicklungsprojekte (vgl. Tabelle 7-4 und Tabelle 7-5).

Attribut	Bedeutung des Attributwertes	Attributwerte
Phase	In welcher Phase wurde der Fehler gefunden?	▪ Entwicklung (Zeitraum: 18.10.2004 – 06.12.2004) ▪ Produktion (Zeitraum: 06.12.2004 – 02.05.2005)

Tabelle 7-4: Konfiguration des Attributs Phase für das Softwareentwicklungsprojekt LV-Neu 4.2

Attribut	Bedeutung des Attributwertes	Attributwerte
Phase	In welcher Phase wurde der Fehler gefunden?	▪ Entwicklung (Zeitaum: 06.12.2004 - 02.05.2005) ▪ Produktion (Zeitraum: 02.05.2005 – 19.09.2005)

Tabelle 7-5: Konfiguration des Attributs Phase für das Softwareentwicklungsprojekt LV-Neu 4.3

Konfiguration des Attributs „Gegenstand der Änderung“

Die Tabelle 7-6 bis Tabelle 7-9 beinhalten die Verknüpfung ausgewählter Werte des Attributs „Gegenstand der Änderung“ mit Prozessen der Entwicklungsaufgaben und der Qualitätssicherung.[722] Dabei wird zwischen den Komponenten Anwendungskern und Schnittstellen (AK+SNT) zum einen und der Komponente Produktdaten und Plausibilitätsbedingungen (PDP) zum anderen unterschieden.

	Gegenstand der Änderung		
Teilprozesse	Zuweisung / Initialisierung	Bedingung	Algorithmus
Vorgaben		x	
Fachkonzept			
Implementierung (PDP)	x	x	
Integration			

Tabelle 7-6: Verknüpfungen zwischen Fehlertypen und Teilprozessen (Produktdaten und Plausibilitätsbedingungen)

[722] Vgl. hierzu die Ausführungen in Kapitel 7.4.1.1

	Gegenstand der Änderung		
QS-Prozesse	Zuweisung / Initialisierung	Bedingung	Algorithmus
Review PDP	x	x	
Integrationstest			
Systemtest			
Abnahmetest			
Update-Test			

Tabelle 7-7: Verknüpfungen zwischen Fehlertypen und QS-Prozessen (Produktdaten und Plausibilitätsbedingungen)

	Gegenstand der Änderung		
Teilprozesse	Zuweisung / Initialisierung	Bedingung	Algorithmus
Vorgaben			
Fachkonzept		x	x
Implementierung (AK+SNT)	x	x	x
Integration			

Tabelle 7-8: Verknüpfungen zwischen Fehlertypen und Teilprozessen (Anwendungskern + Schnittstellen)

	Gegenstand der Änderung		
QS-Prozesse	Zuweisung / Initialisierung	Bedingung	Algorithmus
Review Fachkonzept		x	
Review DV-Konzept			
Entwicklertest	x	x	x
Integrationstest			x
Systemtest			
Abnahmetest			
Update-Test			

Tabelle 7-9: Verknüpfungen zwischen Fehlertypen und QS-Prozessen (Anwendungskern + Schnittstellen)

Konfiguration des Attributs Komponente

Der Wertebereich des Attributs Komponente umfasst die in der Tabelle 7-10 aufgeführten fünf Werte.

Attribut	Bedeutung des Attributwertes	Attributwerte
Komponente	Welche Komponente des Arbeitsergebnisses enthielt den Fehler?	▪ Anwendungskern ▪ versicherungsmathematischer Rechenkern ▪ Schnittstellen zu Randsystemen ▪ Produktdaten und Plausibilitätsbedingungen ▪ unbekannt

Tabelle 7-10: Konfiguration des Attributs Komponente

7.4.1.3 Beobachtungen im Rahmen der Konfiguration

Die Konfiguration des Fehleranalyseverfahrens konnte ohne inhaltliche Probleme durchgeführt werden. Nachfolgend werden wesentliche Beobachtungen aufgeführt, die während der Konfiguration gemacht wurden.

Beobachtung 1:

> *Ein einheitliches Begriffsverständnis ist kritisch für eine erfolgreiche Anwendung des Fehleranalyseverfahrens.*

Im Rahmen der Konfiguration des Fehleranalyseverfahrens stellte sich heraus, wie wichtig die präzise Definition von Begriffen und ein einheitliches Verständnis der Beteiligten über diese Begriffe sind. Die Beteiligten des Softwareentwicklungsvorhabens verwandten sowohl bedeutungsgleiche oder zumindest bedeutungsähnliche Wörter (so genannte Synonyme) als auch ein Wort für verschiedene Begriffe (so genannte Homonyme).
Beispielsweise wurde ein Anforderungsdokument bisweilen auch mit folgenden Wörtern bezeichnet: Fachkonzept, Pflichtenheft oder Kundenvorgaben. Für die Dokumentation des Grobentwurfs kursierten Wörter wie DV-Konzept, Fachkonzept oder Systemarchitektur. Das Wort „Fachkonzept“ trat folglich zugleich als Synonym und als Homonym auf. Unter der Entwicklung verstanden manche Beteiligte nur die Implementierung, während andere darunter ein gesamtes Softwareentwicklungsprojekt zusammenfassten.
Neben einem uneinheitlichen Begriffssystem der Beteiligten bedeutete die Einführung eines neuen Verfahrens wie dem Fehleranalyseverfahren auch, dass die Beteiligten mit einem weiteren Begriffssystem konfrontiert wurden. So kristallisierte sich beispielsweise schnell heraus, dass der Begriff der Qualitätssicherung im Softwareentwicklungsvorhaben synonym für eine Prüfaktivität und nicht im Sinne der vorliegenden Arbeit als Oberbegriff für das Prüfen und Testen verstanden wurde.

Beobachtung 2:

> *Werden verschiedene Software-Komponenten nach unterschiedlichen Softwareentwicklungsprozessen entwickelt, muss die Konfiguration des Fehleranalyseverfahrens diesen Umstand berücksichtigen.*

In der betrachteten Fallstudie sind drei Organisationseinheiten bei IT-Power und der GDV am Softwareentwicklungsvorhaben beteiligt.[723] Jede Organisationseinheit entwickelt eine der fol-

[723] Diese drei Organisationseinheiten können in weitere organisatorisch unabhängige Einheiten zerlegt werden. Vgl. hierzu Fallstudiendokument 5 in Anhang C.1.

genden drei zentralen Komponenten: den Anwendungskern, den versicherungsmathematischen Rechenkern oder die Produktdaten und Plausibilitätsbedingungen. Sie verfolgen bis zur Integration der Softwarekomponenten dabei zwar abgestimmte, jedoch trotzdem unterschiedliche Prozesse für die Entwicklungsaufgaben und die Qualitätssicherung. Folglich musste bei der Konfiguration des Attributs „Gegenstand der Änderung“ nach Komponten unterschieden werden.[724]

Beobachtung 3:

Eine angemessene Einarbeitung in das Klassifikationsschema ist erforderlich.

Da das Klassifikationsschema im Wesentlichen eine Auflistung von Attributen und dazugehörigen Attributwerten ist, hat es bei den Teilnehmern der Fallstudie bisweilen den Eindruck erweckt, dass es vollständig verstanden werden kann, sobald es einmal grob überblickt wurde. Verschiedene anfängliche Verständnisprobleme haben jedoch gezeigt, dass eine angemessene Einarbeitung beziehungsweise Schulung von Mitarbeitern dringend erforderlich ist. So wurde beispielsweise bei den QS-Kriterien für funktionale Tests häufig übersehen, das Fehlern erst dann die Kriterien Funktionensequenz und Funktioneninteraktionen zugeordnet werden dürfen, wenn zuvor die relevanten Funktionen einzeln mit den Kriterien Funktionsabdeckung oder Funktionsvariation erfolgreich getestet worden sind (vgl. Tabelle 6-7 und Tabelle 6-8). Diese Beobachtung bestätigt Erkenntnisse des Laborexperiments von Henningsson und Wohlin zu ODC.[725]

Erfahrung 4:

Zur Charakterisierung von QS-Aktivitäten sind neben zugeordneten QS-Kriterien zusätzliche Informationen zweckmäßig.

Die Zuordnung von QS-Kriterien zu den projektspezifischen QS-Aktivitäten half dabei, die QS-Aktivitäten inhaltlich zu charakterisieren. In der vorliegenden Fallstudie wurde jedoch nur eine geringe Anzahl von QS-Kriterien eingesetzt, so dass eine inhaltliche Abgrenzung der QS-Aktivitäten allein auf der Grundlage der QS-Kriterien nicht möglich war. Beispielsweise wurden im Integrations- und Systemtest die identischen Testkriterien angewandt. Deshalb wurden zu folgenden Fragen zusätzliche Informationen zu den QS-Aktivitäten erhoben (vgl. Tabelle 7-3):

---

[724] Vgl. Kapitel 7.4.1.2.

[725] Vgl. Henningsson, Wohlin /Fault classification agreement/ 100-104. Siehe hierzu auch Kapitel 5.2.3.3.

- In welchem Ausmaß (primär/sekundär) wurde ein QS-Kriterium in einer QS-Aktivität eingesetzt?
- Wie viele Testfälle wurden im Rahmen von automatisierten Funktionstests ausgeführt?
- Was wurde schwerpunktsmäßig getestet?

Die zusätzlichen Informationen ermöglichten es, die QS-Aktivitäten besser zu verstehen und Unterschiede zu erkennen.

7.4.2 Fehler klassifizieren

7.4.2.1 Ablauf der Fehlerklassifikation

Die Fehler wurden in dieser Fallstudie auf der Grundlagen einer Problemdatenbank nachträglich klassifiziert. Zunächst wird in Kapitel 7.4.2.1.1 diese Problemdatenbank dargestellt. Anschließend wird in Kapitel 7.4.2.1.2 der Ablauf der nachträglichen Fehlerklassifikation beschrieben.

7.4.2.1.1 Darstellung der Problemdatenbank

Die Problemdatenbank verwaltet Problemberichte zur Anwendungssoftware LV-Neu sowie zu den Migrations- und Batchprogrammen.[726] Ein Problembericht dokumentiert entweder eine wahrgenommene Abweichung eines Ist-Verhaltens von einem erwarteten Soll-Verhalten der berücksichtigten Software oder einen Änderungsantrag.
Ein Problembericht besteht aus folgenden inhaltlichen Bereichen:

- Identifikation des Problems (vgl. Tabelle 7-11)
- Status der Problembearbeitung (vgl. Tabelle 7-12)
- Problembeschreibung (vgl. Tabelle 7-13)
- Problemanalyse (vgl. Tabelle 7-14)
- Problembehebung (vgl. Tabelle 7-15)

[726] Vgl. zu den Migrations- und Batchprogrammen Kapitel 7.2.2.1.

In den nachfolgenden Tabellen werden die für die Fallstudie relevanten Attribute der Problemdatenbank dargestellt.[727] Anschließend wird der Umgang mit der Problemdatenbank erläutert.

[727] Vgl. Interview mit Testmanager 1 (GDV) vom 08.07.05 (Ereignis 8) in Anhang C.3 sowie Fallstudiendokumente 12, 14, 16, 24 und 25 in Anhang C.1.

Attribute der Problemdatenbank

Attribut	Erläuterung zum Attribut
Berichtsnummer	▪ Welche eindeutige Nummer hat der Problembericht? ▪ Nummer wird automatisch festgelegt und kann durch den Benutzer nicht geändert werden.
gefunden in Version	▪ Auf welches Release bezieht sich das wahrgenommene Problem? ▪ Auswahlliste mit fest definierten Werten
behoben in Version	▪ In welchem Release wurde das Problem behoben? ▪ Auswahlliste mit fest definierten Werten
Datum / Uhrzeit	▪ Wann wurde der Problembericht angelegt? ▪ Das Datum und die Uhrzeit werden automatisch durch die Software festgesetzt und können durch den Benutzer nicht geändert werden.
Tester	▪ Welche Person aus welcher Organisation hat das Problem wahrgenommen? ▪ Auswahlliste mit vorgegebenen Namen und Organisationszugehörigkeit
Klassifizierung	▪ Um was für einen Problembericht handelt es sich? ▪ Mögliche Werte sind: o Fehler: Ein Fehlverhalten wird wahrgenommen und es wird ein Implementierungsfehler in der Software vermutet. o Änderungsantrag: Es liegt ein Änderungsantrag vor. D. h. ausgehend von einer Änderung in einem Anforderungsdokument wird eine Änderung der Software gefordert. o zurückgewiesen: Der Problembericht wurde zurückgewiesen. o doppelt: Der Problembericht ist redundant zu einem existierenden und wird deshalb geschlossen.
Grund für Zurückweisung	▪ Warum wurde ein Problembericht zurückgewiesen? ▪ Sofern ein Bericht vom Typ „zurückgewiesen“ vorliegt, muss einer der folgenden Gründe ausgewählt werden: Benutzerfehler, Systemrestriktion, nicht reproduzierbar.
Fehlerklasse	▪ Welche Schwere hat ein Fehlverhalten? ▪ Sofern ein Bericht vom Typ „Fehler“ vorliegt, wird das Fehlverhalten (im Sinne dieser Arbeit) nach seiner Schwere kategorisiert. ▪ Die Schwere eines Fehlverhaltens beschreibt die Intensität der Auswirkung eines Fehlers für die Produktion und gibt somit auch eine Reihenfolge für die Behebung vor. ▪ Mögliche Werte sind: schwerer Fehler, mittlerer Fehler, leichter Fehler, offener Punkt.
aufgetreten in Umgebung	▪ Auf welcher technischen Produktions- oder Testumgebung ist ein Fehlverhalten aufgetreten? ▪ Insbesondere im Rahmen der Tests wird ein Softwarerelease von LV-Neu auf verschiedenen technischen Umgebungen getestet. ▪ Um die Lokalisierung eines Fehlers zu erleichtern, kann in einem freien Textfeld die entsprechende Umgebung angegeben werden.
Kontaktperson	▪ Wer koordiniert die Behebung eines Problemberichts? ▪ Auswahlliste mit vorgegebenen Namen und Organisationszugehörigkeit
Priorität	▪ Wie dringlich ist die Umsetzung eines Änderungsantrags? ▪ Sofern ein Problembericht vom Typ „Änderungsantrag“ vorliegt, wird die Dringlichkeit seiner Umsetzung angegeben. ▪ Mögliche Werte sind: hoch, mittel, gering.
Aufwandsschätzung	▪ Welcher Aufwand wird für die Umsetzung eines Änderungsantrags geschätzt? ▪ Sofern ein Problembericht vom Typ „Änderungsantrag“ vorliegt, wird der Aufwand für die Umsetzung geschätzt. ▪ Auswahlliste mit fest definierten Werten. Mögliche Werte sind: nicht angegeben, weniger als 4 Personentage, 4 bis 8 Personentage, mehr als 8 Personentage.
Fehlerbehebung angefordert für Version	▪ Für welches Release der Anwendungssoftware wird die Fehlerbehebung angefordert? ▪ Auswahlliste mit fest definierten Werten
Behebung angefordert für Version	▪ Für welches Release der Anwendungssoftware wird die Umsetzung eines Änderungsantrags angefordert? ▪ Auswahlliste mit fest definierten Werten

Tabelle 7-11: Attribute der Problemdatenbank im Bereich Identifikation

Attribut	Erläuterung zum Attribut
Komponentenzuordnung	▪ Auf welche Komponente bezieht sich das auftgetretene Problem oder der Änderungsantrag? ▪ Auswahlliste mit fest definierten Werten
übergeben an / liegt bei	▪ Wer ist mit der Behebung des Fehlers bzw. der Umsetzung der gewünschten Änderung beauftragt? ▪ Auswahlliste mit vorgegebenen Namen und Organisationszugehörigkeit
übergeben am / seit	▪ Wann wurde die Person beauftragt? ▪ Datumsfeld, dass manuell ausgefüllt werden muss.
aktueller Status	▪ Welchen Bearbeitungsstatus hat der angelegte Bericht? ▪ Eines der folgenden Werte kann ausgewählt werden: o erfasst o in Analyse o Analyse abgeschlossen o in Bearbeitung o Behebung abgeschlossen o in Test o Problembericht abgeschlossen o Rückfrage
in Bearbeitung seit	▪ Seit wann ist das Problem in Bearbeitung? ▪ Datumsfeld, das automatisch gefüllt wird, sobald vom Anfangsstatus „erfasst“ ein Wechsel in einen anderen Status erfolgt.
Problem behoben am	▪ Seit wann ist das Problem behoben? ▪ Datumsfeld, das automatisch gefüllt wird, sobald ein Wechsel zum Status „Behebung abgeschlossen“ erfolgt.
abgeschlossen seit	▪ Seit wann ist das Problem abgeschlossen? ▪ Datumsfeld, das automatisch gefüllt wird, sobald ein Wechsel zum Status „Problembericht abgeschlossen“ erfolgt.
Bestandskorrektur	▪ Ist eine Korrektur der Bestandsdaten[728] erforderlich? ▪ Folgende Werte sind möglich: o nicht angegeben o erforderlich o nicht erforderlich

Tabelle 7-12: Attribute der Problemdatenbank im Bereich Status der Problembearbeitung

Attribut	Erläuterung zum Attribut
Kurztext	▪ Was ist das Problem? ▪ Einzeiliges Textfeld zur kurzen Beschreibung eines Problems.
Langtext	▪ Was ist das Problem? ▪ Mehrzeiliges Textfeld zur ausführlichen Beschreibung eines Problems.
Umgehungsmöglichkeit	▪ Wie kann das Problem umgangen werden? ▪ Mehrzeiliges Textfeld zur Beschreibung einer etwaigen Möglichkeit, ein Problem zu umgehen.
Anhänge zur Beschreibung	▪ Gibt es Dokumente, die die Beschreibung eines Problems unterstützen (Abbildungen von Bildschirmmasken etc.)? ▪ Es können Dateien unterschiedlichen Formats an den Problembericht angehangen werden.

Tabelle 7-13: Attribute der Problemdatenbank im Bereich Problembeschreibung

[728] Vgl. Kapitel 7.2.2.1.

Attribut	Erläuterung zum Attribut
Ergebnis der Analyse	▪ Was ist das Ergebnis der Problemanalyse? ▪ Mehrzeiliges Textfeld zur Beschreibung des Ergebnisses der Problemanalyse.
Anhänge zur Analyse	▪ Gibt es Dokumente, die die Analyse eines Problems unterstützen (Abbildungen von Bildschirmmasken etc.)? ▪ Es können Dateien unterschiedlichen Formats an den Problembericht angehangen werden.

Tabelle 7-14: Attribute der Problemdatenbank im Bereich Analyse

Attribut	Erläuterung zum Attribut
Behebung in Version	▪ In welcher Version wurde das Problem behoben? ▪ Auswahlliste mit fest definierten Werten.
Detailbeschreibung der Lösung	▪ Wie wurde das Problem gelöst? ▪ Mehrzeiliges Textfeld zur Beschreibung der Problemlösung.
Anhänge zur Behebung	▪ Gibt es Dokumente, die die Problemlösung beschreiben (Abbildungen von Bildschirmmasken etc.)? ▪ Es können Dateien unterschiedlichen Formats an den Problembericht angehangen werden.

Tabelle 7-15: Attribute der Problemdatenbank im Bereich Behebung

Umgang mit der Problemdatenbank

In Abbildung 7-4 ist der Lebenzyklus eines Problemberichts in Form eines Zustandsdiagramms dargestellt.[729]

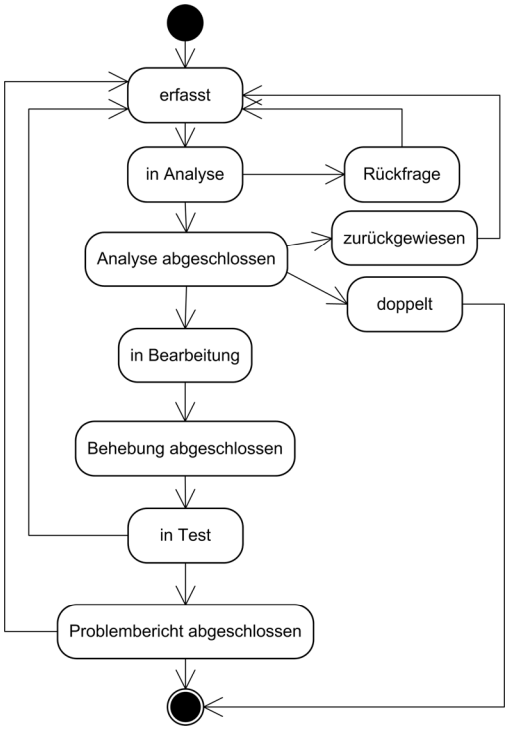


Abbildung 7-4: Zustandsdiagramm für einen Problembericht

[729] Vgl. zum Umgang mit der Problemdatenbank Fallstudiendokumente 3, 23 und 27 in Anhang C.1, die E-Mail-Antworten von Testmanager 1 (GDV) vom 14.06.05 (Ereignis 5) und vom 19.07.05 (Ereignis 9) in Anhang C.3 sowie die Interviews mit ihm vom 08.07.05 (Ereignis 8) und vom 19.07.05 (Ereignis 10).

Sobald ein Tester eine Abweichung in der Anwendungssoftware LV-Neu wahrnimmt,[730] wird ein Problembericht angelegt und in den Status „erfasst“ gesetzt. Wenn ein Entwickler einen Problembericht entgegennimmt, ist der Problembericht im Status „in Analyse“. Ist die Problembeschreibung nicht klar formuliert oder enthält sie offene Punkte, kann der Entwickler eine Rückfrage an den jeweiligen Tester stellen. In diesem Fall wechselt der Problembericht auf den Status „Rückfrage“. Schließt der Entwickler die Problemanalyse ab, geht der Problembericht in den Status „Analyse abgeschlossen“ über. Mit Abschluss der Analyse steht zudem fest, ob der Problembericht

- zurückgewiesen wird,
- doppelt zu einem anderen Problembericht ist,
- ein Änderungsantrag ist oder
- ein Fehlerbericht ist.[731]

Ein Fehlerbericht liegt vor, wenn die Ursache für eine Abweichung ein originärer Implementierungsfehler[732] ist. In Abgrenzung hierzu liegt ein Änderungsantrag vor, wenn ein Implementierungsfehler als Folge eines Anforderungsfehlers vorliegt. In der Problemdatenbank werden nur Änderungsanträge erfasst, sofern sie mit einem geringeren Aufwand als 10 Personentagen umgesetzt werden können.[733] Übersteigt der Aufwand diese Grenze, wird der Änderungsantrag außerhalb der Problemdatenbank verwaltet.

Sobald der Entwickler die Umsetzung eines Änderungsantrags oder die Behebung eines Implementierungsfehlers beginnt, ist der Problembericht im Status „in Bearbeitung“. Ist die Umsetzung bzw. Behebung aus Sicht des Entwicklers erfolgreich abgeschlossen, setzt er den Problembericht auf Status „Behebung abgeschlossen“. Die Umsetzung bzw. Behebung muss noch durch den Tester abgenommen werden, der ursprünglich den Problembericht angelegt hat (Wechsel in den Status „in Test“). Stellt der Tester weiterhin eine Abweichung fest, aktualisiert er die Problembeschreibung und setzt den Status des Problemberichts auf „erfasst“. Andernfalls wird der Problembericht geschlossen (Status „Problembericht abgeschlossen“).

---

[730] Sofern eine Abweichung durch einen Benutzer während der produktiven Nutzung der Software wahrgenommen wird, wird der Problembericht durch jene Benutzer angelegt, die auch in den Teststufen Abnahmetest und Update-Test mitwirken.

[731] Vgl. hierzu die Attribute „Klassifizierung“ und „Grund der Zurückweisung“ in Tabelle 7-11.

[732] Vgl. zum Begriff eines originären Implementierungsfehlers Kapitel 4.1.

[733] Vgl. Interview mit Leiter Qualitätsmanagement (IT-Power) vom 08.09.05 (Ereignis 24) in Anhang C.3.

7.4.2.1.2 Nachträgliche Fehlerklassifikation

Auswahlkriterien für die Problemberichte

Auf der Grundlage der Problemberichte zu LV-Neu in der Problemdatenbank wurden Implementierungsfehler aus der Entwicklung und Produktion der Releases 4.2 und 4.3 ermittelt und nachträglich gemäß dem in Kapitel 6.2.3 dargestellten Schema klassifiziert. Es wurden sowohl originäre Implementierungsfehler als auch Implementierungsfehler berücksichtigt, die Folgefehler waren. Da das Klassifikationsschema sowohl Informationen zur Fehlerentdeckung als auch –behebung erfordert, konnten ausschließlich jene Fehler klassifiziert werden, die bereits behoben worden sind.

Anhand folgender fünf Auswahlkriterien wurden relevante Problemberichte bestimmt:

1. Ein Problem muss im Release 4.2 oder 4.3 entdeckt worden sein *und*
2. ein Problem muss im Release 4.2, 4.3 oder 4.4 behoben worden sein *und*
3. ein Problem muss spätestens im Nachfolgerelease behoben worden sein *und*
4. der Problembericht muss vom Typ Fehlerbericht oder Änderungsantrag sein *und*
5. der Problembericht muss den Bearbeitungsstatus „Problembericht abgeschlossen“, „Behebung abgeschlossen“ oder „in Test“ haben.

Die Kriterien 1 bis 3 stellen sicher, dass die Releases 4.2 und 4.3 unter gleichen Rahmenbedingungen untersucht werden. D. h., dass

- alle erfassten Fehler, die in 4.2 gefunden worden sind, in 4.2 oder spätestens in 4.3 behoben sein müssen sowie
- alle erfassten Fehler, die in 4.3 gefunden worden sind, in 4.3 oder spätestens in 4.4 behoben sein müssen.

Das vierte Kriterium ermöglicht, dass sowohl originäre Implementierungsfehler als auch Implementierungsfehler als Folgefehler berücksichtigt werden. Mittels des fünften Kriteriums wird die Voraussetzung dafür geschaffen, dass die entdeckten Fehler vollständig klassifiziert werden können, da ihre Behebung abgeschlossen ist.

Die ermittelten Problemberichte wurden zudem dahingehend untersucht, ob sie einen oder ggf. mehrere Implementierungsfehler beschreiben.

Bestimmung der relevanten Problemberichte

Da zur nachträglichen Fehlerklassifikation auch freie Textfelder ausgewertet werden müssen, war es nicht möglich, die Fehlerklassifikation zu automatisieren. In der Problemdatenbank besteht die Möglichkeit, die Problemberichte nach dem Release zu gruppieren, in dem ein Pro-

blem gefunden wurde (vgl. Tabelle 7-11). Jedoch zeigte sich, dass dieses Attribut in den Problemberichten nicht korrekt eingetragen worden ist.

Da ein Problembericht zeitnah zur Wahrnehmung eines Problems angelegt wurde, war das Erstellungsdatum ein Indikator für das relevante Release. Da das Datum zudem automatisch durch die Problemdatenbank erzeugt wurde, war es nicht fehlerbehaftet wie ein Eingabefeld, das durch einen Benutzer manuell ausgefüllt werden musste. Jedoch konnte allein auf der Grundlage des Erstellungsdatums ein Problembericht nicht eindeutig einem Release zugeordnet werden, da zeitgleich ein Release in Produktion und eines in Entwicklung war. Diesen Sachverhalt veranschaulichen die Abbildung 7-5 und Abbildung 7-6. Deshalb wurden zusätzlich die angegebenen Informationen in der Problembeschreibung, -analyse und –behebung berücksichtigt. Die Person, die einen Problembericht angelegt hatte, gab aufgrund ihrer Rolle ebenfalls Aufschluss darüber, ob ein Problembericht zu einem bestimmten Datum sich auf das in Produktion oder in Entwicklung befindliche Release bezog. Folglich wurde die Rolle jeder Person bestimmt, die gemäß der Problemdatenbank Problemberichte angelegt hatte. Die Zuordnung von Rollen zu Personen erfolgte unabhängig voneinander mit dem Testmanager 1 (GDV) und dem Tester (GDV).[734] Das konsolidierte Ergebnis wurde dann als weitere Unterstützung zur eindeutigen Klassifikation eingesetzt.[735] Anhand der Rolle einer Person wurden auch die möglichen QS-Kriterien bestimmt, die zur Entdeckung eines Fehlers geführt haben.

Die identifizierten Rollen sind die folgenden:

- Benutzer der GDV

  Benutzer der Anwendungssoftware LV-Neu, die an keiner Teststufe mitwirken und während der produktiven Nutzung der Anwendungssoftware ein Fehlverhalten der Software feststellen können.
- Fachbereichstester der GDV

  Erfahrene Benutzer der Anwendungssoftware LV-Neu aus den Fachbereichen der GDV, die parallel zum Testteams der GDV im Abnahme- und Update-Test nach eigenem Testkonzept testen und ebenso auch Fehlverhalten der Software während der produktiven Nutzung feststellen können.
- Fachliche Tester der GDV

  Tester des Kunden GDV, die auf der Grundlage von Fachkonzepten testen und *keine* Benutzer der Anwendungssoftware LV-Neu sind. Hierbei können verschiedene Test-

[734] Vgl. Interview mit Testmanager 1 (GDV) vom 22.07.05 (Ereignis 12) und mit Tester (GDV) vom 05.09.05 (Ereignis 22) in Anhang C.3.

[735] Vgl. Fallstudiendokument 35 in Anhang C.2.

teams unterschieden werden, die jeweils genau einen Schwerpunkt haben und unabhängig voneinander arbeiten. Die Schwerpunkte lauten: versicherungsmathematischer Rechenkern, Produktdaten und Plausibilitätsbedingungen sowie ein vollständig integriertes Anwendungsrelease.

- Entwicklertester der GDV
  Entwickler der GDV, die die von der GDV beigesteuerten Komponenten testen. Hierbei können zwei unabhängig arbeitende Testteams unterschieden werden, die entweder den versicherungsmathematischen Rechenkern oder die Produktdaten und Plausibilitätsbedingungen testen.
- Entwicklertester der IT-Power
  Entwickler der IT-Power, die auf der Grundlage von Fachkonzepten testen und insbesondere Wissen über den von IT-Power entwickelten Anwendungskern sowie den Schnittstellen zu den Randsystemen verfügen.

Anhand der oben beschriebenen Informationen war eine eindeutige Zuordnung aller Problemberichte und damit auch aller Implementierungsfehler zum Release möglich, in dem sie aufgetreten sind.[736]

[736] Vgl. hierzu Kapitel 7.4.2.1.2.

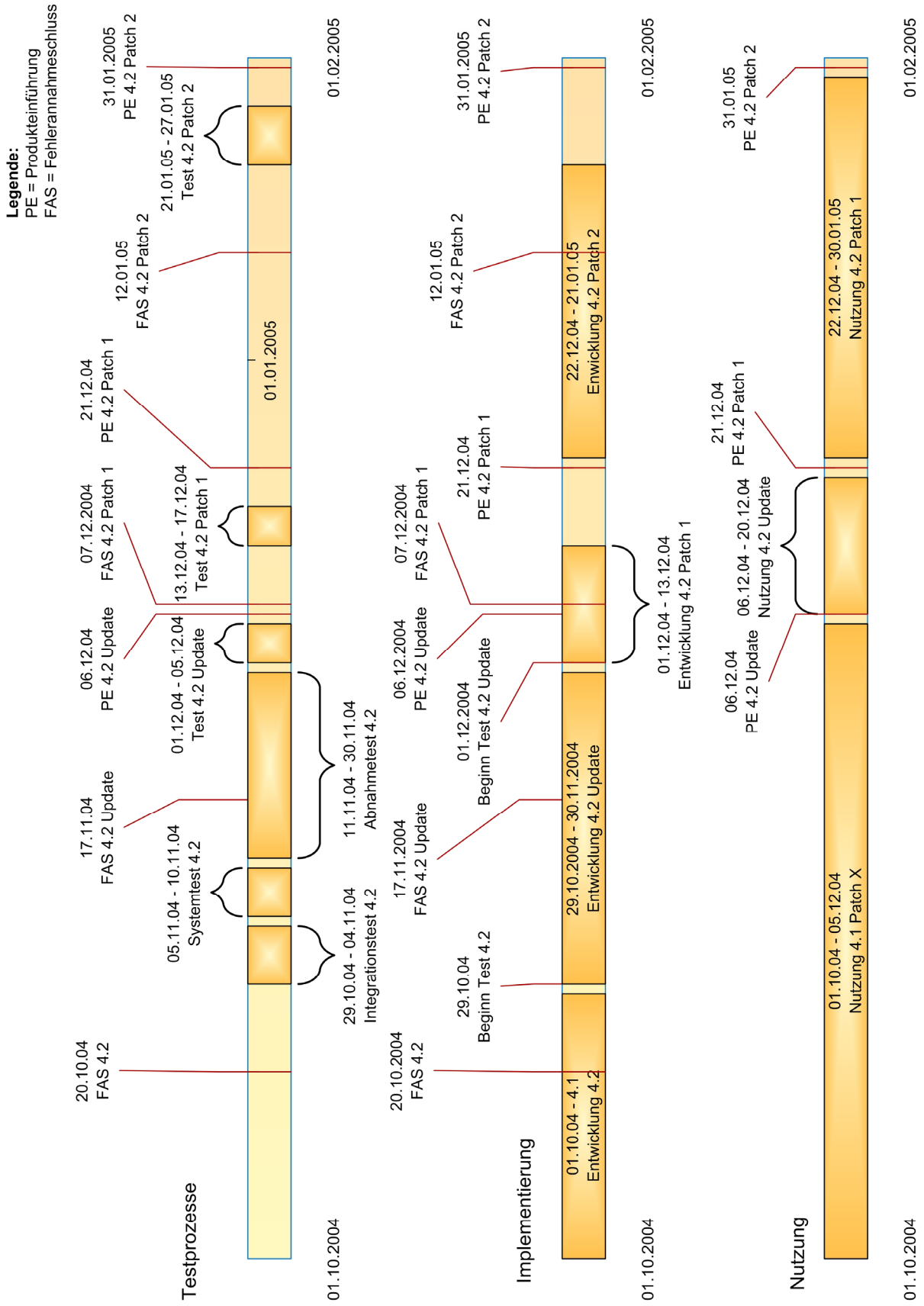


Abbildung 7-5: Zeitplan für Implementierung und Test von Release 4.2 und die produktiven Nutzung von Release 4.1

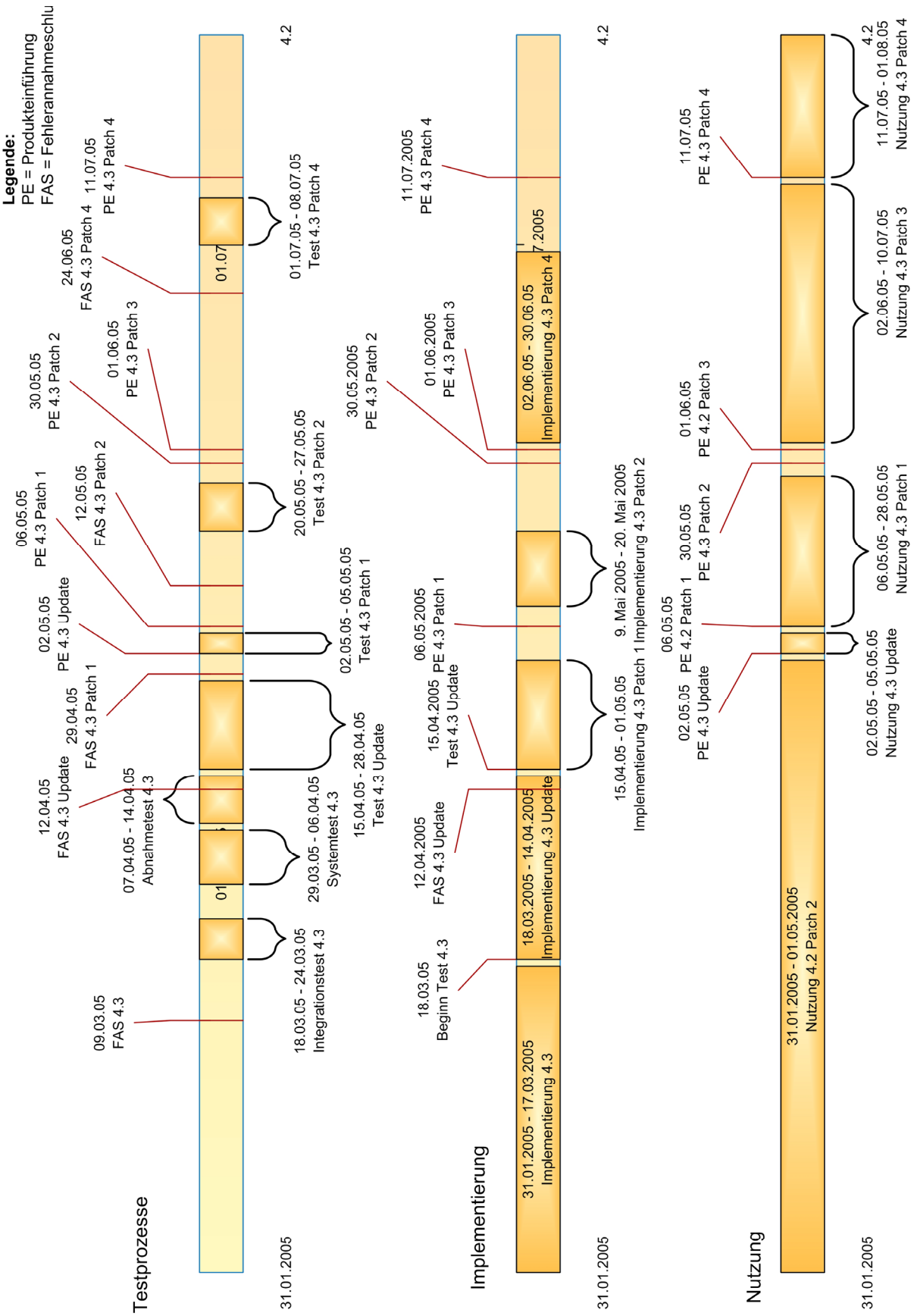


Abbildung 7-6: Zeitplan für Implementierung und Test von Release 4.3 und die produktiven Nutzung von Release 4.2

Klassifikation einer kleinen Fehlerstichprobe als Pilotprojekt

Zunächst wurde eine kleine Stichprobe von 50 Problemberichten untersucht und versucht, identifizierte Implementierungsfehler nachträglich zu klassifizieren. Die aufgetretenen offenen Punkte wurden mit dem internen Berater (IT-Power) und dem Testmanager 1 (GDV) geklärt.[737] Mittels dieser ausgewerteten Stichprobe sowie den in diesem Zusammenhang erhobenen Informationen war es möglich zu bestimmen, in welchem Umfang Implementierungsfehler im Kontext der vorliegenden Fallstudie gemäß dem Schema aus Kapitel 6.2.3 nachträglich klassifiziert werden konnten (vgl. Tabelle 7-16).

Attribut	Implementierungsfehler nachträglich klassifizierbar?
Phase	▪ Implementierungsfehler können nachträglich gemäß diesem Attribut klassifiziert werden.
QS-Aktivität	▪ Implementierungsfehler können nachträglich gemäß diesem Attribut klassifiziert werden.
QS-Kriterium	▪ Implementierungsfehler können ohne erheblichen Aufwand nicht nachträglich gemäß diesem Attribut klassifiziert werden.
Auswirkung auf Kunden	▪ Implementierungsfehler können nachträglich gemäß diesem Attribut klassifiziert werden.
Komponente	▪ Implementierungsfehler können nachträglich gemäß diesem Attribut klassifiziert werden.
Gegenstand der Änderung	▪ Implementierungsfehler können in der Regel nachträglich gemäß diesem Attribut klassifiziert werden.
Art der Änderung	▪ Implementierungsfehler können in der Regel nachträglich gemäß diesem Attribut klassifiziert werden.
Ursprung	▪ Implementierungsfehler können nachträglich gemäß diesem Attribut klassifiziert werden.
Vorgeschichte	▪ Implementierungsfehler können aufgrund fehlender Informationen nicht nachträglich gemäß diesem Attribut klassifiziert werden.

Tabelle 7-16: Nachträgliche Klassifizierung von Implementierungsfehlern in der vorliegenden Fallstudie

Nachfolgend wird der Ablauf der nachträglichen Fehlerklassifikation für jedes Attribut erläutert.

Attribute Phase und QS-Aktivität

Das Erstellungsdatum des Problemberichts, die Rolle der Person, die einen Problembericht angelegt hat, sowie die Informationen und Anhänge in der Problembeschreibung, -analyse und –behebung waren ausreichend, um die die Attribute Phase und QS-Aktivität mit Werten zu belegen.

Attribut QS-Kriterium

Die Problemberichte gaben kaum Anhaltspunkte darüber, welches QS-Kriterium dazu beitrug, einen Implementierungsfehler aufzudecken. Theoretisch bestand die Möglichkeit, den

[737] Vgl. Interview mit internem Berater (IT-Power) und Testmanager 1 (GDV) vom 08.07.05 (Ereignis 8) sowie die E-Mail-Antwort vom Testmanager 1 (GDV) vom 19.07.05 (Ereignis 9) in Anhang C.3.

Testfällen für die automatisierten Funktionstests jeweils ein QS-Kriterium zuzuordnen und den Implementierungsfehlern das QS-Kriterium des Testfalls zu übertragen, der den Fehler aufdeckte.[738] Praktisch würde dies jedoch die nachträgliche Klassifikation von 10 000 Testfällen bedeuten. Zudem wurde nicht festgehalten, welcher Testfall welche Implementierungsfehler aufgedeckt hat. Bei den Interviews kristallisierte sich jedoch heraus, dass die Personen, die Problemberichte anlegten, aufgrund ihrer jeweiligen Rolle verschiedene Gruppen von QS-Kriterien einsetzten. Folglich konnte aufgrund der Zuordnung von Problemberichten zu Personen sowie von Personen zu Rollen zumindest teilweise einzelne Fehler mit einer *Gruppe von QS-Kriterien* verknüpft werden.

Rolle	Eingesetzte QS-Kriterien
Benutzer der GDV	▪ unbekannt
Fachbereichstester der GDV	▪ Funktionsvariation (primär) ▪ Funktionsabdeckung (primär) ▪ Funktionensequenz (sekundär)
Fachliche Tester der GDV	▪ Funktionsabdeckung (primär) ▪ Funktionensequenz (sekundär)
Entwicklertester der GDV	▪ unbekannt
Entwicklertester der IT-Power	▪ unbekannt

Tabelle 7-17: Rollen und eingesetzte QS-Kriterien

Attribut Auswirkung auf den Kunden

Implementierungsfehler werden in dem betrachteten Softwareentwicklungsvorhaben als eine Abweichung vom Fachkonzept verstanden. Ein Fachkonzept im Sinne von IT-Power ist ein Dokument, das sowohl fachliche Anforderungen als auch Entscheidungen aus dem Feinentwurf umfasst. In beiden Fällen sind der Schwerpunkt jedoch Funktionen der Software. Folglich kann in den Fällen, in denen ein Problembericht vom Typ Fehler ist, in der Regel für das Attribut „Auswirkung auf den Kunden“ der Wert „bekannte Softwareanforderung“ ausgewählt werden (vgl. Tabelle 6-9). Ein Änderungsantrag in der Problemdatenbank impliziert einen Implementierungsfehler, der die Folge eines Fehlers in einem Fachkonzepts ist. In solchen Fällen kann in der Regel für das Attribut „Auswirkung auf den Kunden“ der Wert „unbekannte Softwareanforderung“ ausgewählt werden (vgl. Tabelle 6-9).

Nur in wenigen Fällen wurden in den Problemberichten Implementierungsfehler beschrieben, die Auswirkungen auf die Benutzbarkeit, Integrität und Sicherheit, Zuverlässigkeit oder Versionswechsel hatten.[739]

---

[738] Vgl. hierzu Bassin, Biyani, Santhanam /Sydney Olympics/ 267 f.

[739] Vgl. hierzu Tabelle 6-9.

Attribut Komponente

Obwohl in der Problemdatenbank explizit ein Feld für die Komponente vorgesehen ist, konnten darin enthaltene Informationen nicht ohne weiteres übernommen werden. Komponenten im Sinne der Problemdatenbank waren meist fachliche Geschäftsvorgänge, bei denen ein Fehlverhalten der Software aufgetreten war. Komponenten in diesem Sinne erfüllen nicht die Bedingung, dass ein Implementierungsfehler eindeutig genau einer Komponente zu geordnet werden kann,[740] da ein Implementierungsfehler Fehlverhalten der Anwendungssoftware in verschiedenen Geschäftsvorgängen verursachen kann.

In der vorliegenden Fallstudie war eine nachträgliche eindeutige Zuordnung nur zu folgenden Komponenten möglich:

- Anwendungskern
- Schnittstellen zu Randsystemen
- versicherungsmathematischer Rechenkern
- Produktdaten und Plausibilitätsbedingungen

Bei einer direkten Fehlerklassifikation wäre es zweckmäßig, die Komponente Anwendungskern noch weiter zu zerlegen. Bei der nachträglichen Fehlerklassifikation in der vorliegenden Fallstudie wäre dies jedoch mit einem erheblichen Aufwand verbunden, da entsprechende Informationen nicht bereit standen.

Bei der nachträglichen Fehlerklassifikation stellte sich heraus, dass die Komponentenangabe nur als ein vorläufiger Anhaltspunkt für die eigentliche Komponentenzuordnung dienen konnte. Hilfreicher in diesem Zusammenhang war die Angabe darüber, welche Person für die Behebung zuständig war, da die Komponenten von unterschiedlichen Organisationseinheiten entwickelt wurden.[741] Zusätzlich wurden die Informationen und Anhänge zur Problemanalyse und -behebung berücksichtigt (vgl. Tabelle 7-14 und Tabelle 7-15).

Sofern ein Fehler nicht eindeutig einer Komponente zugeordnet werden konnte, wurde der Wert „unbekannt“ gewählt.

Attribute Gegenstand der Änderung und Art der Änderung

Um die Attribute „Gegenstand der Änderung“ und „Art der Änderung“ für jeden Implementierungsfehler mit angemessenen Werten zu belegen, wurden die Informationen und Anhänge zur Problemanalyse und -behebung in einem Problembericht detailliert ausgewertet. In den Fällen, in denen eine angemessene Belegung aufgrund der vorliegenden Informationen nicht

[740] Vgl. hierzu Kapitel 6.3.

[741] Vgl. hierzu Kapitel 7.2.2.2.

möglich war, wurden die Attribute auf den Wert „unklar“ gesetzt. Sofern Informationen zur Problembehebung im Problemberichten fehlten, wurden die Attribute auf den Wert „unbekannt“ gesetzt

Ursprung

Das Attribut Ursprung konnte in der vorliegenden Fallstudie sehr einfach mit Werten belegt werden. Der Attributwert „portiert“ war für das Softwareentwicklungsvorhaben LV-Neu irrelevant, da kein Code portiert wurde. Die Anwendungssoftware LV-Neu wurde durch IT-Power als Auftragnehmer für den Auftraggeber GDV entwickelt. Zugleich entwickelten aber zwei unterschiedliche organisatorische Einheiten von GDV den versicherungsmathematischen Rechenkern sowie die Produktdaten und Plausibilitätsbedingungen. Obwohl folglich GDV der Abnehmer der Anwendungssoftware war, traten diese beiden organisatorischen Einheiten der GDV als Zulieferer von IT-Power auf.[742] Da der versicherungsmathematische Rechenkern auch in einer anderen Anwendungssoftware eingesetzt wurde,[743] hatte er den Charakter einer Standard-Bibliothek, die wieder verwendet wurde.

Die Komponente Produktdaten und Plausibilitätsbedingungen wurde aus Sicht von IT-Power speziell für die Anwendungssoftware LV-Neu extern entwickelt. Entsprechende Implementierungsfehler hatten folglich für das Attribut Ursprung den Wert „Entwicklung durch externen Auftragnehmer“.

Der Anwendungskern und die Schnittstellen zu Randsystemen wurden innerhalb von IT-Power entwickelt. Deshalb wurden entsprechende Implementierungsfehler für das Attribut Ursprung mit dem Wert „Eigenentwicklung“ belegt.

In den Fällen, in denen eine angemessene Belegung aufgrund der vorliegenden Informationen nicht möglich war, wurde dieses Attribut auf den Wert „unklar“ gesetzt. Sofern Informationen zur Problembehebung im Problemberichten fehlten, wurden die Attribute auf den Wert „unbekannt“ gesetzt

Attribut Vorgeschichte

Die Informationen in einem Problembericht waren in der Regel nicht ausreichend, um Implementierungsfehler nachträglich gemäß dem Attribut Vorgeschichte zu klassifizieren.

---

[742] Vgl. hierzu Kapitel 7.2.2.2.

[743] Vgl. hierzu Kapitel 7.2.2.1.

Nachträgliche Fehlerklassifikation in der Tabellenkalkulationssoftware Microsoft Excel

Im Anschluss an die nachträgliche Fehlerklassifikation für eine kleine Stichprobe von Problemberichten sowie der Klärung der offenen Punkte, wurden alle relevanten Problemberichte nach Implementierungsfehlern durchsucht.

Das Ergebnis der nachträglichen Fehlerklassifikation wurde mit Hilfe der Tabellenkalkulationssoftware Microsoft Excel festgehalten. Die entsprechende Datei hat die in Tabelle 7-18 beschriebene inhaltliche Struktur.

Attribut	Beschreibung des Attributs
Berichtsnummer	▪ Wie lautet die Nummer des Problemberichts, in dem der Implementierungsfehler beschrieben wird?
Fehler aufgetreten am	▪ Wann wurde der Implementierungsfehler entdeckt?
Fehler behoben am	▪ Wann wurde der Implementierungsfehler behoben?
Fehler gefunden in Release	▪ In welchem Release wurde der Implementierungsfehler entdeckt? ▪ Aufgrund der Auswahlkriterien sind nur folgende Releasenummern möglich: 4.2 oder 4.3.
Fehler behoben in Release	▪ In welchem Release wurde der Implementierungsfehler behoben? ▪ Aufgrund der Auswahlkriterien sind nur folgende Releasenummern möglich: 4.2, 4.3 oder 4.4.
Tester	▪ Wie heißt die Person, die den Problembericht angelegt hat, in dem der Implementierungsfehler beschrieben wird?
Organisation des Testers	▪ Zu welcher Organisation gehört die Person, die den Problembericht angelegt hat, in dem der Implementierungsfehler beschrieben wird? ▪ Mögliche Ausprägungen sind: IT-Power oder GDV.
Rolle des Testers	▪ Welche Rolle hatte die Person, die den Problembericht angelegt hat, in dem der Implementierungsfehler beschrieben wird? ▪ Folgende Ausprägungen sind möglich: o Benutzer der GDV o Fachbereichstester der GDV o Fachliche Tester der GDV o Entwicklertester der GDV o Entwicklertester der IT-Power
Bearbeitungsstatus	▪ Bearbeitungsstatus des Problemsberichts, in dem der Implementierungsfehler beschrieben wird. ▪ Aufgrund der Auswahlkriterien sind nur folgende Ausprägungen möglich: „Problembericht abgeschlossen“, „Behebung abgeschlossen“ oder „in Test“.
Phase	▪ Attribut gemäß dem Fehlerklassifkationsschema in Kapitel 6.2.3.
QS-Aktivität	▪ Attribut gemäß dem Fehlerklassifkationsschema in Kapitel 6.2.3.
Gruppe von QS-Kriterien	▪ Vgl. Tabelle 7-17.
Auswirkung beim Kunden	▪ Attribut gemäß dem Fehlerklassifkationsschema in Kapitel 6.2.3.
Komponente gemäß Problem-datenbank	▪ Vgl. Tabelle 7-12.
Komponente	▪ Attribut gemäß dem Fehlerklassifkationsschema in Kapitel 6.2.3.
Fehler behoben durch	▪ Wie heißt die Person, die den Implementierungsfehler behoben hat?
Organisation der Person	▪ Zu welcher Organisation gehört die Person, die den Implementierungsfehler behoben hat?
Gegenstand der Änderung	▪ Attribut gemäß dem Fehlerklassifkationsschema in Kapitel 6.2.3.
Art der Ändeurng	▪ Attribut gemäß dem Fehlerklassifkationsschema in Kapitel 6.2.3.
Ursprung	▪ Attribut gemäß dem Fehlerklassifkationsschema in Kapitel 6.2.3.
Kurzbeschreibung des Problems gemäß Problem-datenbank	▪ Vgl. Tabelle 7-13.

Tabelle 7-18: Inhaltliche Struktur der Excel-Datei

7.4.2.2 Charakterisierung der untersuchten Fehlerstichprobe

Der Autor der vorliegenden Arbeit hat am 04.05.05, 01.08.05 und 23.09.05 jeweils einen aktualisierten Stand der Problemdatenbank erhalten. Der letzte Stand umfasst 3076 Problemberichte zur Anwendungssoftware LV-Neu sowie zu den Migrations- und Batchprogrammen[744]. Insgesamt 645 Problemberichte erfüllen die in Kapitel 7.4.2.1 beschriebenen Auswahlkriterien. Diese 645 Berichte beinhalten 678 potenzielle Implementierungsfehler. Die Erfassungszeitpunkte der Einträge dieser Fehlerstichprobe decken den Zeitraum vom 04.11.2004 bis zum 30.08.2005 ab.

Die Analyse der 678 potenziellen Implementierungfehler führt zu folgendem Ergebnis (vgl. Abbildung 7-7):

- 368 der potenziellen Implementierungsfehler sind tatsächliche Fehler in LV-Neu.
- 228 der Fehlerkandidaten sind Fehler in den Randsystemen von LV-Neu, in den zugehörigen Migrations- oder Batchprogrammen sowie in den Bestandsdaten.
- 82 der Fehlerkandidaten sind keine Fehler, d. h. es liegt entweder kein Fehlverhalten der Software vor oder der Fehlerkandidat beschreibt einen bekannten Fehler und ist damit redundant.

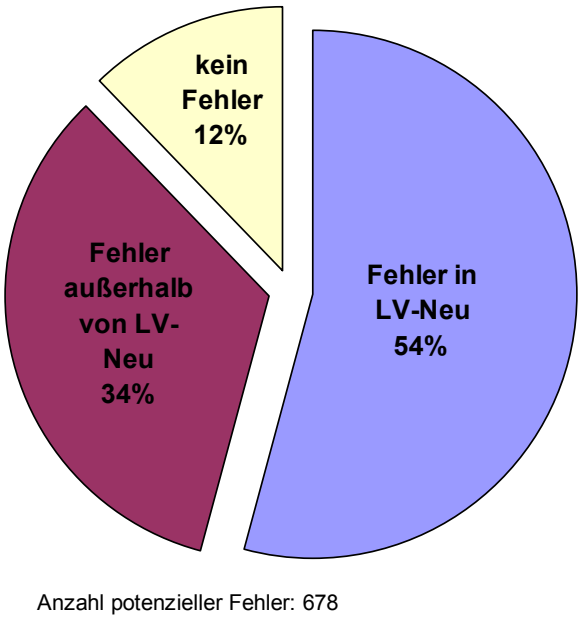


Abbildung 7-7: Verteilung der untersuchten Fehlerkandidaten

Folglich sind nur 54% der Fehlerkandidaten tatsächliche Fehler, die durch die Prozesse der Entwicklungsaufgaben im Vorhaben LV-Neu verursacht worden sind. Bei diesem Verhältnis

[744] Vgl. zu den Migrations- und Batchprogrammen Kapitel 7.2.2.1.

würde eine Fehlerauswertung ohne diese vorherige Datenbereinigung zu falschen Diagnosen führen. Diese 368 zu berücksichtigenden Fehler sind in 343 Problemberichten enthalten.
Die im Kapitel 7.4.2.1 beschriebenen Maßnahmen zur Verbesserung der Datenqualität in der Fehlerstrichprobe führten dazu, dass in den 368 tatsächlichen Fehlern 276 Korrekturen vorgenommen wurden. Dabei wurden folgende Elemente korrigiert:

- In 90 Fällen wurde die Komponentenzuordnung korrigiert.
- In 115 Fällen wurde das Release korrigiert, in der ein Fehler entdeckt worden ist.
- In 69 Fällen wurde das Release korrigiert, in der ein Fehler behoben worden ist.

Von den 368 tatsächlichen Fehlern wurden 225 Fehler im Release 4.2 und 143 Fehler im Release 4.3 entdeckt (vgl. die prozentuale Verteilung in Abbildung 7-8).

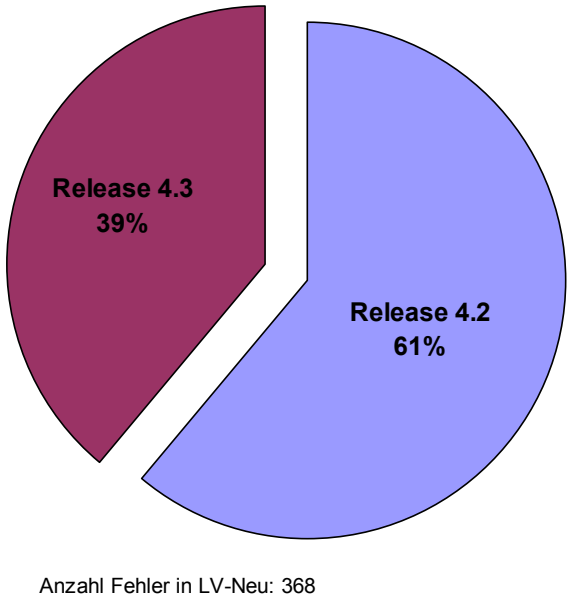
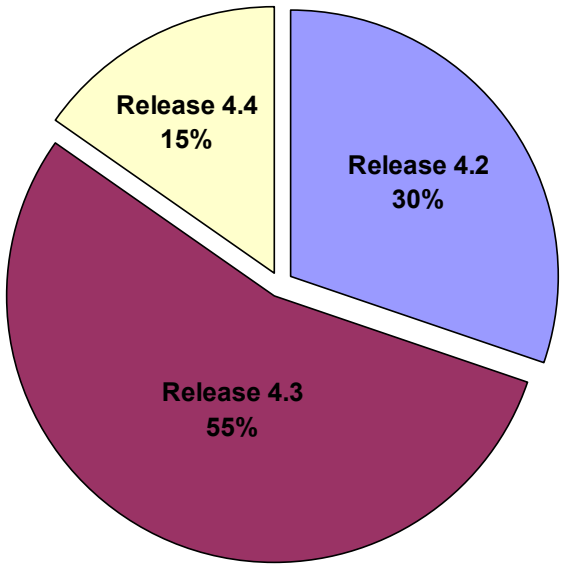


Abbildung 7-8: In welchem Release wurden die Fehler entdeckt?

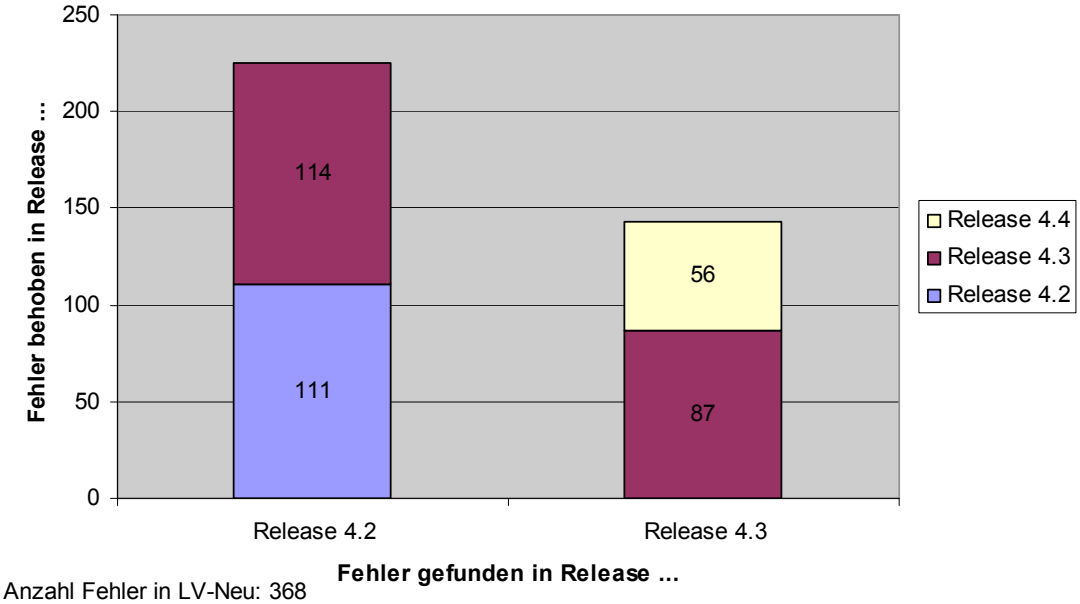
Analog wurden von den 368 tatsächlichen Fehlern in LV-Neu 111 in Release 4.2, 201 in Release 4.3 und 56 in Release 4.4 behoben (vgl. die prozentuale Verteilung in Abbildung 7-9).



Anzahl Fehler in LV-Neu: 368

Abbildung 7-9: In welchem Release wurden die Fehler behoben?

Die Abbildung 7-10 veranschaulicht zu den in einem bestimmten Release entdeckten Fehlern, in welchem Release sie behoben worden sind. Demnach wurden 49% der Fehler, die im Release 4.2 entdeckt worden sind, im gleichen Release und 51% im nachfolgenden Release behoben. Für Fehler, die im Release 4.3 entdeckt worden sind, wurden zu 61% im gleichen und zu 39% im nachfolgenden Release behoben.



Anzahl Fehler in LV-Neu: 368

Abbildung 7-10: In welchen Releases wurden Fehler gefunden und behoben?

Interessant für Charakterisierung ist der untersuchten Fehlerstrichprobe von LV-Neu ist, welchen Status der zu Grunde liegende Problembericht eines Fehlers hat (vgl. Abbildung 7-11). Er gibt Aufschluss darüber, wie wahrscheinlich noch Änderungen am Problembericht und damit an der Fehlerdokumentation sind. 89% der in Release 4.2 und 76% der in Release 4.3 entdeckten Fehler haben den Status „Problembericht abgeschlossen“. D. h., in den Fehlern wurde ein Fehler korrigiert und die Korrektur erfolgreich abgenommen.

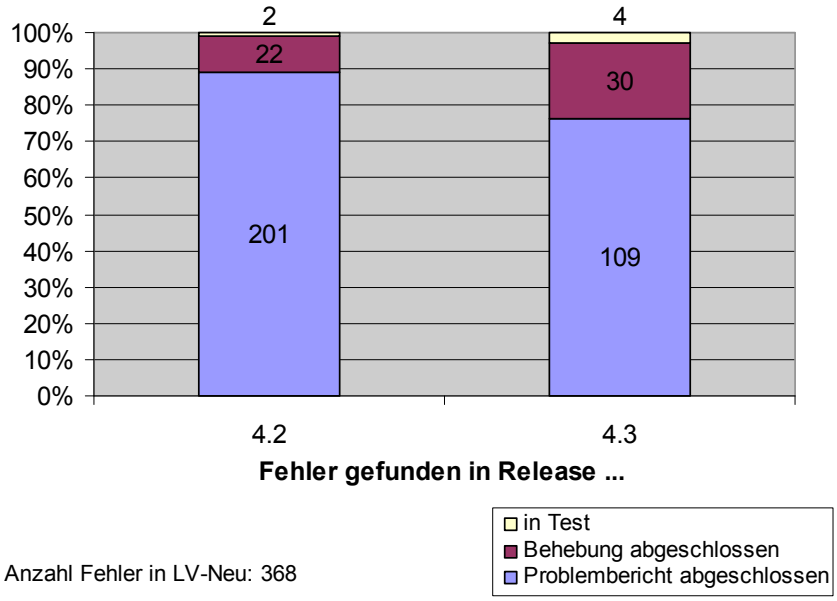


Abbildung 7-11: Welchen Status haben die Fehler?

7.4.2.3 Ergebnis der Fehlerklassifikation

Das Ergebnis der Fehlerklassifikation sind 368 klassifizierte Fehler. Diese werden im Anhang C.6 aufgeführt.

7.4.2.4 Beobachtungen im Rahmen der Fehlerklassifikation

Die nachträgliche Klassifikation von Fehlern konnte mit wenigen Ausnahmen ohne Schwierigkeiten durchgeführt werden. Nachfolgend werden wesentliche Beobachtungen aufgeführt, die während der Fehlerklassifikation gemacht wurden.

Beobachtung 1:

Für Produktionsfehler ist die Klassifikation gemäß dem Attribut QS-Aktivität nicht eindeutig durchführbar.

Chillarege empfiehlt im Fall eines ausgelieferten Fehlers für das ODC-Attribut Aktivität jene QS-Aktivität zu wählen, in der der Fehler am ehesten hätte entdeckt werden können.[745] Streng genommen widerspricht dieser Vorschlag Chillareges Forderung nach einer semantischen Klassifikation.[746] In der vorliegenden Fallstudie ist eine Klassifikation von Produktionsfehler gemäß den Attribut QS-Aktivität nicht eindeutig möglich gewesen. Deshalb wurde sie ausschließlich bei Entwicklungsfehlern vorgenommen.

Beobachtung 2:

> *Komponenten, die extern entwickelt werden, unterliegen Softwareentwicklungsprozesse des externen Auftragnehmers. Bei Fehlern in entsprechenden Komponenten liegen häufig keine Informationen bzgl. der Fehlerkorrektur und damit den Attributen „Gegenstand der Änderung" und „Art der Änderung" vor.*

Eine Besonderheit dieses Softwareentwicklungsvorhabens ist, dass die GDV nicht nur Kunde der Anwendungssoftware ist.[747] Vielmehr entwickelt sie mit dem versicherungsmathematischen Rechenkern sowie den Produktdaten und Plausibilitätsbedingungen zwei wichtige Komponenten im eigenen Hause und tritt folglich zugleich gegenüber IT-Power als Zulieferer auf. Die Entwicklung dieser Komponenten unterliegt Prozessen der GDV und nicht von IT-Power. Entsprechende Fehler werden zwar in der Problemdatenbank eingetragen, jedoch fehlen häufig Informationen zur Fehlerkorrektur. Deshalb wurden die Attribute „Gegenstand der Änderung" und „Art der Änderung" bei diesen Fehlern mit dem Wert „irrelevant" belegt

Beobachtung 3:

> *Implementierungsfehler, die sich nicht im Softwarecode befinden, müssen bei der Fehlerklassifikation so behandelt werden, als befänden sie sich im Softwarecode.*

Das Fehlerklassifikationsschema bezieht sich auf Softwarecode und Entwurfsergebnisse, die einem Softwarecode sehr ähneln. Dieser Umstand führt dazu, dass das Fehlerklassifikationsschema nicht direkt anwendbar ist, wenn ein Implementierungsfehler außerhalb des Softwarecodes vorliegt.

Diese Herausforderung bestand in der Fallstudie bei der Komponente „Produktdaten und Plausibilitätsbedingungen". [748] Die Produktdaten und Plausibilitätsbedingungen werden in ei-

---

[745] Vgl. hierzu Kapitel 5.1.3.3 und IBM Research /ODC/ o. S.

[746] Vgl. hierzu Kapitel 5.1.2.2.

[747] Vgl. zu diesem Absatz Interview mit Leiter Qualitätsmanagement (IT-Power) vom 08.09.05 (Ereignis 24) in Anhang C.3.

[748] Vgl. zu diesem Absatz das Interview mit dem Produktdatenverantwortlichen (GDV) vom 02.11.05 (Ereignis 27) in Anhang C.3.

ner relationen Datenbank mit dem Datenbankmanagementsystem Microsoft Access verwaltet. Für den versicherungsmathematischen Rechenkern wird später automatisch C-Code generiert (nur mit Produktdaten), während für den Anwendungskern komprimierte Binärdateien erzeugt werden.

Fehler in dieser Komponente werden in der relationalen Datenbank korrigiert und nicht im Softwarecode. Die Klassifikation entsprechender Fehler war trotzdem möglich, wenn sie gedanklich als Fehler im Softwarecode behandelt wurden. Diese Komponente beinhaltet im Kern zum einen Parameter mit denen Versicherungsprodukte über bestimmte Versicherungstarife definiert werden und zum anderen aus Plausiblitätsbedingungen. Diese Parameter entsprechen Zuweisungen im Softwarecode. Folglich können beispielsweise Fehler in Parametern als Zuweisungsfehler (Gegenstand der Änderung) berücksichtigt werden. Analog sind Fehler in Plausbilitätsbedingungen vergleichbar mit Bedingungsfehlern im Softwarecode.

Beobachtung 4:

Die nachträgliche Klassifkation von Fehler durch Projektaußenstehende ist sehr zeitaufwändig.

Im Rahmen der Fallstudie wurden 678 potenzielle Implementierungsfehler analysiert und als Ergebnis dieser Analyse 368 tatsächliche Fehler in LV-Neu klassifiziert.[749] Der durchschnittliche Aufwand für die Analyse und Klassifikation beträgt ungefähr 6,5 Minuten pro potenziellen Fehler. Dies ist im Wesentlich aber darauf zurückzuführen, dass die Klassifikation zum einen nachträglich und zum anderen durch den Autor der vorliegenden Arbeit als Projektaußenstehendern durchgeführt worden ist. Eine direkte Klassifikation eines Fehlers durch Tester und Entwickler mit tiefgehender Kenntnis der Software und des Softwareentwicklungsvorhaben wäre sicherlich mit einem deutlich geringeren Aufwand möglich. Nach Chillarege u. a. wird für die Klassifikation von Fehler gemäß ODC durchschnittlich eine bis vier Minuten pro Fehler in Abhängigkeit von der eingesetzten Fehlerverfolgungssoftware benötigt.[750] Auf der Grundlage der Erfahrungen dieser Fallstudie scheint dieser durchschnittliche Aufwand realistisch zu sein.

[749] Vgl. hierzu Kapitel 7.4.2.2.

[750] Vgl. Chillarege u. a. /In-process measurements/ 954.

Beobachtung 5:

Zum besseren Verständnis der Werte der kausalen Attributen „Gegenstand der Änderung“ und „QS-Kriterium“ sind sind Beispiele in Form von von Beschreibungen von Fehlern mit ihrer Klassifikation sehr hilfreich.

In der Fallstudie hat es sich an mehreren Stellen als hilfreich erwiesen, wenn relevante Werte der kausalen Attribute durch konkrete Beispiele wie in IBM Research /ODC/ verdeutlicht wurden. Dies half insbesondere verschiedene Werte eines Attributs besser voneinander abzugrenzen.

7.4.3 Klassifizierte Fehler auswerten

7.4.3.1 Ablauf der Fehlerauswertung

Der Ablauf der Fehlerauswertung orientiert sich an der Beschreibung des Fehleranalyseverfahrens in Kapitel 6.5. Die Auswertung der klassifizierten Fehler wurde mit Hilfe der Tabellenkalkulationssoftware Microsoft Excel durchgeführt.

Aufgrund der Rahmenbedingungen der Fallstudie sind folgende Punkte bei der Fehlerauswertung zu beachten:

- Wegen der nachträglichen Fehlerklassifikation konnten in bestimmten Fällen nicht alle Attribute eines Fehlers mit angemessenen Werten belegt werden. Für solche Fälle wurden spezielle Werte vorgesehen (vgl. 7.4.2.1.2):
  - QS-Kriterium: unbekannt
  - Komponente: unbekannt
  - Gegenstand der Änderung: unklar, unbekannt
  - Art der Änderung: unklar, unbekannt
  - Ursprung: unklar, unbekannt

  Diese Werte wurden in den Auswertungen quantitativ berücksichtigt. Da sie jedoch nicht interpretiert werden konnten, wurden sie im weiteren Verlauf bei der Ableitung von Erkenntnissen ignoriert.
- Komponenten, die extern entwickelt werden, unterliegen Softwareentwicklungsprozessen des externen Auftragnehmers. Folglich können keine Rückschlüsse auf die eigenen Softwareentwicklungsprozesse gezogen werden. Fehler in den entsprechenden Komponenten haben für die Attribute „Gegenstand der Änderung“ und „Art der Änderung“ deshalb den Wert „irrelevant“ erhalten. Dieser Wert wird in den Auswertungen quantitativ berücksichtigt, jedoch im weiteren Verlauf bei der Ableitung von Kenntnissen ignoriert.

- Das Attribut Vorgeschichte konnte bei der Fehlerklassifikation nicht berücksichtigt werden und kann infolge auch nicht im Rahmen der Fehlerauswertung berücksichtigt werden.[751]
- Da das Attribut QS-Aktivität nur auf Entwicklungsfehler angewandt werden konnte,[752] beschränken sich ein- und zweidimensionale Attribute mit diesem Attribut jeweils auf Entwicklungsfehler.
- Die Bestimmung auffälliger Zusammenhänge in den Fehlerdaten[753] musste der Autor der vorliegenden Arbeit aufgrund zeitlicher Restriktion der Schlüsselpersonen aus dem Softwareentwicklungsprojekt ohne ihr Beisein durchführen. Aus diesem Grund wurden ausschließlich Auswertungen interpretiert, deren Differenz zwischen beobachteter und erwarteter relativer Häufigkeit positiv war, da sie tendenziell weniger Domänenwissen über das Softwareentwicklungsprojekt erfordern.

7.4.3.2 Ergebnisse der Fehlerauswertung für Release 4.2

7.4.3.2.1 Identifikation von Fehlermustern mit Attribute Focusing

Schritt 1: Ein- und zweidimensionale Auswertungen durchführen

Die Ergebnisse der ein- und zweidimensionalen Auswertungen sind für das Release 4.2 vollständig in Anhang C.7 aufgeführt.

Schritt 2: Reihenfolge der Auswertungen ermitteln

Werden die Auswertungen nach ihrer potenziellen Relevanz absteigend sortiert, ergibt sich die in Tabelle 7-19 dargestellte Reihenfolge der ersten 20 Auswertungstypen.

[751] Vgl. Kapitel 7.4.2.1.2.

[752] Vgl. Kapitel 7.4.2.4.

[753] Vgl. zum Schritt 3 der Fehlerauswertung Kapitel 6.5.

Nr.	Auswertungstyp
1	Auswirkungen
2	QS-Aktivität
3	Gegenstand der Änderung
4	Komponente
5	Art der Änderung
6	Gegenstand der Änderung x Komponente
7	QS-Kriterium
8	Gegenstand der Änderung x Art der Änderung
9	Art der Änderung x Komponente
10	Phase
11	QS-Kriterium x Komponente
12	Komponente x Auswirkung
13	Gegenstand der Änderung x QS-Kriterium
14	Komponente x Phase
15	Art der Änderung x QS-Kriterium
16	QS-Kriterium x QS-Aktivität
17	Art der Änderung x Phase
18	Art der Änderung x QS-Aktivität
19	Komponente x QS-Aktivität
20	Gegenstand der Änderung x Auswirkung

Tabelle 7-19: Reihenfolge der Auswertungen

Schritt 3: Auffällige Zusammenhänge in den Fehlerdaten bestimmen

Nachfolgend werden die 20 Tabellen für jeden Auswertungstyp aus der Tabelle 7-19 aufgeführt und relevante Erkenntnisse erläutert. Die Berechnung der erwarteten Häufigkeit bei ein- und zweidimensionalen Auswertungen wird in Kapitel 5.1.3.4 dargestellt. Es werden ausschließlich Auswertungen aufgeführt, die eine potenzielle Relevanz von mindestens 2% haben.

Auswirkung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
bekannte Softwareanforderung	79%	8%	71%	71%
Installierbarkeit	0%	8%	-8%	8%
Analysierbarkeit von Fehlverhalten	0%	8%	-8%	8%
Standards für Softwareprodukte	0%	8%	-8%	8%
Benutzerdokumentation in der Software	0%	8%	-8%	8%
Wartbarkeit	0%	8%	-8%	8%
Barrierefreiheit	0%	8%	-8%	8%

Tabelle 7-20: Eindimensionale Auswertungen für das Attribut „Auswirkung“

Erkenntnisse aus den Auswertungen in der Tabelle 7-20:

- Verhältnismäßig viele Fehler haben die Auswirkung „bekannte Softwareanforderung“, d. h. die Umsetzung einer bekannten Softwareanforderung entspricht nicht der Erwartung der Kunden.

QS-Aktivität	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Abnahmetest	73%	25%	48%	48%
Integrationstest	4%	25%	-21%	21%
Systemtest	11%	25%	-14%	14%
Update-Test	12%	25%	-13%	13%

Tabelle 7-21: Eindimensionale Auswertungen für das Attributs „QS-Aktivität“

Erkenntnisse aus den Auswertungen in der Tabelle 7-21:

- Verhältnismäßig viele Fehler werden in der QS-Aktivität Abnahmetest gefunden.

Gegenstand der Änderung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Zuweisung/Initialisierung	35%	9%	26%	26%
Entwicklerdokumentation	0%	9%	-9%	9%
Timing/Serialisierung	0%	9%	-9%	9%
Beziehung	0%	9%	-9%	9%
Algorithmus	17%	9%	8%	8%
interne Schnittstelle	1%	9%	-8%	8%
Bedingung	16%	9%	7%	7%

Tabelle 7-22: Eindimensionale Auswertungen für das Attribut „Gegenstand der Änderung“

Erkenntnisse aus den Auswertungen in der Tabelle 7-22:

- Es werden verhältnismäßig deutlich mehr Zuweisungsfehler gefunden als erwartet.
- Es werden verhältnismäßig mehr Algorithmusfehler gefunden als erwartet.
- Es werden verhältnismäßig mehr Bedingungsfehler gefunden als erwartet.

Komponente	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Produktdaten und Plausibilitätsbedingungen	40%	20%	20%	20%
unbekannt	1%	20%	-19%	19%
Anwendungskern	38%	20%	18%	18%
Schnittstellen zu Randsystemen	7%	20%	-13%	13%
versicherungsmathematischer Rechenkern	14%	20%	-6%	6%

Tabelle 7-23: Eindimensionale Auswertungen für das Attribut „Komponente“

Bei eindimensionalen Auswertungen mit Attribute Focusing ist ein Punkt zu beachten: der Verzicht auf historische Daten wird bei Attribut Focusing durch eine zentrale Annahme kompensiert. Die Annahme besagt, dass das Auftreten jedes Wertes eines Attributs gleich wahrscheinlich ist. Im Fall des Attributs Komponente bedeutet dies, dass die Fehler auf alle Komponenten gleichverteilt sind. Hierbei wird folglich nicht berücksichtigt, wie groß eine Komponente ist oder in welchem Umfang eine Komponente geändert wurde. Dies verdeutlicht, dass ohne ein Domänenwissen über den Softwareentwicklungsprozess nicht beurteilt werden kann, ob Handlungsbedarf besteht oder nicht.

Erkenntnisse aus den Auswertungen in der Tabelle 7-23:

- Die Komponente „Produktdaten und Plausibilitätsbedingungen“ hat verhältnismäßig mehr Fehler als erwartet.
- Die Komponente „Anwendungskern“ hat verhältnismäßig mehr Fehler als erwartet.

Art der Änderung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
modifiziert	36%	17%	20%	20%
eingefügt	30%	17%	14%	14%
unbekannt	4%	17%	-13%	13%
entfernt	4%	17%	-12%	12%
unklar	12%	17%	-5%	5%
irrelevant	14%	17%	-3%	3%

Tabelle 7-24: Eindimensionale Auswertungen für das Attribut „Art der Änderung“

Erkenntnisse aus den Auswertungen in der Tabelle 7-24:

- Verhältnismäßig viele Fehler werden korrigiert, indem etwas Bestehendes modifziert wird.
- Verhältnismäßig viele Fehler werden korrigiert, indem etwas Fehlendes eingefügt wird.

Gegenstand der Änderung	Komponente	Gegenstand der Änderung	Komponente	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Zuweisung/Initialisierung	Produktdaten und Plausibilitäts-bedingungen	35%	40%	31%	14%	17%	17%
irrelevant	versicherungsmathematischer Rechenkern	14%	14%	14%	2%	12%	12%
Zuweisung/Initialisierung	Anwendungskern	35%	38%	4%	13%	-10%	10%
Algorithmus	Anwendungskern	17%	38%	15%	6%	8%	8%
Algorithmus	Produktdaten und Plausibilitätsbedingungen	17%	40%	0%	7%	-7%	7%
irrelevant	Produktdaten und Plausibilitätsbedingungen	14%	40%	0%	6%	-6%	6%
irrelevant	Anwendungskern	14%	38%	0%	5%	-5%	5%

Tabelle 7-25: Zweidimensionale Auswertungen: Gegenstand der Änderung x Komponente

Erkenntnisse aus den Auswertungen in der Tabelle 7-25:

- Die Komponente „Produktdaten und Plausibilitätsbedingungen“ enthält verhältnismäßig mehr Zuweisungsfehler als erwartet.
- Die Komponente „Anwendungskern“ enthält verhältnismäßig mehr Algorithmusfehler als erwartet.

QS-Kriterium	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Abdeckung / Sequenz	18%	33%	-16%	16%
unbekannt	46%	33%	12%	12%
Variation / Abdeckung / Sequenz	36%	33%	3%	3%

Tabelle 7-26: Eindimensionale Auswertungen für das Attribut „QS-Kriterium“

Erkenntnisse aus den Auswertungen in der Tabelle 7-26:

- Die QS-Kriterien-Gruppe „Variation / Abdeckung / Sequenz“ deckt verhältnismäßig mehr Fehler auf als erwartet.

Gegenstand der Änderung	Art der Änderung	Gegenstand der Änderung	Art der Änderung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
irrelevant	irrelevant	14%	14%	14%	2%	12%	12%
Zuweisung/ Initialisierung	modifiziert	35%	36%	23%	13%	10%	10%
unklar	unklar	11%	12%	10%	1%	9%	9%
irrelevant	modifiziert	14%	36%	0%	5%	-5%	5%
Bedingung	eingefügt	16%	30%	10%	5%	5%	5%
Zuweisung/ Initialisierung	irrelevant	35%	14%	0%	5%	-5%	5%
irrelevant	eingefügt	14%	30%	0%	4%	-4%	4%

Tabelle 7-27: Zweidimensionale Auswertungen: Gegenstand der Änderung x Art der Änderung

Erkenntnisse aus den Auswertungen in der Tabelle 7-27:

- Zuweisungsfehler werden mehr als erwartet korrigiert, indem etwas Bestehendes modifiziert wird.
- Bedingungsfehler werden mehr als erwartet korrigiert, indem etwas Fehlendes eingefügt wird.

Art der Änderung	Komponente	Art der Änderung	Komponente	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
irrelevant	versicherungsmathematischer Rechenkern	14%	14%	14%	2%	12%	12%
modifiziert	Produktdaten und Plausibilitätsbedingungen	36%	40%	23%	15%	8%	8%
eingefügt	Anwendungskern	30%	38%	18%	12%	7%	7%
irrelevant	Produktdaten und Plausibilitätsbedingungen	14%	40%	0%	6%	-6%	6%
irrelevant	Anwendungskern	14%	38%	0%	5%	-5%	5%
modifiziert	versicherungsmathematischer Rechenkern	36%	14%	0%	5%	-5%	5%
eingefügt	versicherungsmathematischer Rechenkern	30%	14%	0%	4%	-4%	4%

Tabelle 7-28: Zweidimensionale Auswertungen: Art der Änderung x Komponente

Erkenntnisse aus den Auswertungen in der Tabelle 7-28:

- Fehler in den Produktdaten und Plausibilitätsbedingungen werden mehr als erwartet korrigiert, indem etwas Bestehendes modifiziert wird.
- Fehler im Anwendungskern werden mehr als erwartet korrigiert, indem etwas Fehlendes eingefügt wird.

Phase	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Entwicklung	40%	50%	-10%	10%
Produktion	60%	50%	10%	10%

Tabelle 7-29: Eindimensionale Auswertungen für das Attribut „Phase“

Erkenntnisse aus den Auswertungen in der Tabelle 7-29:

- In Produktion treten mehr Fehler auf als erwartet.

QS-Kriterium	Komponente	QS-Kriterium	Komponente	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Abdeckung / Sequenz	Anwendungskern	18%	38%	12%	7%	5%	5%
unbekannt	Anwendungskern	46%	38%	12%	17%	-5%	5%
unbekannt	versicherungsmathematischer Rechenkern	46%	14%	10%	6%	4%	4%
Variation / Abdeckung / Sequenz	Produktdaten und Plausibilitäts-bedingungen	36%	40%	18%	15%	4%	4%
Abdeckung / Sequenz	Produktdaten und Plausibilitäts-bedingungen	18%	40%	4%	7%	-4%	4%
Abdeckung / Sequenz	versicherungsmathematischer Rechenkern	18%	14%	0%	2%	-2%	2%
Variation / Abdeckung / Sequenz	versicherungsmathematischer Rechenkern	36%	14%	3%	5%	-2%	2%

Tabelle 7-30: Zweidimensionale Auswertungen: QS-Kriterium x Komponente

Erkenntnisse aus den Auswertungen in der Tabelle 7-30:

- Fehler in der Komponente „Anwendungskern“ werden mehr als erwartet durch die QS-Kriterien-Gruppe „Abdeckung/Sequenz“ aufgedeckt.
- Fehler in der Komponente „Produktdaten und Plausibilitätsbedingungen“ werden mehr als erwartet durch die QS-Kriterien-Gruppe „Variation/Abdeckung/Sequenz“ aufgedeckt.

Komponente	Auswirkung	Komponente	Auswirkung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Anwendungskern	unbekannte Software-anforderung	38%	12%	9%	5%	5%	5%
Anwendungskern	bekannte Software-anforderung	38%	79%	25%	30%	-5%	5%
Produktdaten und Plausibilitätsbedingungen	unbekannte Software-anforderung	40%	12%	1%	5%	-4%	4%
Produktdaten und Plausibilitätsbedingungen	bekannte Software-anforderung	40%	79%	35%	31%	3%	3%
versicherungsmathematischer Rechenkern	bekannte Software-anforderung	14%	79%	13%	11%	2%	2%
versicherungsmathematischer Rechenkern	unbekannte Software-anforderung	14%	12%	0%	2%	-2%	2%
Schnittstellen zu Randsystemen	unbekannte Softwareanforderung	7%	12%	2%	1%	1%	1%

Tabelle 7-31: Zweidimensionale Auswertungen: Komponente x Auswirkung

Erkenntnisse aus den Auswertungen in der Tabelle 7-31:

- Fehler in der Komponente „Anwendungskern“ haben mehr als erwartet die Auswirkung „unbekannte Softwareanforderung“.
- Fehler in der Komponente „Produktdaten und Plausibilitätsbedingungen“ haben mehr als erwartet die Auswirkung „bekannte Softwareanforderung“.

Gegenstand der Änderung	QS-Aktivität	Gegenstand der Änderung	QS-Aktivität	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Bedingung	Update-Test	18%	12%	5%	2%	3%	3%
irrelevant	Systemtest	10%	11%	4%	1%	3%	3%
Zuweisung/Initialisierung	Systemtest	44%	11%	2%	5%	-3%	3%
Zuweisung/Initialisierung	Abnahmetest	44%	73%	34%	32%	2%	2%
Bedingung	Systemtest	18%	11%	0%	2%	-2%	2%
irrelevant	Abnahmetest	10%	73%	5%	7%	-2%	2%
Algorithmus	Update-Test	13%	12%	0%	2%	-2%	2%

Tabelle 7-32: Zweidimensionale Auswertungen: Gegenstand der Änderung x QS-Aktivität

Erkenntnisse aus den Auswertungen in der Tabelle 7-32:

- Bedingungsfehler werden mehr als erwartet im Update-Test gefunden.
- Zuweisungsfehler werden mehr als erwartet im Abnahmetest gefunden.

Komponente	Phase	Komponente	Phase	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Schnittstellen zu Randsystemen	Produktion	7%	60%	0%	4%	-4%	4%
Produktdaten und Plausibilitäts-bedingungen	Entwicklung	40%	40%	19%	16%	3%	3%
Produktdaten und Plausibilitätsbedingungen	Produktion	40%	60%	21%	24%	-3%	3%
Schnittstellen zu Randsystemen	Entwicklung	7%	40%	0%	3%	-2%	2%
versicherungsmathematischer Rechenkern	Produktion	14%	60%	10%	8%	2%	2%
versicherungsmathematischer Rechenkern	Entwicklung	14%	40%	4%	6%	-2%	2%

Tabelle 7-33: Zweidimensionale Auswertungen: Komponente x Phase

Erkenntnisse aus den Auswertungen in der Tabelle 7-33:

- Fehler in den Produktdaten und Plausibilitätsbedingungen werden mehr als erwartet in der Phase „Entwicklung“ gefunden.
- Fehler im versicherungsmathematischen Rechenkern werden mehr als erwartet in der Phase „Produktion“ gefunden.

Art der Änderung	QS-Kriterium	Art der Änderung	QS-Kriterium	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
irrelevant	unbekannt	14%	46%	10%	6%	4%	4%
unbekannt	Abdeckung / Sequenz	4%	18%	3%	1%	2%	2%
modifiziert	unbekannt	36%	46%	19%	17%	2%	2%
eingefügt	unbekannt	30%	46%	12%	14%	-2%	2%
unklar	unbekannt	12%	46%	3%	5%	-2%	2%
unklar	Variation / Abdeckung / Sequenz	12%	36%	6%	4%	2%	2%
irrelevant	Abdeckung / Sequenz	14%	18%	0%	2%	-2%	2%

Tabelle 7-34: Zweidimensionale Auswertungen: Art der Änderung x QS-Kriterium

Aus den Auswertungen in der Tabelle 7-34 können keine Erkenntnisse abgeleitet werden.[754]

[754] Vgl. hierzu Kapitel 7.4.3.1.

QS-Kriterium	QS-Aktivität	QS-Kriterium	QS-Aktivität	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
unbekannt	Update-Test	41%	12%	1%	5%	-4%	4%
Variation / Abdeckung / Sequenz	Update-Test	36%	12%	8%	4%	3%	3%
unbekannt	Integrationstest	41%	4%	4%	2%	3%	3%
unbekannt	Abnahmetest	41%	73%	32%	29%	2%	2%
Variation / Abdeckung / Sequenz	Abnahmetest	36%	73%	24%	26%	-2%	2%
Variation / Abdeckung / Sequenz	Integrationstest	36%	4%	0%	2%	-2%	2%

Tabelle 7-35: Zweidimensionale Auswertungen: QS-Kriterium x QS-Aktivität

Erkenntnisse aus den Auswertungen in der Tabelle 7-35:

- Im Update-Test werden mehr Fehler als erwartet mit der QS-Kriterien-Gruppe „Variation/Abdeckung/Sequenz“ gefunden.

Art der Änderung	Phase	Art der Änderung	Phase	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
eingefügt	Entwicklung	30%	40%	16%	12%	4%	4%
eingefügt	Produktion	30%	60%	14%	18%	-4%	4%
irrelevant	Produktion	14%	60%	10%	8%	2%	2%
irrelevant	Entwicklung	14%	40%	4%	6%	-2%	2%

Tabelle 7-36: Zweidimensionale Auswertungen: Art der Änderung x Phase

Erkenntnisse aus den Auswertungen in der Tabelle 7-36:

- Entwicklungsfehler werden mehr als erwartet korrigiert, indem etwas Fehlendes eingefügt wird.

Art der Änderung	QS-Aktivität	Art der Änderung	QS-Aktivität	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
modifiziert	Systemtest	33%	11%	0%	4%	-4%	4%
irrelevant	Systemtest	10%	11%	4%	1%	3%	3%
modifiziert	Abnahmetest	33%	73%	26%	24%	2%	2%
modifiziert	Integrationstest	33%	4%	3%	1%	2%	2%
eingefügt	Update-Test	40%	12%	7%	5%	2%	2%
irrelevant	Abnahmetest	10%	73%	5%	7%	-2%	2%

Tabelle 7-37: Zweidimensionale Auswertungen: Art der Änderung x QS-Aktivität

Erkenntnisse aus den Auswertungen in der Tabelle 7-37:

- Fehler aus dem Abnahmetest werden mehr als erwartet korrigiert, indem etwas modifiziert wird.
- Fehler aus dem Integrationstest werden mehr als erwartet korrigiert, indem etwas modifiziert wird.
- Fehler aus dem Update-Test werden mehr als erwartet korrigiert, indem etwas Fehlendes eingefügt wird.

Komponente	QS-Aktivität	Komponente	QS-Aktivität	Beobachtet %	Erwartet (%)	Differenz	Relevanz
versicherungsmathematischer Rechenkern	Systemtest	10%	11%	4%	1%	3%	3%
Produktdaten und Plausibilitätsbedingungen	Systemtest	47%	11%	2%	5%	-3%	3%
Produktdaten und Plausibilitätsbedingungen	Update-Test	47%	12%	8%	6%	2%	2%
versicherungsmathematischer Rechenkern	Abnahmetest	10%	73%	5%	7%	-2%	2%

Tabelle 7-38: Zweidimensionale Auswertungen: Komponente x QS-Aktivität

Erkenntnisse aus den Auswertungen in der Tabelle 7-38:

- Fehler im versicherungsmathematischen Rechenkern werden mehr als erwartet im Systemtest gefunden.
- Fehler in den Produktdaten und Plausibilitätsbedingungen werden mehr als erwartet im Update-Test gefunden.

Gegenstand der Änderung	Auswirkung	Gegenstand der Änderung	Auswirkung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Zuweisung/Initialisierung	unbekannte Softwareanforderung	35%	12%	1%	4%	-3%	3%
Bedingung	bekannte Softwareanforderung	16%	79%	10%	13%	-3%	3%
Bedingung	unbekannte Softwareanforderung	16%	12%	4%	2%	2%	2%
Algorithmus	bekannte Softwareanforderung	17%	79%	11%	13%	-2%	2%
Zuweisung/Initialisierung	bekannte Softwareanforderung	35%	79%	29%	27%	2%	2%
irrelevant	bekannte Softwareanforderung	14%	79%	13%	11%	2%	2%
Zuweisung/Initialisierung	Zuverlässigkeit	35%	5%	4%	2%	2%	2%
irrelevant	unbekannte Softwareanforderung	14%	12%	0%	2%	-2%	2%
Algorithmus	unbekannte Softwareanforderung	17%	12%	4%	2%	2%	2%

Tabelle 7-39: Zweidimensionale Auswertungen: Gegenstand der Änderung x Auswirkung

Erkenntnisse aus den Auswertungen in der Tabelle 7-38:

- Bedingungsfehler haben mehr als erwartet die Auswirkung „unbekannte Softwareanforderung“.
- Zuweisungsfehler haben mehr als erwartet die Auswirkung „bekannte Softwareanforderung“.
- Zuweisungsfehler haben mehr als erwartet die Auswirkung „Zuverlässigkeit“.
- Algorithmusfehler haben mehr als erwartet die Auswirkung „unbekannte Softwareanforderung“.

Schritt 4: Auffällige Zusammenhänge zu Fehlermustern mit potenziellem Handlungsbedarf verdichten

Auffällige Zusammenhänge in den Fehlerdaten werden nun zu möglichen Fehlermustern verdichtet. Zunächst werden nochmals die Erkenntnisse im Zusammenhang mit den Zuweisungsfehlern aufgelistet.

Erkenntnisse zu Zuweisungsfehlern:

- Es werden verhältnismäßig viel mehr Zuweisungsfehler gefunden als erwartet.
- Zuweisungsfehler werden mehr als erwartet korrigiert, indem etwas Bestehendes modifiziert wird.
- Verhältnismäßig viele Fehler werden korrigiert, indem etwas Bestehendes modfiziert wird.
- Die Komponente „Produktdaten und Plausibilitätsbedingungen“ enthält verhältnismäßig mehr Zuweisungsfehler als erwartet.

- Die Komponente „Produktdaten und Plausibilitätsbedingungen“ hat verhältnismäßig mehr Fehler als erwartet.
- Fehler in den Produktdaten und Plausibilitätsbedingungen werden mehr als erwartet korrigiert, indem etwas Bestehendes modifiziert wird.
- Zuweisungsfehler haben mehr als erwartet die Auswirkung „bekannte Softwareanforderung“.
- Fehler in der Komponente „Produktdaten und Plausibilitätsbedingungen“ haben mehr als erwartet die Auswirkung „bekannte Softwareanforderung“.
- Zuweisungsfehler haben mehr als erwartet die Auswirkung „Zuverlässigkeit“.
- Fehler in der Komponente „Produktdaten und Plausibilitätsbedingungen“ werden mehr als erwartet durch die QS-Kriterien-Gruppe „Variation/Abdeckung/Sequenz“ aufgedeckt.
- Fehler in den Produktdaten und Plausibilitätsbedingungen werden mehr als erwartet in der Phase „Entwicklung“ gefunden.
- Fehler in den Produktdaten und Plausibilitätsbedingungen werden mehr als erwartet im Update-Test gefunden.
- Zuweisungsfehler werden mehr als erwartet im Abnahmetest gefunden.
- Fehler aus dem Abnahmetest werden mehr als erwartet korrigiert, indem etwas modifiziert wird.
- Im Update-Test werden mehr Fehler als erwartet mit der QS-Kriterien-Gruppe „Variation/Abdeckung/Sequenz“ aufgedeckt.

Aus diesen Erkenntnissen lässt sich ein mögliches Fehlermuster mit folgenden Merkmalen ableiten:[755]

Fehlermuster 1: Zuweisungsfehler
▪ Gegenstand der Änderung = Zuweisung/Initialisierung ▪ Art der Änderung: modifziert ▪ Komponente: Produktdaten und Plausibilitätsbedingungen ▪ Auswirkung: bekannte Softwareanforderung oder Zuverlässigkeit ▪ Entwicklungsfehler vom Typ Zuweisung/Initialisierung werden erst im Abnahme- und Updatetest entdeckt. ▪ QS-Kriterium = Variation/Abdeckung/Sequenz

Tabelle 7-40: Merkmale eines möglichen Fehlermusters 1

[755] Vgl. Schritt 4 in Kapitel 6.5.

Neben Erkenntnissen zu Zuweisungsfehlern gibt es einige Erkenntnisse zu Bedingungsfehlern:

- Bedingungsfehler werden mehr als erwartet korrigiert, indem etwas Fehlendes eingefügt wird.
- Verhältnismäßig viele Fehler werden korrigiert, indem etwas Fehlendes eingefügt wird.
- Bedingungsfehler haben mehr als erwartet die Auswirkung „unbekannte Softwareanforderung“.
- Bedingungsfehler werden mehr als erwartet im Update-Test gefunden.

Aus diesen Erkenntnissen lässt sich ein weiteres Fehlermuster mit folgenden Merkmalen ableiten:

Fehlermuster 2: Bedingungsfehler
▪ Gegenstand der Änderung: Bedingung ▪ Art der Änderung: eingefügt ▪ Auswirkung: unbekannte Softwareanforderung ▪ Entwicklungsfehler vom Typ Bedingung werden erst spät im Update-Test entdeckt.

Tabelle 7-41: Merkmale eines möglichen Fehlermusters 2

Folgende Erkenntnisse zu Algorithmusfehlen liegen vor:

- Die Komponente „Anwendungskern“ enthält verhältnismäßig mehr Algorithmusfehler als erwartet.
- Algorithmusfehler haben mehr als erwartet die Auswirkung „unbekannte Softwareanforderung“.
- Die Komponente „Anwendungskern“ hat verhältnismäßig mehr Fehler als erwartet.
- Fehler im Anwendungskern werden mehr als erwartet korrigiert, indem etwas Fehlendes eingefügt wird.
- Fehler aus dem Update-Test werden mehr als erwartet korrigiert, indem etwas Fehlendes eingefügt wird.
- Fehler in der Komponente „Anwendungskern“ werden mehr als erwartet durch die QS-Kriterien-Gruppe „Abdeckung/Sequenz“ aufgedeckt.
- Fehler in der Komponente „Anwendungskern“ haben mehr als erwartet die Auswirkung „unbekannte Softwareanforderung“.

Aus diesen Erkenntnissen lässt sich ein drittes Fehlermuster mit folgenden Merkmalen ableiten:

Fehlermuster 3: Algorithmusfehler
▪ Gegenstand der Änderung: Algorithmus ▪ Art der Änderung: eingefügt ▪ Komponente: Anwendungskern ▪ Auswirkung: unbekannte Softwareanforderung ▪ Entwicklungsfehler werden erst spät im Update-Test entdeckt. ▪ QS-Kriterium: Abdeckung / Sequenz

Tabelle 7-42: Merkmale eines möglichen Fehlermusters 3

Schritt 5: Konsolidierung der Fehlermuster durch manuelle Auswertungen

Der fünfte Schritt der Fehlerauswertung wird im Kapitel 7.4.3.2.2 dargestellt.

Schritt 6: Fehlermuster mit Handlungsbedarf bestimmen

Der sechste Schritt der Fehlerauswertung wird im Kapitel 7.4.3.4 dargestellt.

7.4.3.2.2 Vertiefende Untersuchung ausgewählter Fehlermuster

Die drei Fehlermuster aus Kapitel 7.4.3.2.1 werden nun durch weitere manuelle Auswertungen untersucht.

Die Abbildung 7-12 und Abbildung 7-13 zeigen die Verteilung der Entwicklungs- und Produktionsfehler gemäß dem Attribut „Gegenstand der Änderung“ für das Release 4.2. Diese Abbildungen zeigen auf, dass Zuweisungs-, Bedingungs- und Algorithmusfehler absolut und relativ einen hohen Anteil an den Entwicklungs- und Produktionsfehlern haben.

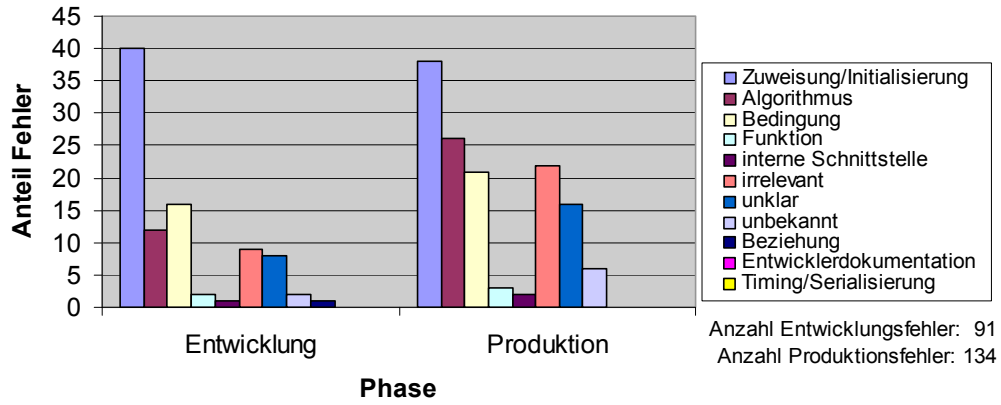


Abbildung 7-12: Verteilung der Fehler nach "Gegenstand der Änderung" (absolut)

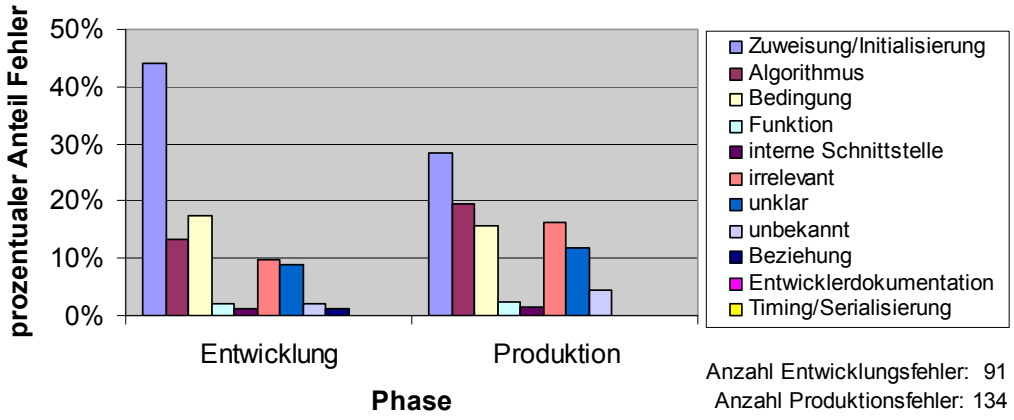


Abbildung 7-13: Verteilung der Fehler nach "Gegenstand der Änderung" (prozentual)

Zuweisungsfehler im Release 4.2

Zuweisungsfehler werden sehr spät entdeckt: entweder in den letzten beiden QS-Aktivitäten (insbesondere dem Abnahmetest) oder sogar erst während der Produktion (vgl. Abbildung 7-14 und Abbildung 7-15).

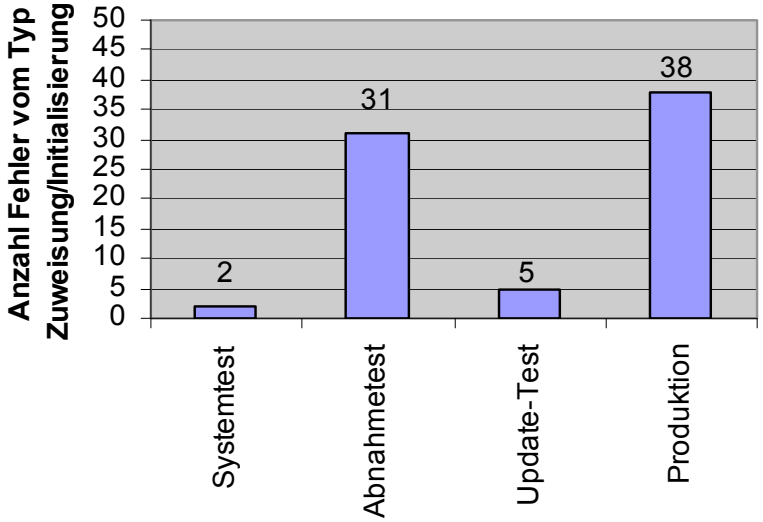


Abbildung 7-14: Verteilung der Zuweisungsfehler auf QS-Aktivitäten und die Produktion (absolut)

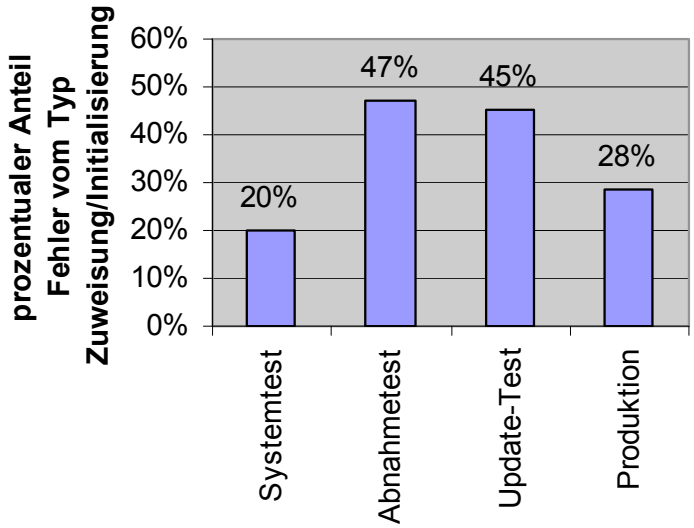


Abbildung 7-15: Verteilung der Zuweisungsfehler auf QS-Aktivitäten und die Produktion (prozentual)

65% der 78 Entwicklungs- und Produktionsfehler vom Typ Zuweisung/Initialisierung wurden behoben, indem etwas Bestehendes modifiziert wurde (vgl. Abbildung 7-16).

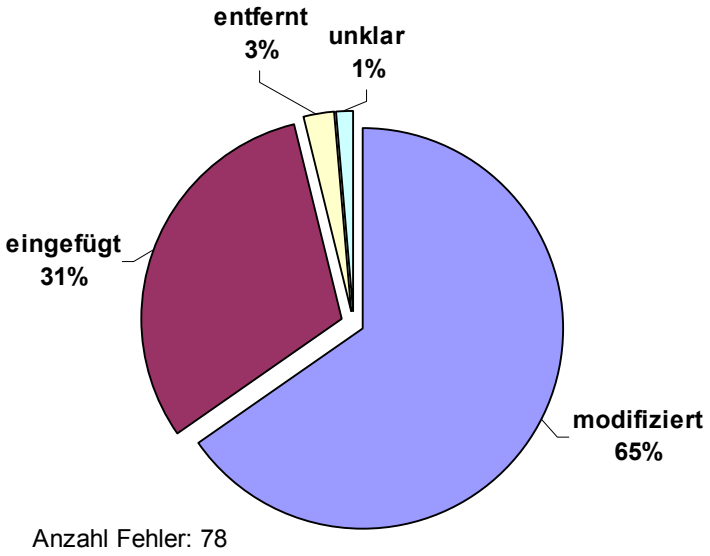


Abbildung 7-16: Verteilung der Zuweisungsfehler nach Art der Änderung

Auffällig ist zudem, dass 90% der Zuweisungsfehler der Komponente „Produktdaten und Plausibilitätsbedingungen“ zugeordnet sind (vgl. Abbildung 7-17).

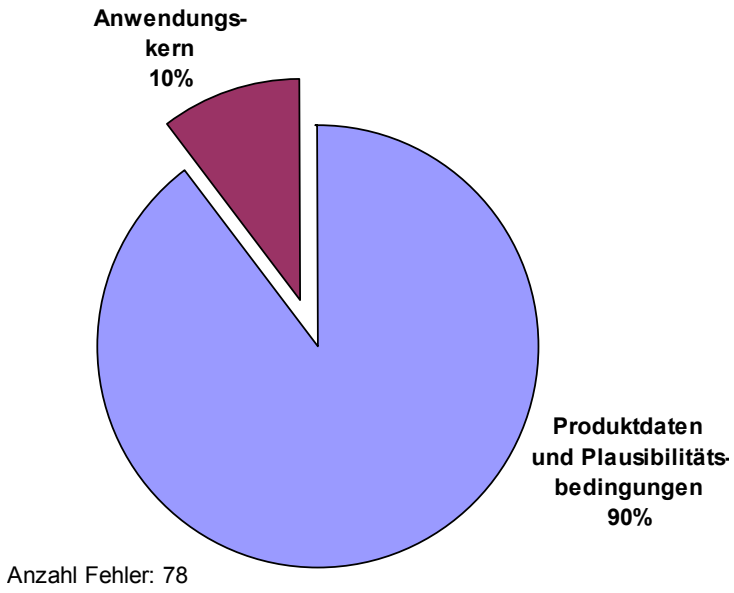


Abbildung 7-17: Verteilung der Zuweisungsfehler nach Komponenten

49% der Zuweisungsfehler in den Produktdaten und Plausibilitätsbedingungen werden ausgeliefert (vgl. Abbildung 7-18).

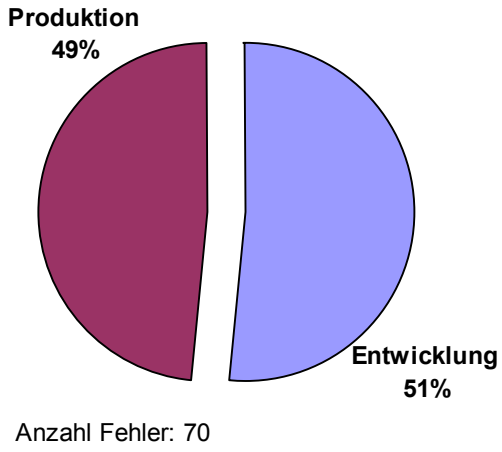


Abbildung 7-18: Verteilung der Zuweisungsfehler in den Produktdaten nach Phase

67% der Zuweisungsfehler in den Produktdaten und Plausibilitätsbedingungen werden behoben, indem etwas Bestehendes modifiziert wird (vgl. Abbildung 7-19).

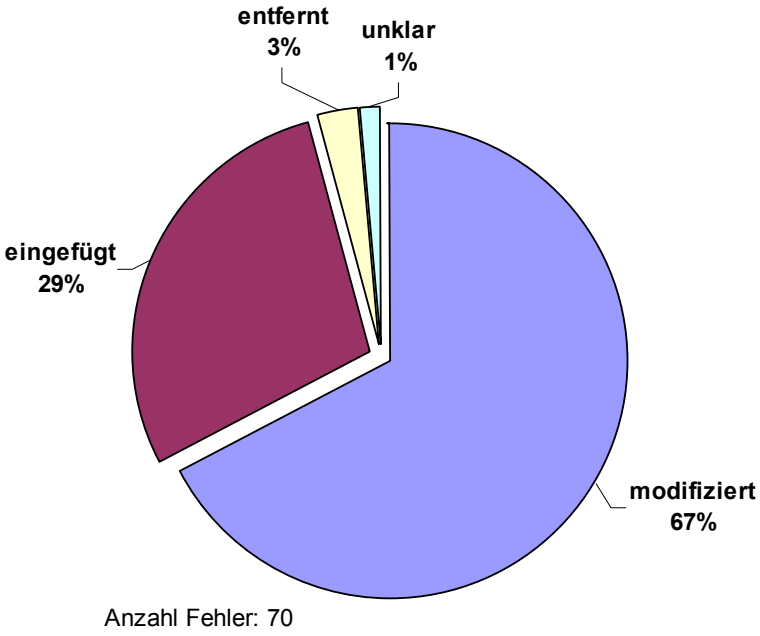


Abbildung 7-19: Verteilung der Zuweisungsfehler in den Produktdaten nach Art der Änderung

Eine Verteilung der Zuweisungsfehler in den Produktdaten und Plausibilitätsbedingungen nach QS-Kriterien zeigt keine Auffälligkeiten (vgl. Abbildung 7-20).

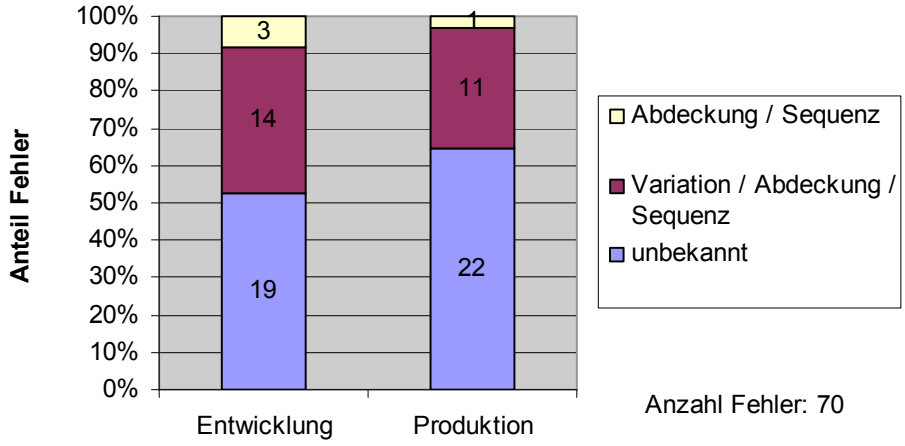


Abbildung 7-20: Verteilung der Zuweisungsfehler in den Produktdaten nach QS-Kriterium

88% der 70 Zuweisungsfehler in den Produktdaten und Plausibilitätsbedingungen weisen als Auswirkung die Ausprägung „bekannte Softwareanforderung“ aus (vgl. Abbildung 7-21).

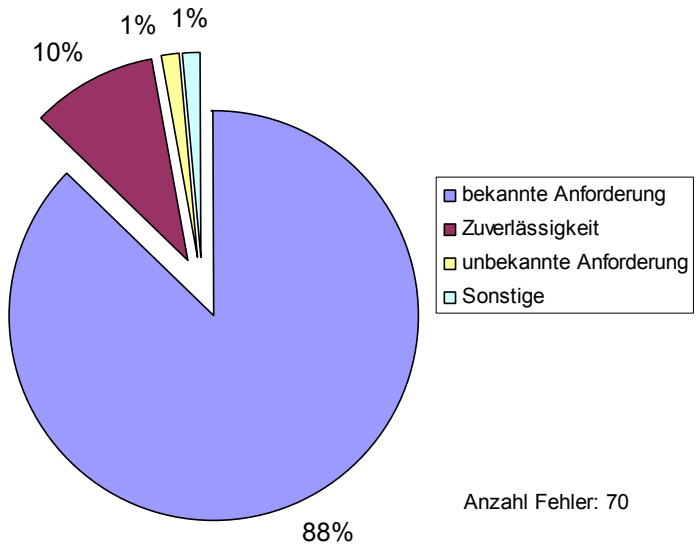


Abbildung 7-21: Verteilung der Zuweisungsfehler in den Produktdaten nach ihrer Auswirkung

Mit den zusätzlichen manuellen Auswertungen (Abbildung 7-16 bis Abbildung 7-21) kann das Fehlermuster 1 konsolidiert werden (vgl. Tabelle 7-43).

Fehlermuster 1: Zuweisungsfehler
▪ Gegenstand der Änderung = Zuweisung/Initialisierung ▪ Art der Änderung: modifziert ▪ Komponente: Produktdaten und Plausibilitätsbedingungen ▪ Auswirkung: bekannte Softwareanforderung ▪ Viele Zuweisungsfehler werden zu spät im Test gefunden oder sogar ausgeliefert.

Tabelle 7-43: Merkmale des konsoldierten Fehlermusters 1

Das konsoldierte Fehlermuster 1 mit den genannten Merkmalen umfasst insgesamt 43 Fehler und damit 19% aller Entwicklungs- und Produktionsfehler des Releases 4.2.

Bedingungsfehler im Release 4.2

Bedingungsfehler werden wie Zuweisungsfehler sehr spät entdeckt: entweder in den letzten beiden QS-Aktivitäten (insbesondere dem Abnahmetest) oder sogar zu einem großen Anteil während der Produktion (vgl. Abbildung 7-22 und Abbildung 7-23).

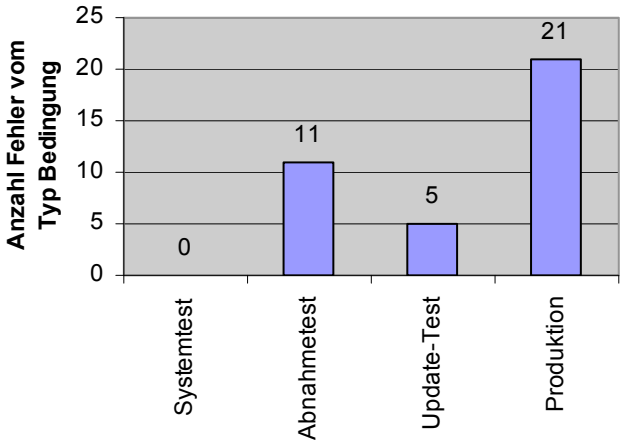


Abbildung 7-22: Verteilung der Bedingungsfehler auf QS-Aktivitäten und die Produktion (absolut)

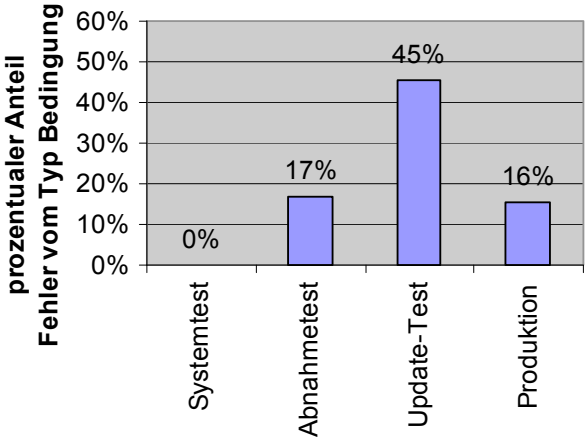


Abbildung 7-23: Verteilung der Bedingungsfehler auf QS-Aktivitäten und die Produktion (prozentual)

59% der 37 Entwicklungs- und Produktionsfehler vom Typ Bedingungen wurden behoben, indem etwas eingefügt wurde (vgl. Abbildung 7-24).

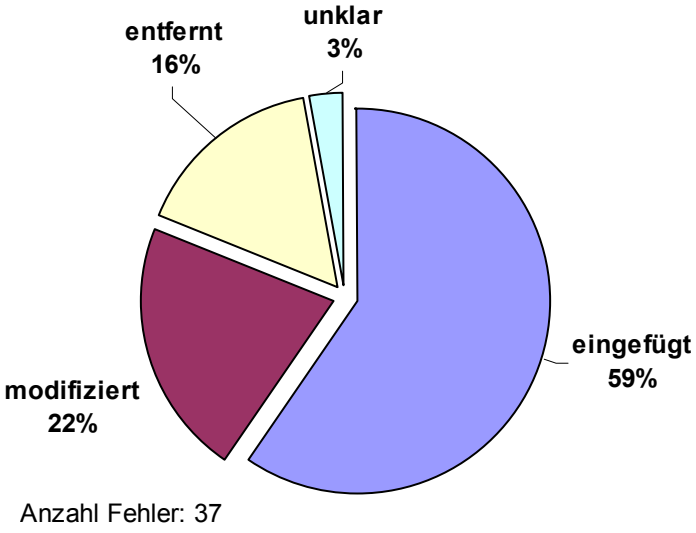


Abbildung 7-24: Verteilung der Bedingungsfehler nach Art der Änderung

Bei der Verteilung der Bedingungsfehler auf Komponenten ist kein Schwerpunkt erkennbar: 51% der Bedingungsfehler sind der Komponente „Anwendungskern“ und 38% der Komponente „Produktdaten und Plausibilitätsbedingungen“ zugeordnet (vgl. Abbildung 7-25). Deshalb werden nachfolgend die Bedingungsfehler in beiden Komponenten getrennt untersucht.

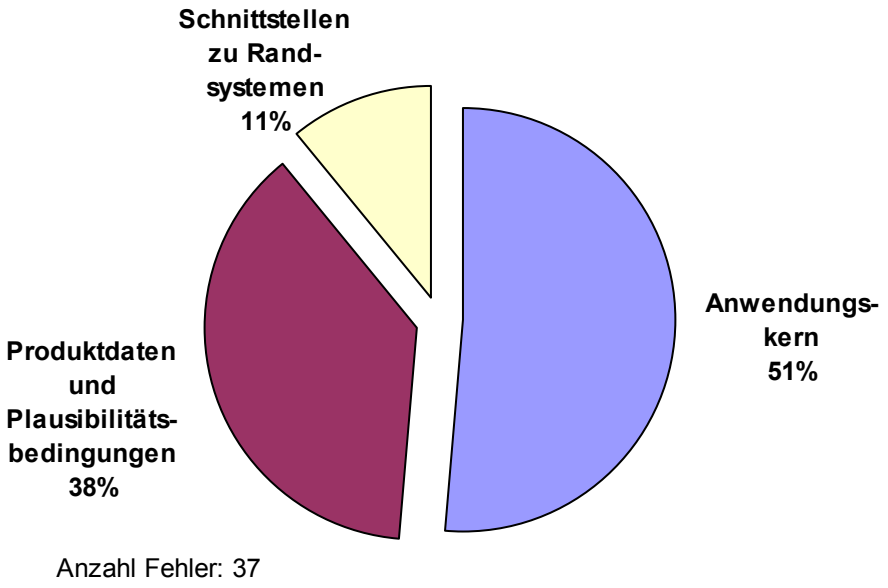


Abbildung 7-25: Verteilung der Bedingungsfehler nach Komponenten

63% der Bedingungsfehler im Anwendungskern werden nach Auslieferung der Software entdeckt (vgl. Abbildung 7-26).

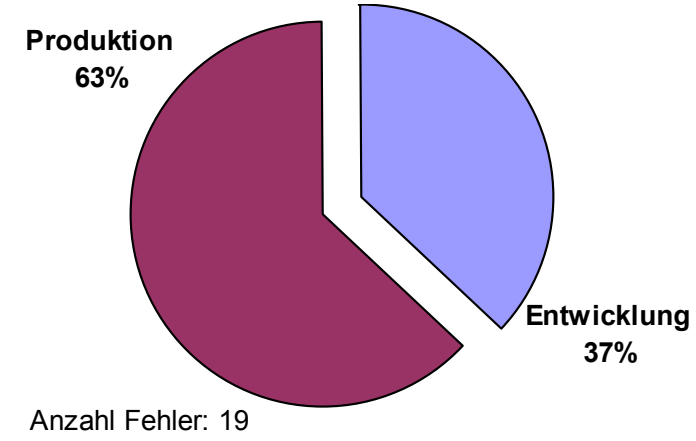


Abbildung 7-26: Verteilung der Bedingungsfehler im Anwendungskern nach Phase

84% der Bedingungsfehler im Anwendungskern werden behoben, indem etwas eingefügt wird ist (vgl. Abbildung 7-27).

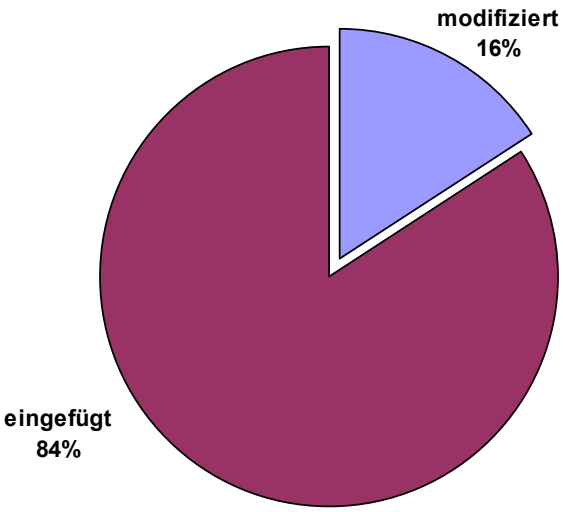


Abbildung 7-27: Verteilung der Bedingungsfehler im Anwendungskern nach Art der Änderung

Eine Verteilung der Bedingungsfehler im Anwendungskern nach QS-Kriterien zeigt keine besonderen Auffälligkeiten auf (vgl. Abbildung 7-28).

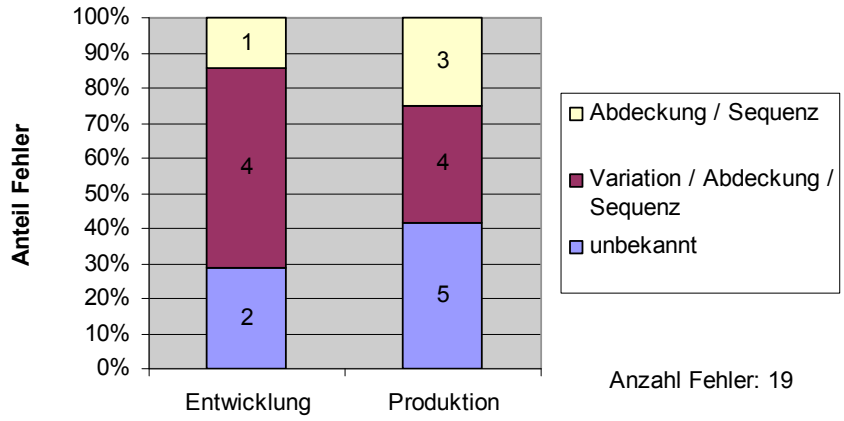


Abbildung 7-28: Verteilung der Bedingungsfehler im Anwendungskern nach QS-Kriterium

89% der 19 Bedingungsfehler im Anwendungskern weisen als Auswirkung entweder die Ausprägung „bekannte Softwareanforderung“ oder „unbekannte Softwareanforderung“ auf (vgl. Abbildung 7-29).

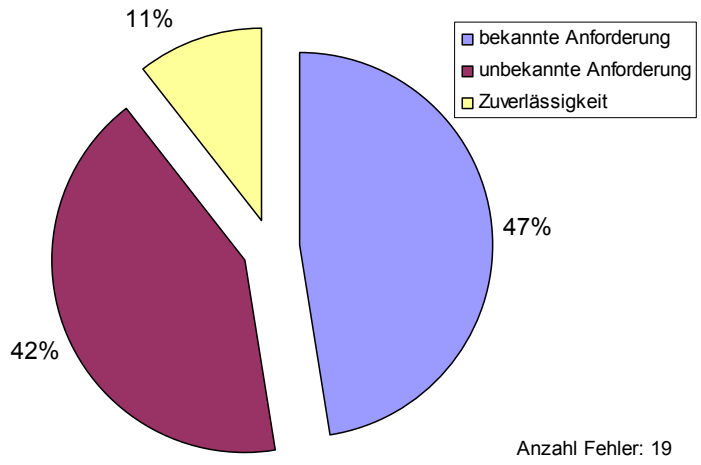


Abbildung 7-29: Verteilung der Bedingungsfehler im Anwendungskern nach ihrer Auswirkung

Mit den zusätzlichen manuellen Auswertungen kann das Fehlermuster für die Bedingungsfehler im Anwendungskern konsolidiert werden (vgl. Tabelle 7-44).

Fehlermuster 2a: Bedingungsfehler im Anwendungskern
▪ Gegenstand der Änderung = Bedingung ▪ Art der Änderung: eingefügt ▪ Komponente: Anwendungskern ▪ Viele Bedingungsfehler werden zu spät im Test gefunden oder sogar ausgeliefert.

Tabelle 7-44: Merkmale des konsoldierten Fehlermusters 2a

Das Fehlermuster 2a mit den genannten Merkmalen umfasst insgesamt 16 Fehler und damit 7% aller Entwicklungs- und Produktionsfehler des Releases 4.2.

50% der Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen werden nach Auslieferung der Software gefunden.

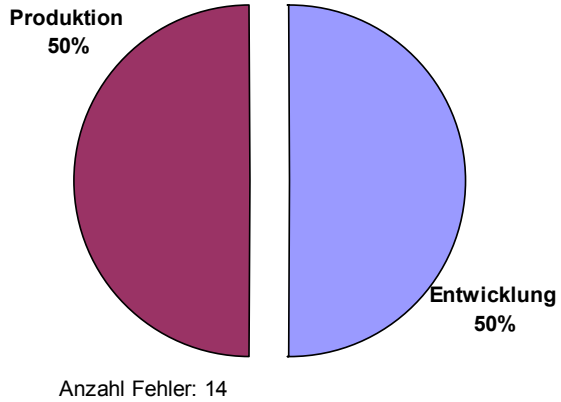


Abbildung 7-30: Verteilung der Bedingungsfehler in den Produktdaten nach Phase

Die Verteilung der Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen nach Art der Änderung zeigt keine Auffälligkeiten (vgl. Abbildung 7-31).

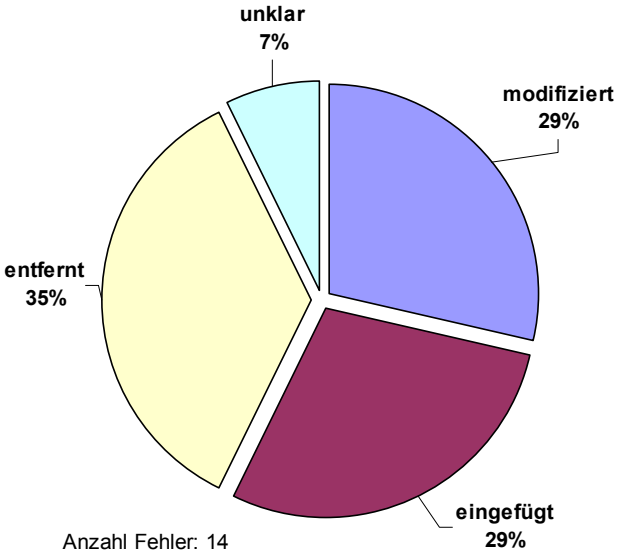


Abbildung 7-31: Verteilung der Bedingungsfehler in den Produktdaten nach Art der Änderung

Die Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen werden überwiegend durch die QS-Kriterium-Gruppe „Variation/Abdeckung/Sequenz“ gefunden (vgl. Abbildung 7-32).

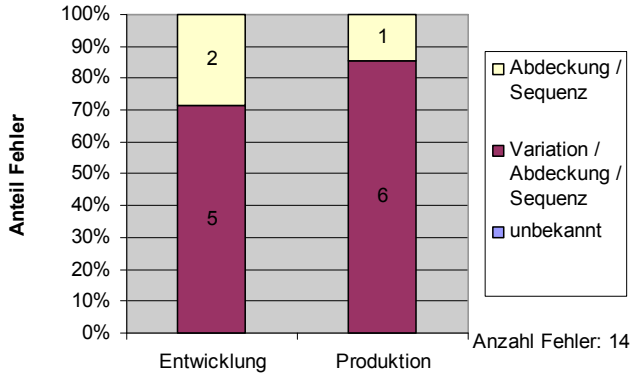


Abbildung 7-32: Verteilung der Bedingungsfehler in den Produktdaten nach QS-Kriterium

86% der Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen haben die Auswirkung „bekannte Softwareanforderung“ (vgl. Abbildung 7-33).

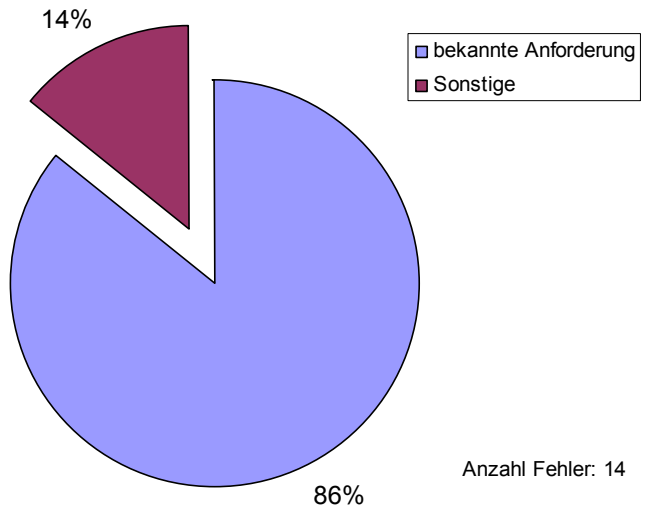


Abbildung 7-33: Verteilung der Bedingungsfehler in den Produktdaten nach ihrer Auswirkung

Mit den zusätzlichen manuellen Auswertungen kann das Fehlermuster für die Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen konkretisiert werden (vgl. Tabelle 7-43).

Fehlermuster 2b: Bedingungsfehler in Produktdaten
▪ Gegenstand der Änderung = Bedingung ▪ Komponente: Produktdaten und Plausibilitätsbedingungen ▪ QS-Kriterium: Variation/Abdeckung/Sequenz ▪ Auswirkung: bekannte Softwareanforderung ▪ Viele Bedingungsfehler werden zu spät im Test gefunden oder sogar ausgeliefert.

Tabelle 7-45: Merkmale des konsoldierten Fehlermusters 2b

Das Fehlermuster 2b mit den genannten Merkmalen umfasst insgesamt 10 Fehler und damit 4% aller Entwicklungs- und Produktionsfehler des Releases 4.2.

Algorithmusfehler im Release 4.2

Algorithmusfehler werden sehr spät entdeckt: entweder in den letzten beiden QS-Aktivitäten (insbesondere dem Abnahmetest) oder sogar zu einem großen Anteil während der Produktion (vgl. Abbildung 7-22 und Abbildung 7-23).

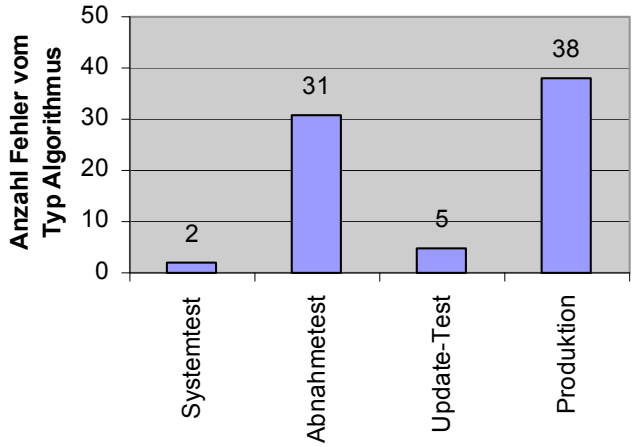


Abbildung 7-34: Verteilung der Algorithmus auf QS-Aktivitäten und die Produktion (absolut)

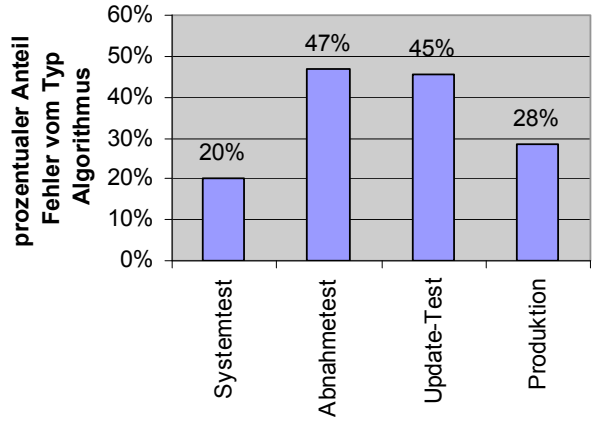


Abbildung 7-35: : Verteilung der Algorithmus auf QS-Aktivitäten und die Produktion (prozentual)

Die Verteilung der Entwicklungs- und Produktionsfehler vom Typ Algorithmus nach der Art der Änderung weist keine Auffälligkeiten auf (vgl. Abbildung 7-36).

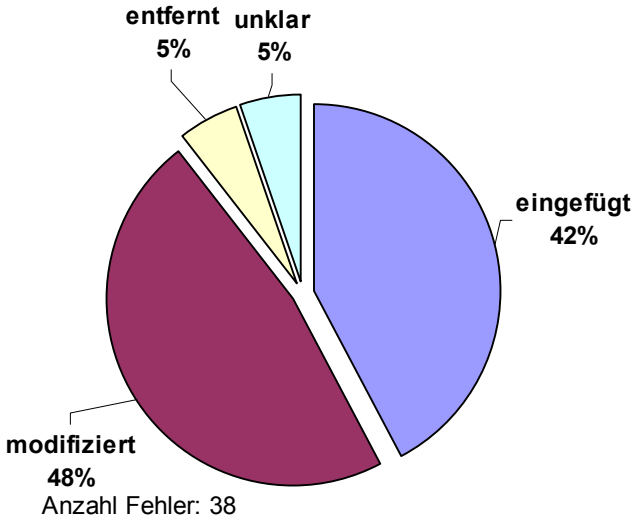


Abbildung 7-36: Verteilung der Algorithmus nach Art der Änderung

Bei der Verteilung der Algorithmusfehler auf Komponenten ist erkennbar, dass 87% der Algorithmusfehler auf die Komponente „Anwendungskern“ entfallen (vgl. Abbildung 7-37). Deshalb werden nachfolgend die Algorithmusfehler in dieser Komponente untersucht.

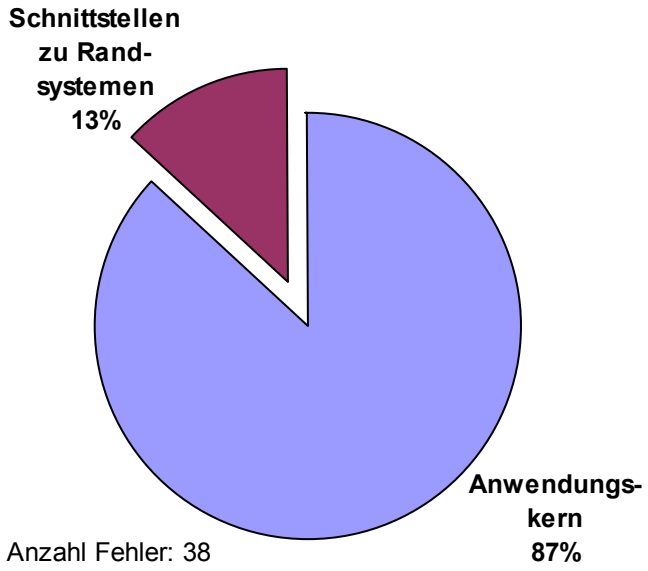


Abbildung 7-37: Verteilung der Algorithmusfehler nach Komponenten

67% der Algorithmusfehler im Anwendungskern wurden erst nach Auslieferung der Software gefunden (vgl. Abbildung 7-38).

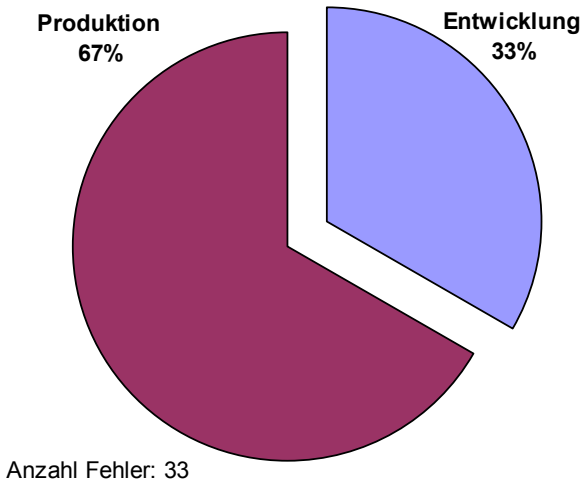


Abbildung 7-38: Verteilung der Algorithmusfehler im Anwendungskern nach Phase

Die Verteilung der Algorithmusfehler im Anwendungskern nach der Art der Änderung zeigt keine besonderen Auffälligkeiten auf (vgl. Abbildung 7-39).

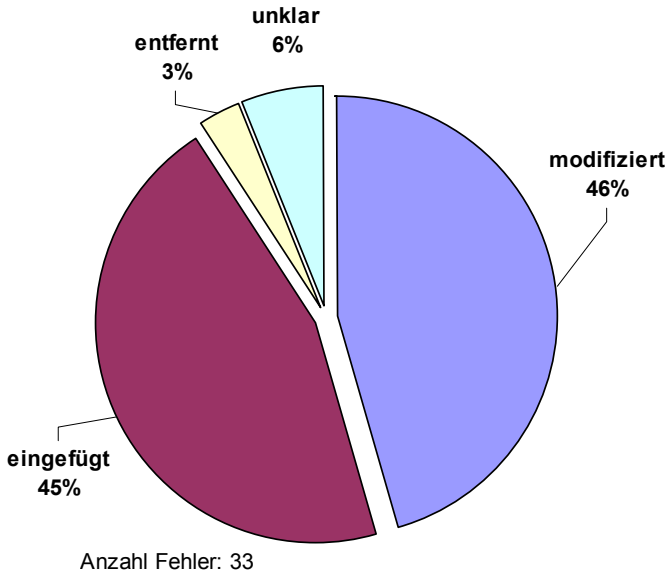


Abbildung 7-39: Verteilung der Algorithmusfehler im Anwendungskern nach Art der Änderung

Eine Verteilung der Algorithmus im Anwendungskern nach QS-Kriterien zeigt keine besonderen Auffälligkeiten auf (vgl. Abbildung 7-40).

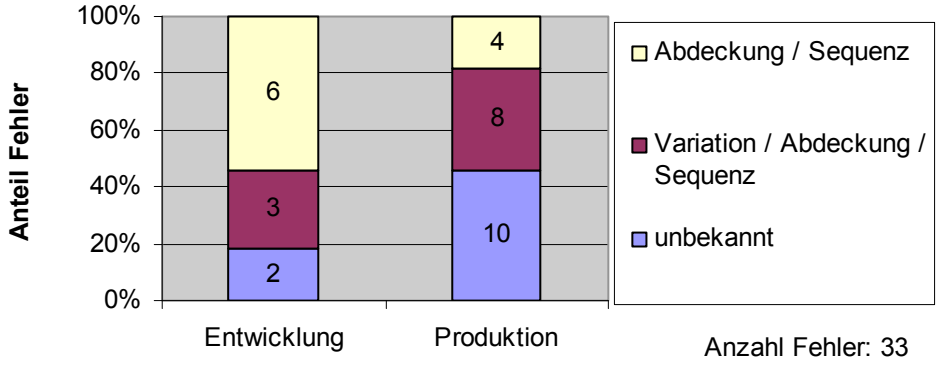


Abbildung 7-40: Verteilung der Algorithmus im Anwendungskern nach QS-Kriterium

67% der 33 Algorithmusfehler im Anwendungskern weisen als Auswirkung „bekannte Softwareanforderung“ aus (vgl. Abbildung 7-41).

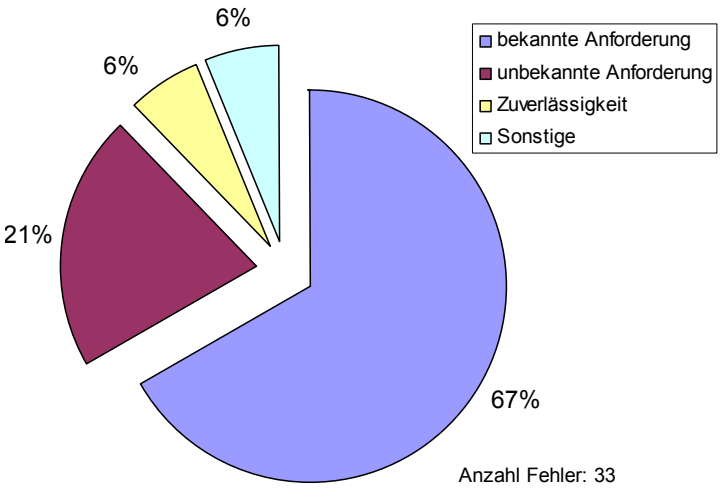


Abbildung 7-41: Verteilung der Algorithmusfehler im Anwendungskern nach ihrer Auswirkung

Mit den zusätzlichen manuellen Auswertungen kann das Fehlermuster für die Bedingungsfehler im Anwendungskern konsolidiert werden (vgl. Tabelle 7-43).

Fehlermuster 3: Algorithmusfehler
▪ Gegenstand der Änderung: Algorithmus ▪ Komponente: Anwendungskern ▪ Auswirkung: bekannte Softwareanforderung ▪ Viele Algorithmusfehler werden zu spät im Test gefunden oder sogar ausgeliefert.

Tabelle 7-46: Merkmale des konsolidierten möglichen Fehlermusters 3

Das Fehlermuster 3 mit den genannten Merkmalen umfasst insgesamt 22 Fehler und damit 10% aller Entwicklungs- und Produktionsfehler des Releases 4.2.

7.4.3.3 Vergleich der Auswertungen für die Releases 4.2 und 4.3

Nachfolgend werden die Auswertungen des Releases 4.3 mit denen des Releases 4.2 verglichen.

Zunächst muss festgehalten werden, dass die absolute Anzahl der Entwicklungs- und Produktionsfehler in Release 4.3 geringer ist als in Release 4.2.

	Release 4.2	Release 4.3
Anzahl Entwicklungsfehler	91	75
Anzahl Produktionsfehler	134	68
Anzahl Fehler insgesamt	225	143

Tabelle 7-47: Vergleich der Fehlerhäufigkeiten in den Releases 4.2 und 4.3

In beiden Releases sind die drei häufigsten Fehlertypen in der Entwicklung und in Produktion gleich: Zuweisungs-, Bedingungs- und Algorithmusfehler (vgl. Abbildung 7-42 und Abbildung 7-43). Der prozentuale Anteil der Zuweisungsfehler fällt sowohl in Entwicklung als auch in Produktion jedoch im Release 4.3 geringer aus. Zudem ist der Anteil der Algorithmusfehler bei den ausgelieferten Fehlern im Release 4.3 kleiner.

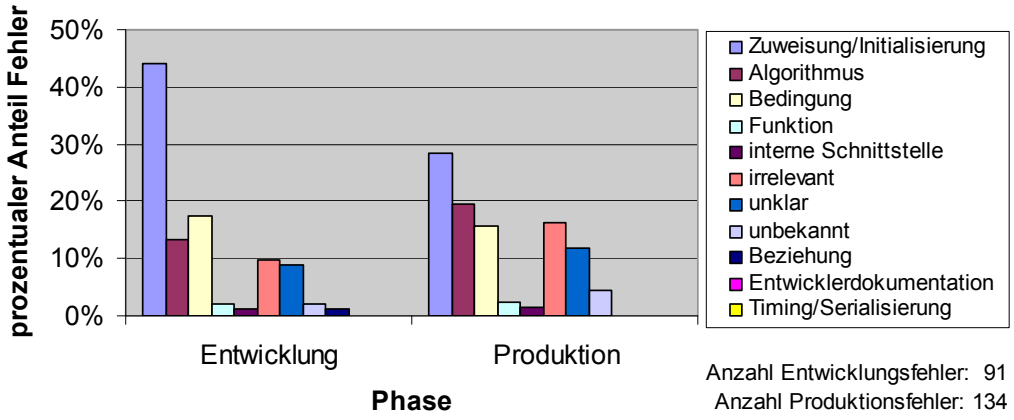


Abbildung 7-42: prozentuale Verteilung der Fehler nach "Gegenstand der Änderung" – Release 4.2

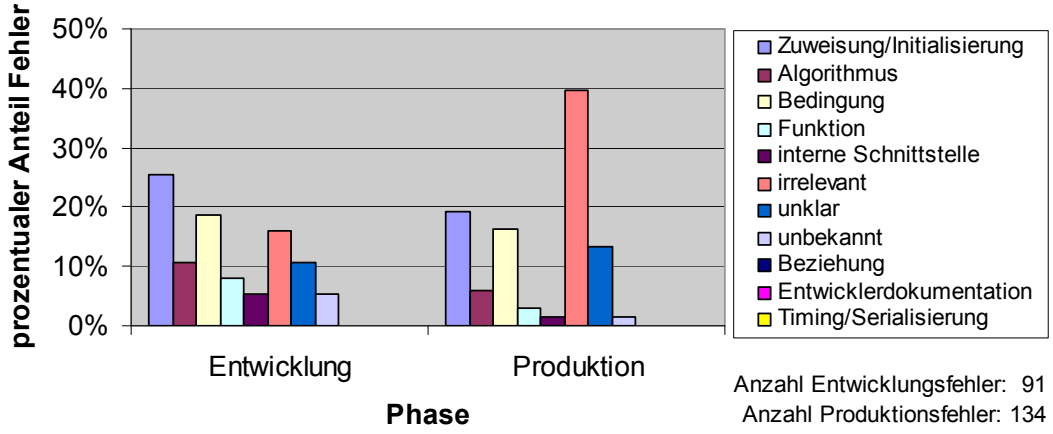


Abbildung 7-43: prozentuale Verteilung der Fehler nach "Gegenstand der Änderung" – Release 4.3

Nachfolgend wird geprüft, ob die im Release 4.2 festgestellten Fehlermuster auch im Release 4.3 vorliegen.

Zuweisungsfehler in den Releases 4.2 und 4.3

Das Fehlermuster 1 des Releases 4.2 hat folgende Merkmale (vgl. Kapitel 7.4.3.2.2):

Fehlermuster 1: Zuweisungsfehler
▪ Gegenstand der Änderung = Zuweisung/Initialisierung ▪ Art der Änderung: modifziert ▪ Komponente: Produktdaten und Plausibilitätsbedingungen ▪ Auswirkung: bekannte Softwareanforderung ▪ Viele Zuweisungsfehler werden zu spät im Test gefunden oder sogar ausgeliefert.

Tabelle 7-48: Merkmale des konsoldierten Fehlermusters 1

Zuweisungsfehler werden in beiden Releases zu spät entdeckt (vgl. Abbildung 7-44 und Abbildung 7-45). Der Anteil der Zuweisungsfehler in den QS-Aktivitäten und der Produktion ist jeweils prozentual geringer,

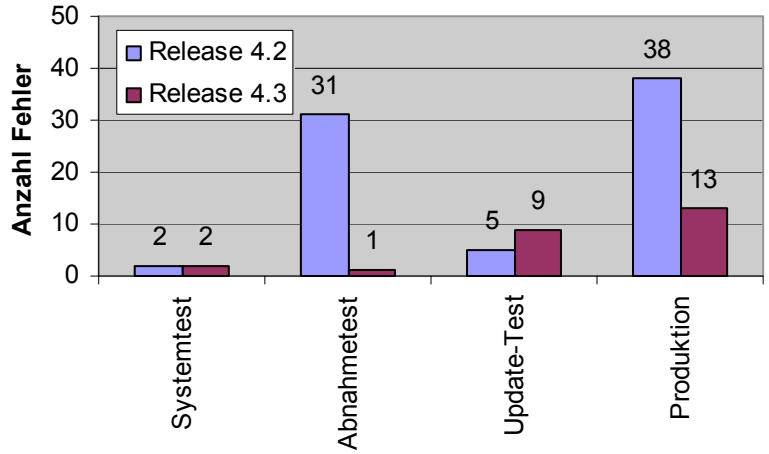


Abbildung 7-44: Verteilung der Zuweisungsfehler auf QS-Aktivitäten und die Produktion (absolut)

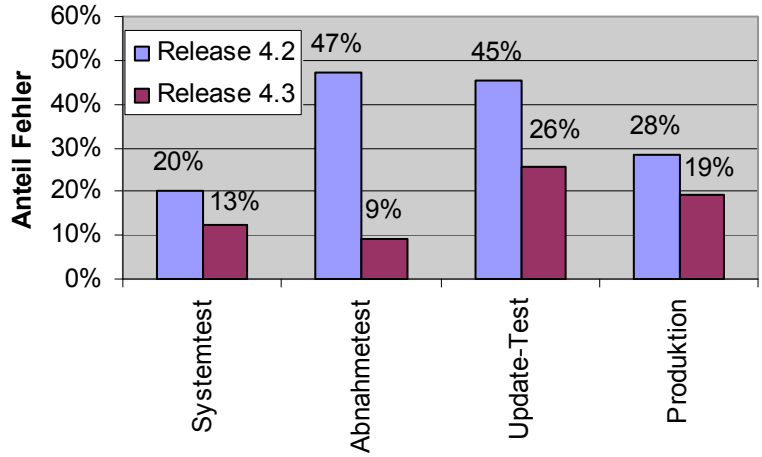


Abbildung 7-45: Verteilung der Zuweisungsfehler auf QS-Aktivitäten und die Produktion (prozentual)

In beiden Releases finden sich Zuweisungsfehler überwiegend in den Produktdaten und Plausibilitätsbedingungen (vgl. Abbildung 7-46 und Abbildung 7-47). Der prozentuale Anteil reduziert sich im Release 4.3 um 28 Prozentpunkte von 90% auf 62%.

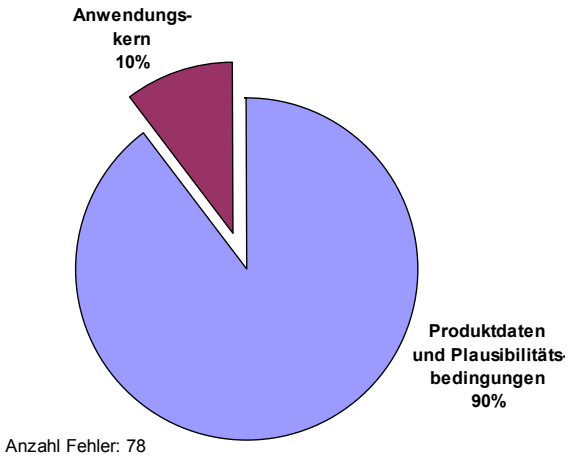


Abbildung 7-46: Verteilung der Zuweisungsfehler nach Komponenten (Release 4.2)

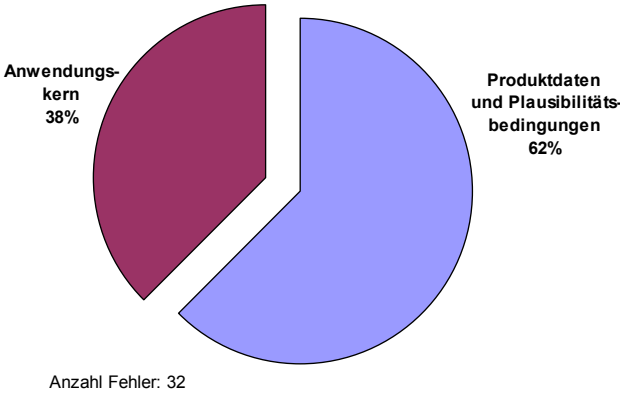


Abbildung 7-47: Verteilung der Zuweisungsfehler nach Komponenten (Release 4.3)

Die Tabelle 7-49 vergleicht ausgewählte Auswertungen für Zuweisungsfehler in der Komponente „Produktdaten und Plausbilitätsbedingungen“ in den beiden Releases. Der Vergleich zeigt auf, dass nur geringe Unterschiede vorliegen und die Fehlermuster in beiden Releases vorhanden sind.

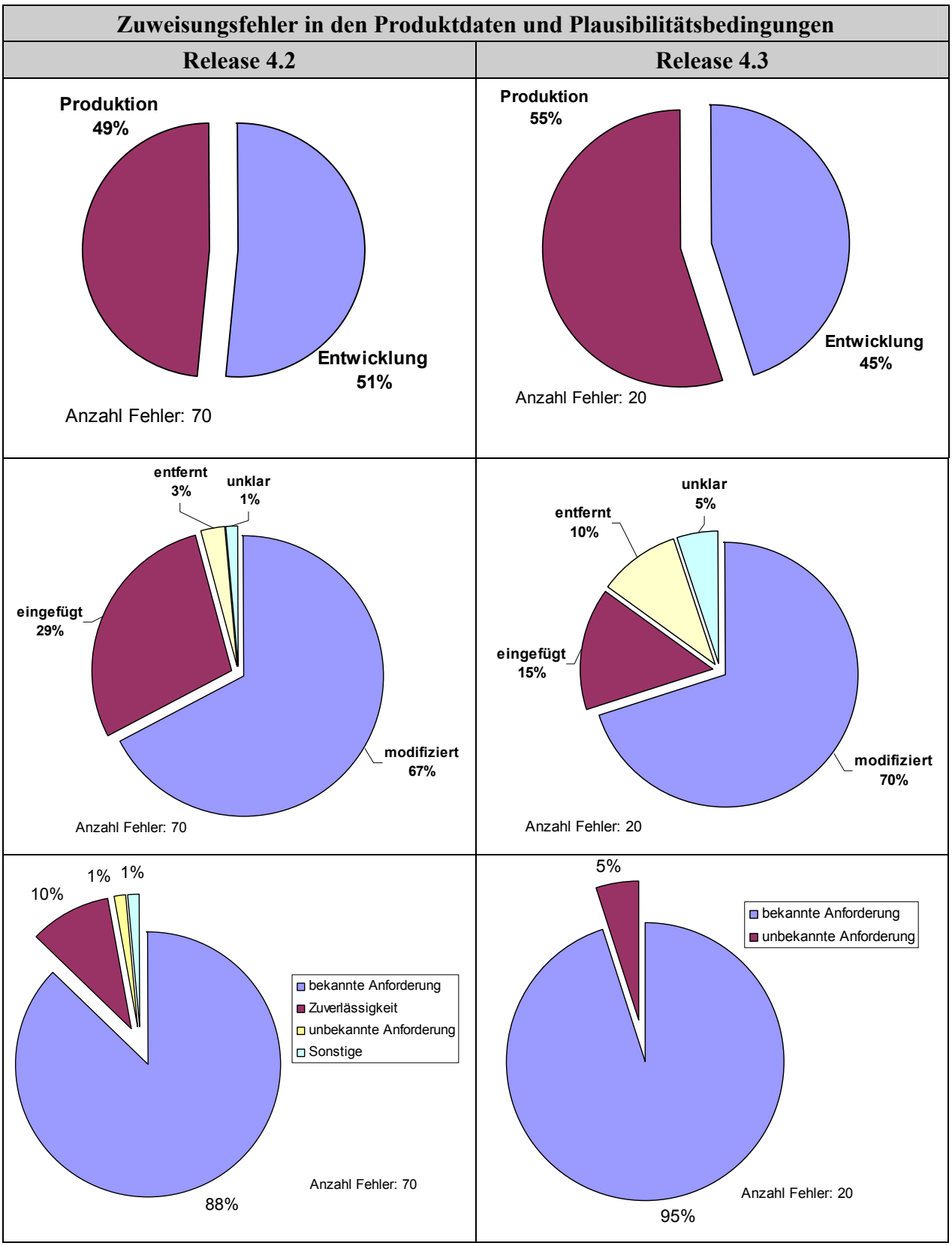


Tabelle 7-49: Zuweisungsfehler in den Produktdaten: Vergleich der Auswertungen für Phase, Art der Änderung und Auswirkung

Bedingungsfehler in den Releases 4.2 und 4.3

Bedingungsfehler werden im Release 4.3 im Vergleich zu Release 4.2 tendenziell früher gefunden (vgl. Abbildung 7-48 und Abbildung 7-49).

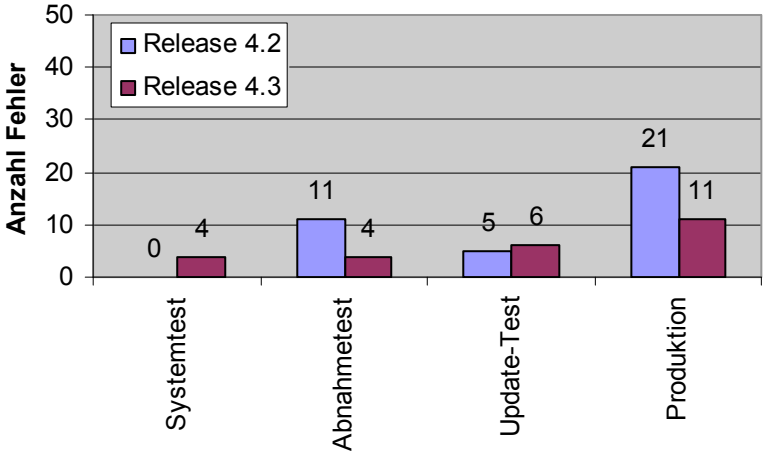


Abbildung 7-48: Verteilung der Bedingungsfehler auf QS-Aktivitäten und die Produktion (absolut)

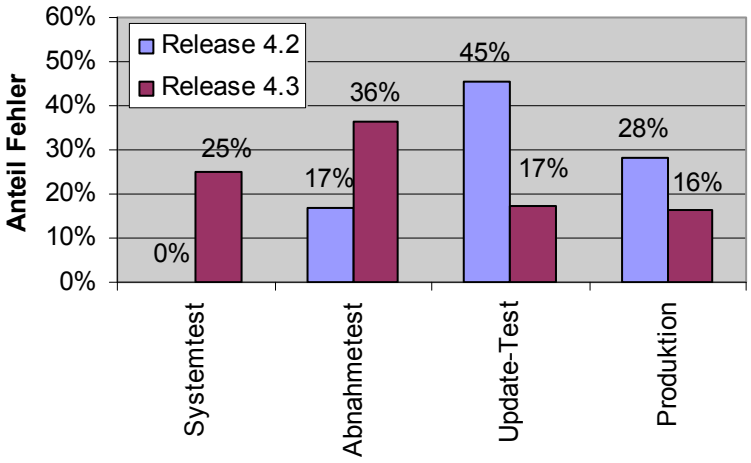


Abbildung 7-49: Verteilung der Bedingungsfehler auf QS-Aktivitäten und die Produktion (prozentual)

Die Verteilung der Bedingungsfehler auf Komponenten ist bei beiden Releases nahezu deckungsgleich (vgl. Abbildung 7-50 und Abbildung 7-51). Bedingungsfehler sind verstärkt im Anwendungskern sowie in der Komponente Produktdaten und Plausibilitätsbedingungen vorzufinden.

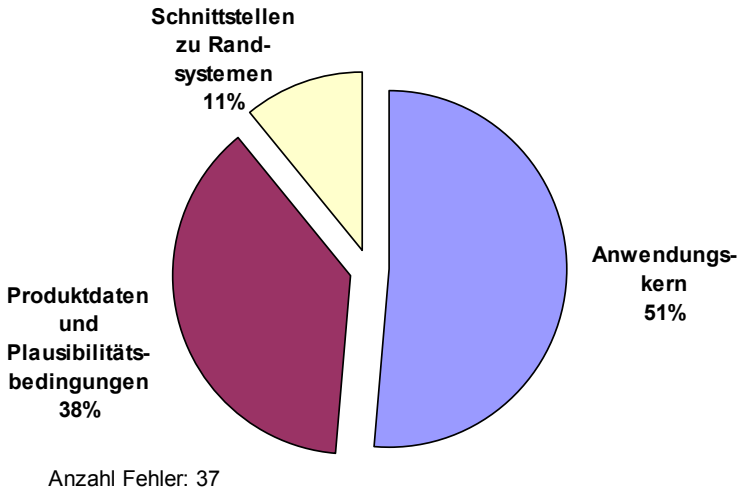


Abbildung 7-50: Verteilung der Bedingungsfehler nach Komponenten (Release 4.2)

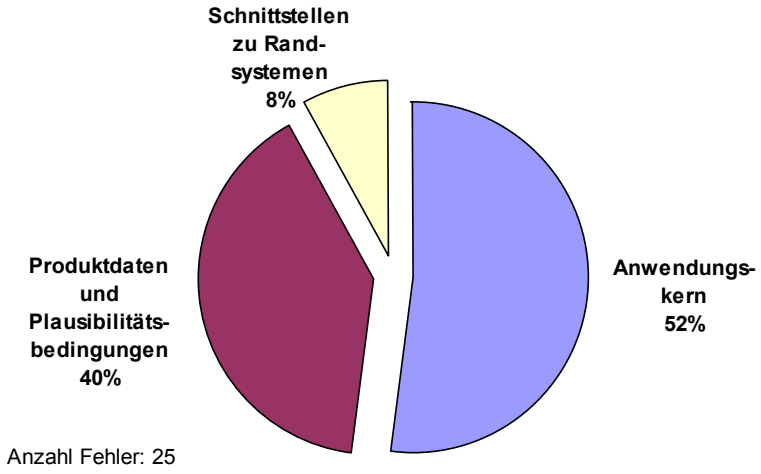


Abbildung 7-51: Verteilung der Bedingungsfehler nach Komponenten (Release 4.2)

Das Fehlermuster zu den Bedingungsfehlern im Anwendungskern des Releases 4.2 hat folgende Merkmale (vgl. Kapitel 7.4.3.2.2):

Fehlermuster 2a: Bedingungsfehler im Anwendungskern
▪ Gegenstand der Änderung = Bedingung ▪ Art der Änderung: eingefügt ▪ Komponente: Anwendungskern ▪ Viele Bedingungsfehler werden zu spät im Test gefunden oder sogar ausgeliefert.

Tabelle 7-50: Merkmale des konsoldierten Fehlermusters 2a

Die Tabelle 7-51 vergleicht ausgewählte Auswertungen für Bedingungsfehler Anwendungskern in den beiden Releases. Trotz vorhandener Unterschiede kann festgehalten werden, dass auch dieses Fehlermuster in beiden Releases vorhanden sind. So werden in beiden Releases viele Fehler erst nach Auslieferung in der Produktion entdeckt.

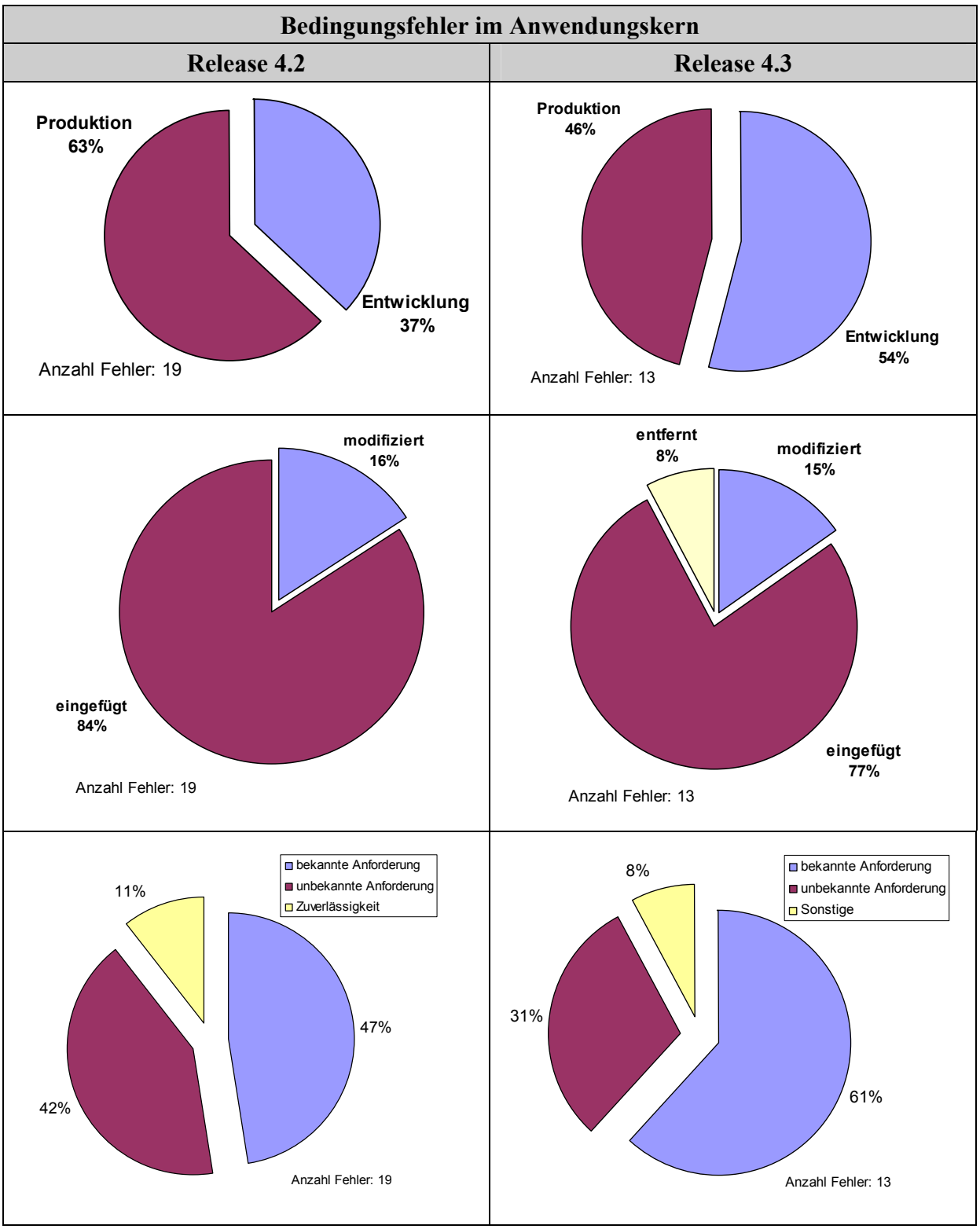


Tabelle 7-51: Bedingungsfehler in den Produktdaten: Vergleich der Auswertungen für Phase, Art der Änderung und Auswirkung

Die Merkmale des Fehlermusters zu den Bedingungsfehlern in den Produktdaten und Plausibilitätsbedingungen des Releases 4.2 hat folgende Merkmale (vgl. Kapitel 7.4.3.2.2):

Fehlermuster 2b: Bedingungsfehler in Produktdaten
▪ Gegenstand der Änderung = Bedingung ▪ Komponente: Produktdaten und Plausibilitätsbedingungen ▪ QS-Kriterium: Variation/Abdeckung/Sequenz ▪ Auswirkung: bekannte Softwareanforderung ▪ Viele Bedingungsfehler werden zu spät im Test gefunden oder sogar ausgeliefert.

Tabelle 7-52: Merkmale des konsoldierten Fehlermusters 2b

Die Tabelle 5-33 vergleicht ausgewählte Auswertungen für Bedingungsfehler in der Komponente „Produktdaten und Plausbilitätsbedingungen“ in den beiden Releases. Auch dieser Vergleich hat zum Ergebnis, dass dieses Fehlermuster für beide Releases gegenwärtig ist.

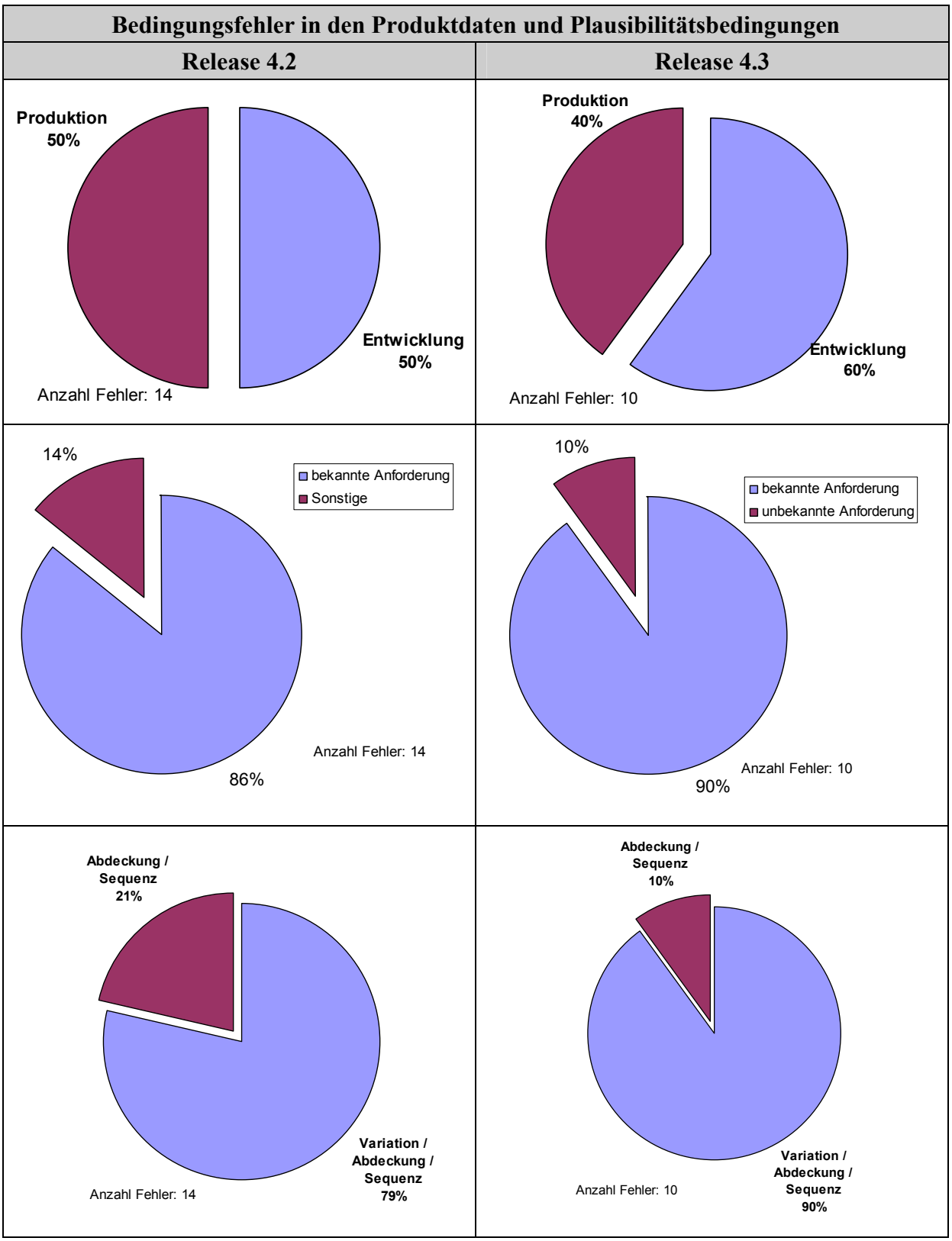


Tabelle 7-53: Bedingungsfehler in den Produktdaten: Vergleich der Auswertungen für Phase, Auswirkung und QS-Kriterium

Algorithmusfehler hatten im Release 4.2 häufig zusätzlich folgende Merkmale (vgl. Kapitel 7.4.3.2.2):

Fehlermuster 3: Algorithmusfehler
▪ Gegenstand der Änderung: Algorithmus ▪ Komponente: Anwendungskern ▪ Auswirkung: bekannte Softwareanforderung ▪ Viele Algorithmusfehler werden zu spät im Test gefunden oder sogar ausgeliefert.

Tabelle 7-54: Merkmale des konsolidierten möglichen Fehlermusters 3

Der prozentuale Anteil der Algorithmusfehler in den einzelnen QS-Aktivitäten und der Produktion ist im Vergleich zum Release 4.2 im Release 4.3 abnehmend (vgl. Abbildung 7-52). Der Trend der Algorithmusfehler im Verlauf der Zeit entwickelt sich damit im Release 4.3 deutlich positiver als im Release 4.2.

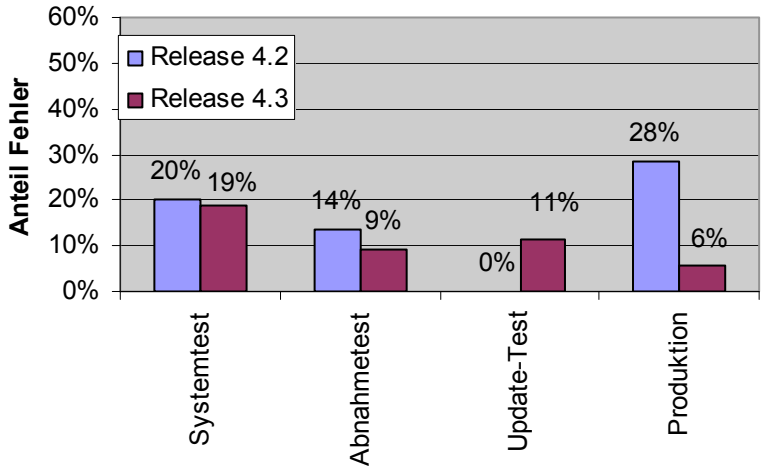


Abbildung 7-52: Verteilung der Bedingungsfehler auf QS-Aktivitäten und die Produktion (prozentual)

In beiden Releases finden sich Algorithmusfehler überwiegend im Anwendungskern (vgl.Abbildung 7-53 und Abbildung 7-54).

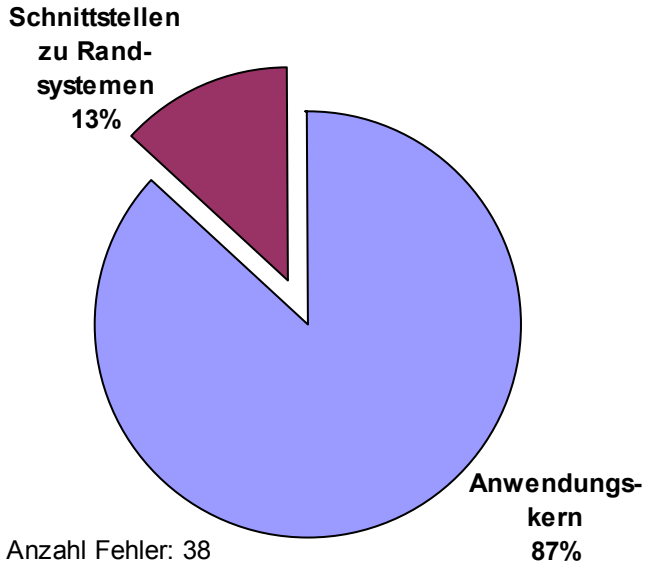


Abbildung 7-53: Verteilung der Algorithmusfehler nach Komponenten (Release 4.2)

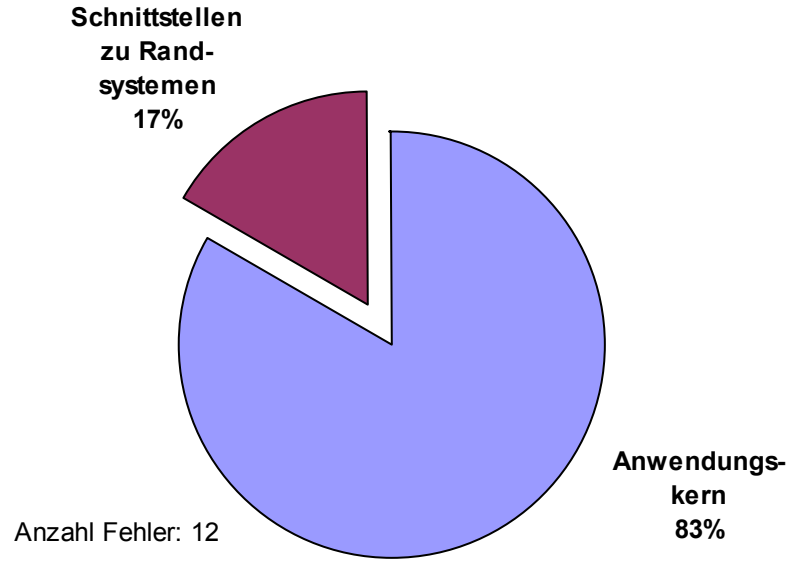


Abbildung 7-54: Verteilung der Algorithmusfehler nach Komponenten (Release 4.3)

Die Tabelle 7-55 vergleicht ausgewählte Auswertungen für Algorithmusfehler im Anwendungskern in den beiden Releases. Der Vergleich zeigt auf, dass die Daten sich stark ähneln, sofern vom Umstand abgesehen wird, dass im Release 4.3 ein größerer Anteil entsprechender Fehler vor Auslieferung entdeckt wird.

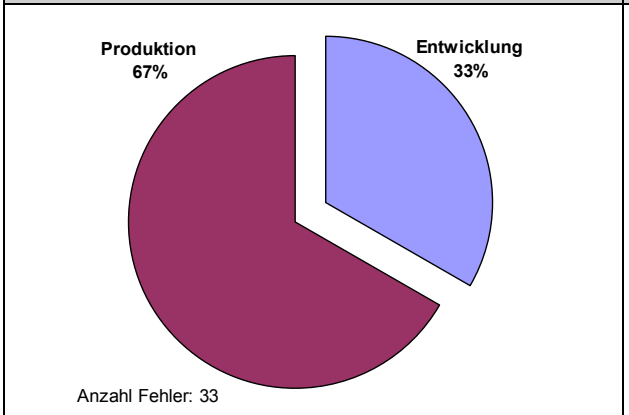
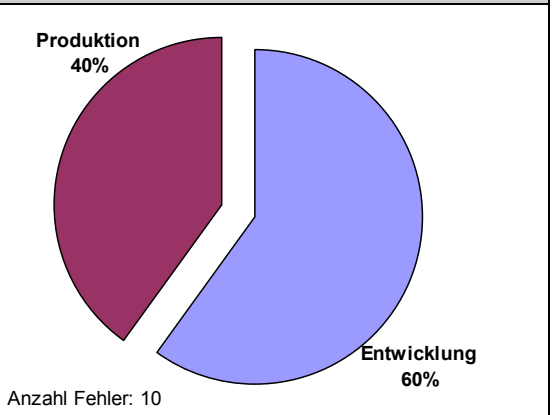
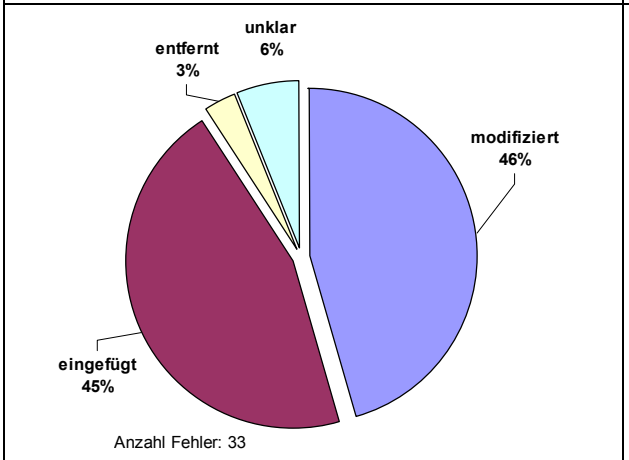
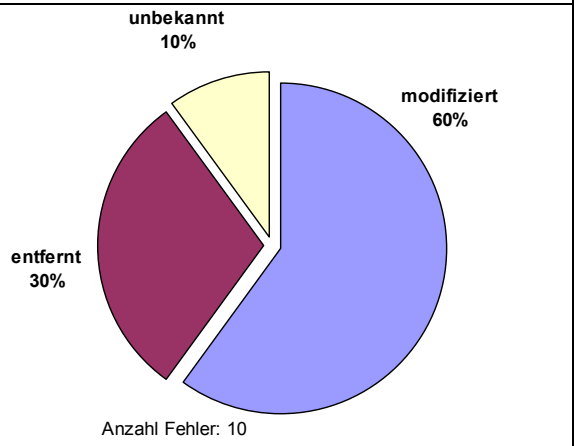
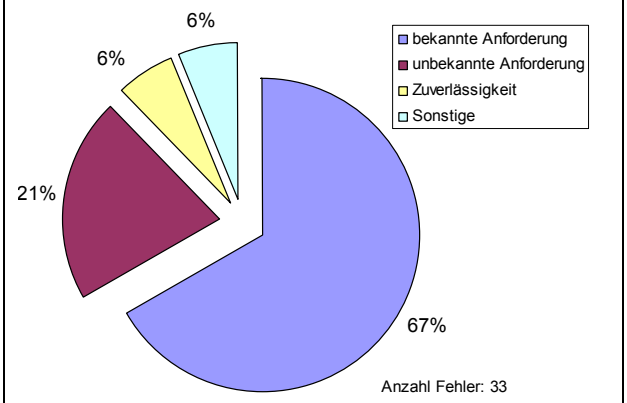
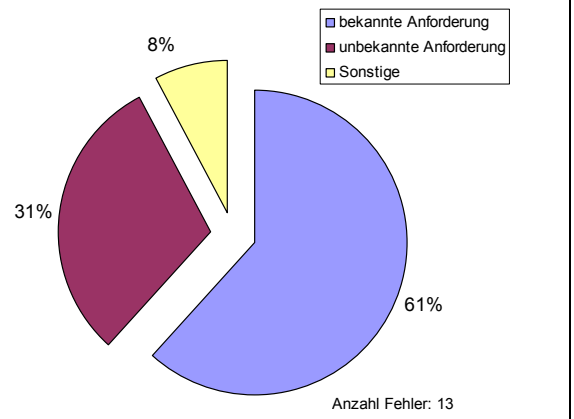
Algorithmusfehler im Anwendungskern	
Release 4.2	**Release 4.3**
	
	
	

Tabelle 7-55: Algorithmus im Anwendungskern: Vergleich der Auswertungen für Phase, Art der Änderung und Auswirkung

7.4.3.4 Ergebnisse der Arbeitssitzung zur Fehlerauswertung

Im Rahmen einer Arbeitssitzung mit Schlüsselpersonen des Softwareentwicklungsvorhabens wurden die identifizierten Fehlermuster vorgestellt und diskutiert.[756] Die Arbeitssitzung dauerte insgesamt 105 Minuten. Neben dem Autor der vorliegenden Arbeit und dem Studenten der Wirtschaftsinformatik der Universität zu Köln als Protokollant haben folgende Personen teilgenommen:[757]

- Interner Berater (IT-Power)
- Fehlermanager (IT-Power)
- Testmanager 1 (GDV)
- Testmanager 2 (IT-Power)

Ziel der Arbeitssitzung war es auffälligen Fehlermuster aus der Entwicklung und der Produktion von LV-Neu 4.2 und 4.3 vorzustellen und zu bestimmen, für welche Fehlermuster Handlungsbedarf besteht.

Zu Beginn der Arbeitssitzung gab der Autor der vorliegenden Arbeit den Teilnehmern eine Einleitung mit folgendem Inhalt:

- Kurzer Überblick über die Schritte des Fehleranalyseverfahrens (vgl. Kapitel 6.1)
- Warum ist die Kenntnis von Prozessmängeln wichtig? (vgl. Kapitel 4.2)
- Charakterisierung der untersuchten Fehlerstichprobe (vgl. Kapitel 7.4.2.2)
- Vergleich des Umfangs der Releases 4.2 und 4.3 von LV-Neu (vgl. Tabelle 7-56)

Wird der relative technische Umfang der Releases verglichen, so haben sie einen nahezu vergleichbaren Umfang.[758]

	Release 4.2	Release 4.3
Absoluter Umfang in Anzahl C Codeanweisungen	316 401	319 004
Anzahl C Dateien	2 051	2 050
Relativer Umfang in Anzahl C Codeanweisungen	6 386	6 517

Tabelle 7-56: Vergleich des Umfangs der Releases 4.2 und 4.3

[756] Vgl. Arbeitssitzung vom 23.11.05 (Ereignis 29) in Anhang C.3.

[757] Vgl. zu diesen Personen Anhang C.4.

[758] Unter dem relativen technischen Umfang eines Releases wird hier die Anzahl neuer und geänderter Codeanweisungen verstanden. Vgl. hierzu Kapitel Anhang C.5.

Als Einstieg für die Darstellung der Fehlerauswertungen wurde das Attribut Phase gewählt, da es ein zentrales Problem des Softwareentwicklungsvorhaben von LV-Neu verdeutlicht: es werden zu viele Implementierungsfehler ausgeliefert. Im Fall des Releases 4.2 sind es 60% aller gefundenen Fehler (vgl. Abbildung 7-55).

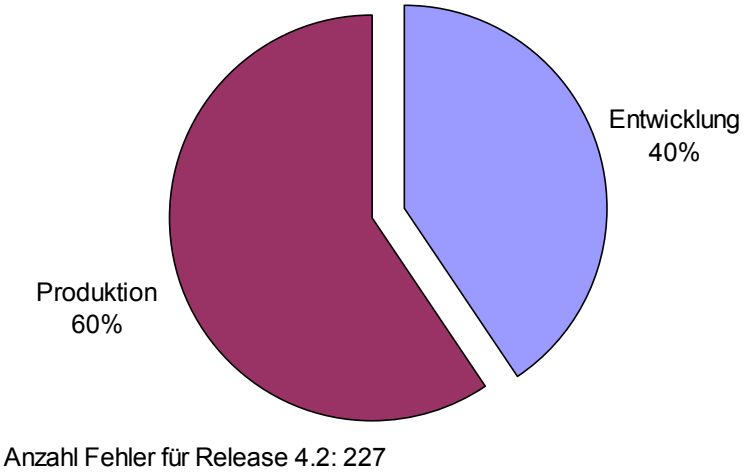


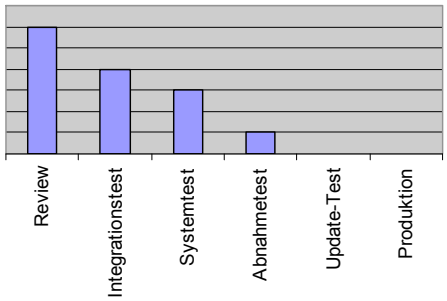
Abbildung 7-55: Verteilung der Fehler nach Phase (Release 4.2)

Anschließend wurden die Fehlermuster 1, 2a, 2b und 3 (vgl. die Kapitel 7.4.3.2.2 und 7.4.3.3) dargestellt, wobei ein Schwerpunkt auf Auswertungen für das Release 4.2 gelegt wurde. Um die Beantwortung der Frage zu erleichtern, ob ein Fehlermuster unbefriedigend ist und Handlungsbedarf aufweist, wurden die relevanten Softwareentwicklungsprozesse von LV-Neu beleuchtet und das Attribut „Gegenstand der Änderung“ für die Werte Zuweisung/Initialisierung, Bedingung und Algorithmus konfiguriert (vgl. Kapitel 7.4.1.2). Folglich wurden diese Fehlertypen mit Prozessen der Entwicklungsaufgaben und Qualitätssicherung verknüpft. Die Verknüpfung von Fehlertypen mit QS-Prozessen besagt, dass der relative Anteil der entdeckten Fehler vom jeweiligen Fehlertyp bei verknüpften QS-Prozessen jeweils ein lokales Maximum erreicht. Hieraus konnte eine schematische Soll-Entwicklung für den jeweiligen Fehlertypen abgeleitet werden. Dieser Soll-Entwicklung wurde dann jeweils die Ist-Entwicklung entgegengesetzt (vgl. Tabelle 7-57).[759]

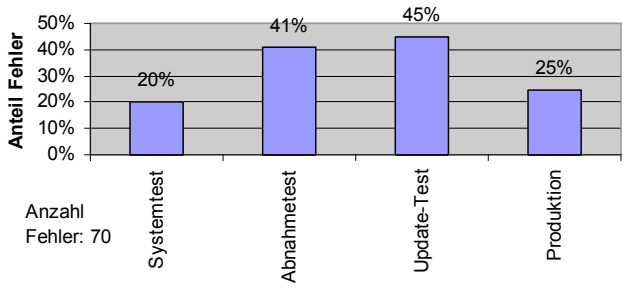
[759] Zur Bestimmung der Soll-Entwicklung vgl. Kapitel 5.1.3.5.

Zuweisungsfehler in den Produktdaten und Plausibilitätsbedingungen

Soll Entwicklung

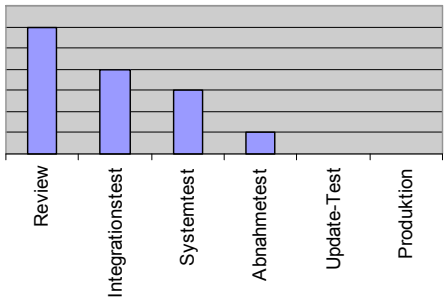


Ist-Entwicklung

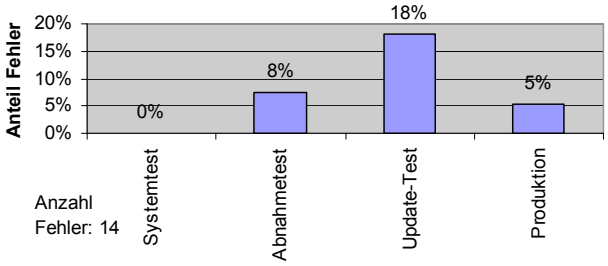


Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen

Soll Entwicklung

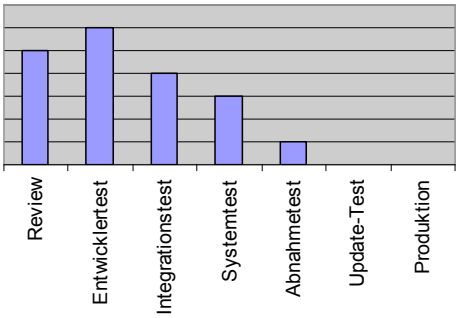


Ist-Entwicklung

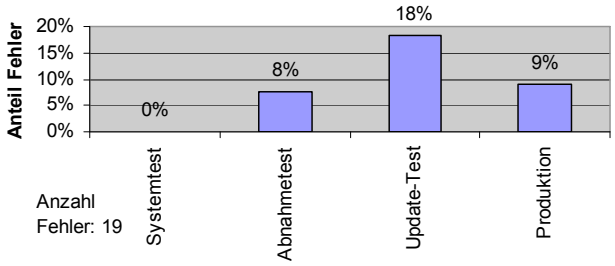


Bedingungsfehler im Anwendungskern

Soll Entwicklung

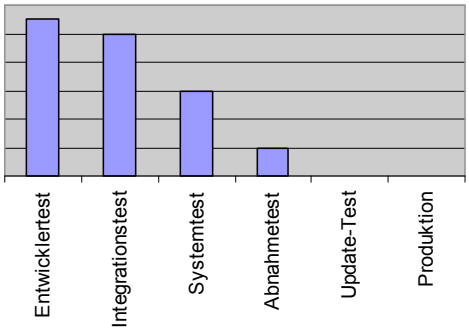


Ist-Entwicklung



Algorithmusfehler im Anwendungskern

Soll Entwicklung



Ist-Entwicklung

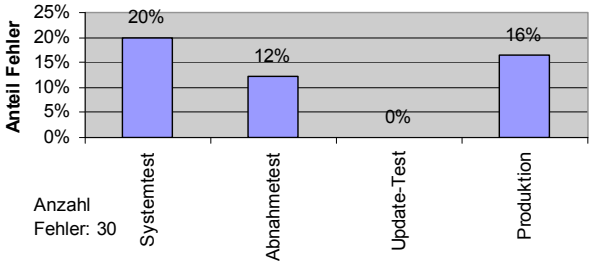


Tabelle 7-57: Soll- und Ist-Entwicklung von Fehlertypen im Release 4.2

Ergebnis der Arbeitssitzung mit den Schlüsselpersonen war:

- Es werden zu viele Implementierungsfehler nach Auslieferung des Releases entdeckt. Hier besteht großer Handlungsbedarf.

- Alle dargestellten Fehlermuster sind unbefriedigend und weisen einen Handlungsbedarf auf. Im Rahmen einer Ursachenanalyse sollen zunächst die Ursachen folgender Fehlermuster identifiziert werden.
    - Fehlermuster 1: Zuweisungsfehler in den Produktdaten und Plausibilitätsbedingungen
    - Fehlermuster 2a: Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen
    - Fehlermuster 2b: Bedingungsfehler im Anwendungskern

7.4.3.5 Beobachtungen im Rahmen der Fehlerauswertung

Sowohl die automatische Auswertung mit Attribute Focusing als auch die vertiefende manuelle Auswertungen konnten ohne Schwierigkeiten durchgeführt werden. Nachfolgend werden wesentliche Beobachtungen aus der Arbeitssitzung zur Fehlerauswertung aufgeführt.

Beobachtung 1:

Das Ergebnis der Fehlerauswertung regt die Teilnehmer an, direkt über mögliche Ursachen zu disktutieren.

Die Diskussion über Fehlermuster regte die Teilnehmer ein, frühzeitig über mögliche Ursachen zu diskutieren. Da jedoch erst nach einer vollständigen Darstellung eines Fehlermusters Rückschlüsse über die Ursachen gezogen werden können, wurden entsprechende Diskussionen auf die nächste Sitzung verlegt.

Beobachtung 2:

Die Teilnehmer der Arbeitssitzung nennen drei Typen des Erkenntnisgewinns:

- *Typ 1: Die Fehlerauswertung bestätigt Sachverhalte, die den Teilnehmern im Prinzip bekannt sind, über deren Außmaß sie aber überrascht sind.*
- *Typ 2: Die Fehlerauswertung deckt Sachverhalte auf, die den Teilnehmern zuvor nicht bekannt waren.*

Nachfolgend wird jeweils ein Beispiel für jeden Erkenntnistypen aufgeführt:

- Typ 1: Die Teilnehmer waren sich des Problems bewusst, dass viele Fehler ausgeliefert werden. Sie waren aber über das Ausmaß des Problems überrascht: im Release 4.2 wurden 60% der Fehler ausgeliefert und im Release 4.3 48%. In der Problemdatenbank werden zwar die Implementierungsfehler erfasst. Jedoch wird zu einem

Fehler nicht in einer automatisch auswertbaren Form festgehalten, ob der Fehler sich auf sich auf das Release in Entwicklung oder das Release in Produktion bezieht.

- Typ 2: Die Teilnehmer waren über die große Anzahl der Fehler in der Komponente „Produktdaten und Plausibilitätsbedingungen“ überrascht. In der Problemdatenbank erfolgte in der Regel die Zuordnung der Komponenten nicht zu einem Fehler, sondern zu einem Fehlverhalten der Software. Da ein Fehler in den Produktdaten und Plausibilitätsbedingungen häufig ein Fehlverhalten im versicherungsmathematischen Rechenkern verursacht, sind Fehler in den Produktdaten und Plausibilitätsbedingungen in der Problemdatenbank häufig als Fehler im versicherungsmathematischen Rechenkern eingetragen.

Beobachtung 3:

Durch die Fehlerauswertung werden quantitative Daten verfügbar, die die Überzeugungsarbeit in den Projekten und die Einführung von gezielten Maßnahmen unterstützen.

Die Teilnehmer weisen daraufhin, dass die Fehlerauswertung selbst in den Fällen hilfreich sein kann, in denen keine neuen Sachverhalte aufgedeckt werden. Die Verfügbarkeit von quantitativen Daten kann helfen, Diskussionen zu versachlichen.

Beobachtung 4:

Die Softwareentwicklungsprojekte LV-Neu 4.2 und 4.3 weisen weitgehend die gleichen Fehlermuster auf.

Eine wichtige Beobachtung im Rahmen der Fallstudie ist, dass trotz unterschiedlicher Anforderungen, die in dem jeweiligen Softwareentwicklungsprojekt umgesetzt werden, beide Projekte die gleichen Fehlermuster aufweisen.[760] Dies stützt die These, dass bei gleichbleibenden Rahmenbedingungen und Softwareentwicklungsprozessen vorhandene Prozessmängel systematisch Fehler mit gleichen Merkmalen verursachen.

[760] Vgl. Kapitel 7.4.3.3.

7.4.4 Ursachenanalyse

7.4.4.1 Ablauf der Ursachenanalyse

Der Ablauf der Ursachenanalyse orientiert sich an der Beschreibung des Fehleranalyseverfahrens in Kapitel 6.6.

Im Rahmen einer Arbeitssitzung mit Schlüsselpersonen des Softwareentwicklungsvorhabens wurden die Ursachen für ausgewählte Fehlermuster bestimmt und erste Gegenmaßnahmen erhoben.[761] Hierzu wurde während der Arbeitssitzung für jedes Fehlermuster ein Ursache-Wirkungsdiagramm nach Ishikawa[762] mit den Teilnehmern auf einer Pinnwand erstellt.

Die Arbeitssitzung dauerte insgesamt 180 Minuten. Neben dem Autor der vorliegenden Arbeit und dem Studenten der Wirtschaftsinformatik der Universität zu Köln als Protokollant haben folgende Personen teilgenommen:[763]

- Fehlermanager (IT-Power)
- Testmanager 1 (GDV)
- Testmanager 2 (IT-Power)
- Produktdatenverantwortlicher (GDV)

Die Arbeitssitzung wurde durch den Autor der vorliegenden Arbeit moderiert.

Aus zeitlichen Gründen konnten in der Sitzung ausschließlich die Ursachen der folgenden zwei Fehlermuster behandelt werden:

- Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen
- Bedingungsfehler im Anwendungskern

Die Zuweisungsfehler in den Produktdaten und Plausibilitätsbedingungen konnten entgegen der ursprünglichen Planung nicht berücksichtigt werden.

7.4.4.2 Ergebnisse der Ursachenanalyse

7.4.4.2.1 Fehlermuster: Bedingungsfehler im Anwendungskern

Charakterisierung Fehlermuster

Zunächst wurden anhand von Abbildungen zu ausgewählten Auswertungen die Merkmale des Fehlermusters zu Bedingungsfehler im Anwendungskern dargestellt (vgl. Tabelle 7-58). Hierbei wurden Auswertungen zum Release 4.2 und 4.3 gleichermaßen berücksichtigt.[764]

---

[761] Vgl. Arbeitssitzung vom 01.12.05 (Ereignis 32) in Anhang C.3.

[762] Vgl. Ishikawa /Quality Control/ 18-29.

[763] Vgl. zu diesen Personen Anhang C.4.

Fehlermuster: Bedingungsfehler im Anwendungskern
▪ Gegenstand der Änderung = Bedingung ▪ Art der Änderung: eingefügt ▪ Komponente: Anwendungskern ▪ Viele Bedingungsfehler werden zu spät im Test gefunden oder sogar ausgeliefert.

Tabelle 7-58: Merkmale des Fehlermusters: Bedingungsfehler im Anwendungskern

Diskussion konkreter Fehler

Neben der allgemeinen Charakterisierung dieses Fehlermusters wurden exemplarisch konkrete Beschreibungen von Fehlern mit den Teilnehmern diskutiert, um ein besseres Verständnis über das Fehlermuster zu bekommen. Hierzu wurden die Nummern alle Problemberichte der Releases 4.2 und 4.3 ausgeführt, die Fehler gemäß dem Fehlermuster beinhalteten. Dies waren im Release 4.2 16 Fehler in 16 Problemberichten und in Release 4.3 10 Fehler in 10 Problemberichten. Die Auswahl der zu diskutierenden Fehler erfolgte durch die Teilnehmer auf der Grundlage der Problemberichtsnummer ohne Kenntnis des genauen Inhalts der Fehler. Es wurden vier Fehler näher diskutiert. Alle diskutierten vier Implementierungsfehler sind eine Folge von Anforderungsfehlern: in drei Fällen fehlte eine Anforderung und in einem Fall lag eine missverständliche Anforderung war.

Ursachenanalyse

Ausgehend von der Charakterisierung des Fehlermusters und der Diskussionen einzelner Fehler wurden in der Gruppe Ursachen für entsprechende Fehler erhoben. Die Ergebnisse wurden in der Arbeitssitzung als Ursache-Wirkungsdiagramm auf einer Pinnwand festgehalten (vgl. Abbildung 7-56). Die einzelnen Ursachen, warum entsprechende Fehler entstehen oder zu spät gefunden werden, lassen sich in die folgenden vier Kategorien gruppieren:

- Überlastung der Verantwortlichen für die Fachkonzepte.
- Fachbereiche können Fachkonzepte nicht angemessen abnehmen.
- Im Fachkonzept fehlen Softwareanforderungen.
- Kommunikationsprobleme zwischen den Entwicklungsteams für den Anwendungskern, die Produktdaten und Plausibilitätsbedingungen und dem versicherungsmathematischen Rechenkern.

[764] Vgl. zu den Auswertungen und entsprechenden Abbildungen zu diesem Fehlermuster die Kapitel 7.4.3.2.2 und 7.4.3.3.

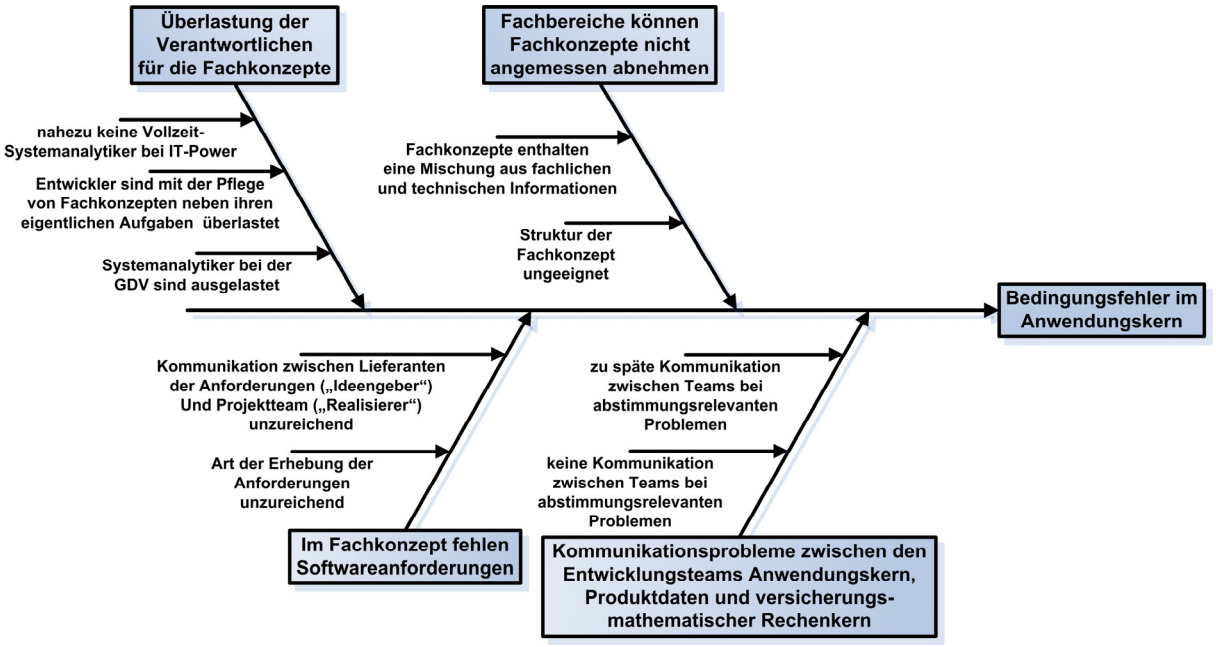


Abbildung 7-56: Ursachen für Bedingungsfehler im Anwendungskern

Gegenmaßnahmen

Dann wurden Gegenmaßnahmen unter der folgenden Fragestellung diskutiert: Wie können die Ursachen behoben werden, damit vergleichbare Bedingungsfehler im Anwendungskern zukünftig vermieden werden können?

Ursache	Maßnahmen
Im Fachkonzept fehlen Softwareanforderungen.	▪ Ablaufplan mit festen Meilensteinen festlegen ▪ Konkretisierung des erforderlichen Umfangs und Detailgrades der Vorgaben
Fachbereiche können Fachkonzepte nicht angemessen abnehmen.	▪ Trennung DV-Konzept und Fachlichkeit
Kommunikation zwischen Entwicklungsteams	▪ Kleine Schnittstellenteams, regelmäßige Sitzungen
Überlastung der Verantwortlichen für die Fachkonzepte.	▪ Entbindung von anderen Aufgaben ▪ Fokussierung auf bestimmte Fachkonzepte ▪ Ressourcenerhöhung

Tabelle 7-59: Maßnahmen zur Vermeidung von Bedingungsfehlern im Anwendungskern

Sofern die Ursachen nicht vollständig behoben werden können: Wie können entsprechende Bedingungsfehler im Anwendungskern früher gefunden werden?

Maßnahmen
▪ Fachbereiche früher in die Tests einbinden • Anwendungsfälle von Fachbereichen als Standardtests per Software aufnehmen und früh ausführen.

Tabelle 7-60: Maßnahmen zur frühen Entdeckung von Bedingungsfehlern im Anwendungskern

7.4.4.2.2 Fehlermuster: Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen

Charakterisierung Fehlermuster

Das Fehlermuster zu Bedingungsfehlern in den Produktdaten und Plausibilitätsbedingungen (vgl. Tabelle 7-61) wurde anhand von Auswertungen zu den Release 4.2 und 4.3 mit dazugehörigen Abbildungen charakterisiert. [765]

Fehlermuster: Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen
▪ Gegenstand der Änderung = Bedingung ▪ Komponente: Produktdaten und Plausibilitätsbedingungen ▪ QS-Kriterium: Variation/Abdeckung/Sequenz ▪ Auswirkung: bekannte Softwareanforderung ▪ Viele Bedingungsfehler werden zu spät im Test gefunden oder sogar ausgeliefert.

Tabelle 7-61: Merkmale des Fehlermusters: Bedingungsfehler in den Produktdaten

Diskussion konkreter Fehler

Exemplarisch wurden die Beschreibung von 4 Fehlern aus diesen Fehlermuster in der Gruppe näher beleuchtet. Insgesamt standen 14 Fehler aus 14 Problemberichten in Release 4.2 sowie 10 Fehler aus 9 Problemberichten in Release 4.3 zur Verfügung.

Zwei der diskutierten Implementierungsfehler sind keine Folgefehler und damit originär. Bekannte Softwareanforderungen wurden versehentlich oder aufgrund eines Missverständnisses falsch umgesetzt. Die zwei übrigen Implementierungsfehler sind eine Folge von unvollständigen Softwareanforderungen.

Ursachenanalyse

Ergebnis der Ursachenanalyse für Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen ist in Abbildung 7-57 dargestellt. Die einzelnen Ursachen lassen sich in die folgende Kategorien gruppieren:

- Vorgaben sind inkonsistent.
- Vorgaben fehlen.
- Produktdaten und Plausibilitätsbedingungen werden nicht durch Fachbereiche abgenommen.

[765] Vgl. zu den Auswertungen und entsprechenden Abbildungen zu diesem Fehlermuster die Kapitel 7.4.3.2.2 und 7.4.3.3.

- Umsetzung von Produktdaten und Plausibilitätsbedingungen.
- Vorgaben kommen zu spät.

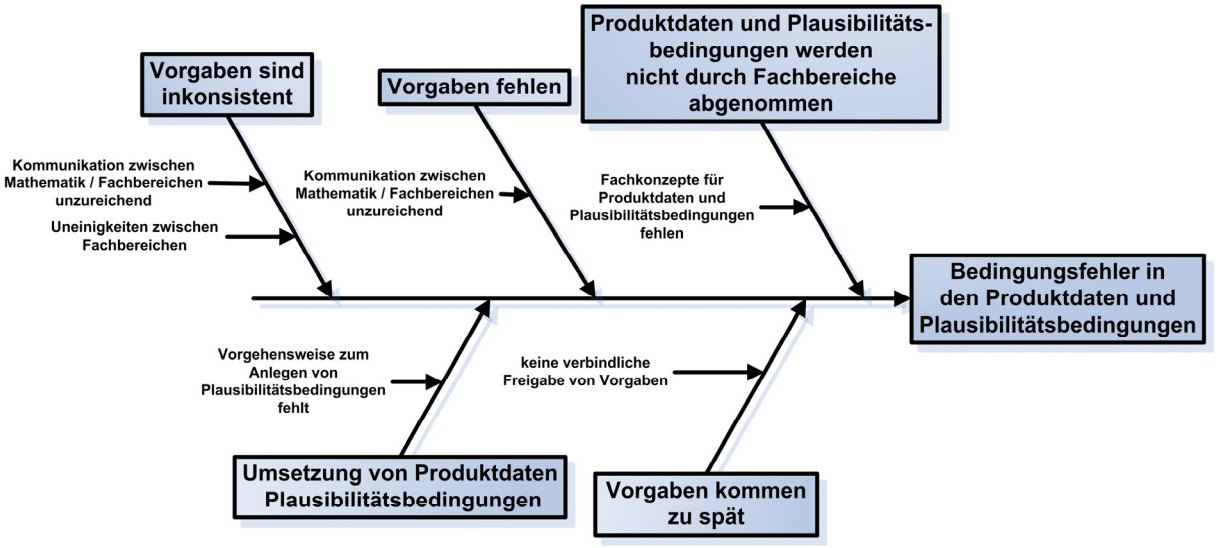


Abbildung 7-57: Ursachen für Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen

Gegenmaßnahmen

Wie können die Ursachen behoben werden, damit vergleichbare Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen zukünftig vermieden werden können?

Ursache	Maßnahmen
Vorgaben kommen zu spät	▪ fester Ablaufplan mit Meilensteinen
Umsetzung von Produktdaten und Plausibilitätsbedingungen	▪ Dokumentation zur Erstellung von Plausibilitätsbedingungen erarbeiten ▪ Qualität, Umfang, Termin für Vorgaben festlegen.
Plausibilitätsbedingungen werden durch Fachbereiche nicht abgenommen.	▪ Fachkonzept für Produktdaten und Plausibilitätsbedingungen erstellen.
Vorgaben sind inkonsistent.	▪ Fachkonzept für Produktdaten und Plausibilitätsbedingungen erstellen.
Vorgaben fehlen.	▪ Fachkonzept für Produktdaten und Plausibilitätsbedingungen erstellen.

Tabelle 7-62: Maßnahmen zur Vermeidung von Bedingungsfehlern in den Produktdaten und Plausibilitätsbedingungen

Sofern die Ursachen nicht vollständig behoben werden können: Wie können entsprechende Bedingungsfehler in den Produktdaten und Plausibilitätsbedingungen früher gefunden werden?

Maßnahmen
• Die Plausibilitätsbedingungen in den Produktdaten bereits ab Entwicklertest aktivieren. • Anwendungsfälle von Fachbereichen als Standardtests per Software aufnehmen und früh ausführen.

Tabelle 7-63: Maßnahmen zur frühen Entdeckung von Bedingungsfehlern in den Produktdaten und Plausibilitätsbedingungen

7.4.4.3 Beobachtungen im Rahmen der Ursachenanalyse

Die Ursachenanalyse konnte für zwei Fehlermuster ohne Schwierigkeiten durchgeführt werden. Nachfolgend werden wesentliche Beobachtungen aufgeführt, die während der Ursachenanalyse gemacht wurden.

Beobachtung 1:

Für den Einstieg in die Ursachenanalyse war es hilfreich, nach einer kurzen allgemeinen Darstellung des Fehlermusters, stichprobenartig konkrete einzelne Fehler aus diesem Fehlermuster zu diskutieren.

Diese Vorgehensweise ermöglicht es, zum einen nicht den Fokus auf das Fehlermuster zu verlieren und zum anderen die Diskussion über potenzielle Ursachen zu konkretisieren.

Beobachtung 2:

Werden die zu diskutierenden konkreten Fehler durch die Teilnehmer ausgewählt, steigert dies das Vertrauen der Teilnehmer in die Qualität der Fehlerklassifikation.

Teilnehmer der Arbeitssitzung äußerten sich positiv darüber, dass sie selber die zu diskutierenden Fehler zu dem jeweiligen Fehlermuster auswählen konnten. Das Vertrauen in die Qualität der Fehlerklassifikation wurde gestärkt, da einzelne Fehler die Merkmale des Fehlermusters aufwiesen.

Beobachtung 3:

Teilnehmer tendieren bisweilen dazu, Schuldzuweisungen an einzelne Personen und kleine Teams zu richten.

Im Mittelpunkt der Ursachenanalyse stehen Prozesse. Um eine offene Diskussionsatmosphäre zu schaffen und aufrecht zu halten, ist der Moderator gefordert, entsprechende Schuldzuweisungen zu unterbinden.

Beobachtung 4:

Teilnehmer tendieren bisweilen dazu, Ursachen außerhalb ihres Aufgabenbereichs zu suchen.

Ergebnis der Ursachenanalyse dürfen ausschließlich Ursachen in Prozessen sein, die die anwesenden Teilnehmer betreuen. Sofern Ursachen tatsächlich in anderen Prozessen liegen, ist der Teilnehmerkreis nicht zweckmäßig ausgewählt worden.

7.5 Diskussion der Fallstudienergebnisse

7.5.1 Zusammenfassung der Fallstudienergebnisse

Die explorative Fallstudie hatte das Ziel, die Frage zu beantworten, welche Erfahrungen bei der Konfiguration des Fehleranalyseverfahrens, der Fehlerklassifikation, der Fehlerauswertung und der Ursachenanalyse gemacht werden. Hierzu wurden die Erfahrungen für jeden dieser Verfahrensschritte gruppiert nach dem Ablauf des jeweiligen Verfahrensschrittes, dem Ergebnis des Verfahrensschrittes und Beobachtungen im Rahmen des Verfahrensschrittes.
Die Untersuchungseinheit der Fallstudie war die Anwendung des Fehleranalyseverfahrens in der Wartung einer geschäftskritischen Anwendungssoftware für eine große deutsche Versicherung. Das Fehleranalyseverfahren wurde auf Entwicklungs- und Produktionsfehler zweier Releases dieser Anwendungssoftware angewandt.

Das Sachziel des hergeleiteten Fehleranalyseverfahrens ist es, Mängel im Prozess der Anforderungsanalyse bereits während der Entwicklung einer Anwendungssoftware zu finden, indem Fehler erfasst und ausgewertet werden.
In der Fallstudie war das Fehleranalyseverfahren wirksam, Prozessmängel in der Anforderungsanalyse zu identifizieren. Beispielsweise deckte die Ursachenanalyse auf, dass Bedingungsfehler im Anwendungskern, die durch Einfügen korrigiert werden, häufig Folgefehler von fehlenden Softwareanforderungen sind.[766] Diese fehlenden Softwareanforderungen sind gemäß der Ursachenanalyse unter anderem eine Folge der unzureichenden Art der Erhebung von Softwareanforderungen sowie der mangelnden Verfügbarkeit von Systemanalytikern. Neben diesen Verbesserungspotenzialen für den Prozess der Anforderungsanalyse ermöglicht die Kenntnis dieser Prozessmängel auch die Verbesserung der Qualitätssicherungsprozesse. So war man in dem Softwareentwicklungsvorhaben insbesondere durch die Steigerung der automatisch durchgeführten Testfälle bis auf ca. 10 000 bemüht, Implementierungsfehler früh zu entdecken. Da jedoch Testfälle von Fachkonzepten abgeleitet wurden, war dies bei fehlenden Softwareanforderungen naheliegender Weise nicht möglich. Folglich wäre die Steigerung der Anzahl der Testfälle nicht effektiv, um Bedingungsfehler im Anwendungskern zu finden.

[766] Vgl. Kapitel 7.4.4.2.1.

Die vorgeschlagene Gegenmaßnahme dagegen, Fachbereiche früher in die Tests einzubinden, verspricht effektiver zu sein.

Da die Fehlerauswertung und die Ursachenanalyse aufgrund der Rahmenbedingungen ex post gemacht wurden, konnten keine Erfahrungen während der Entwicklung der Releases gesammelt werden. Jedoch weisen bereits die zur Entwicklung verfügbaren Fehler auf die identifizierten Fehlermuster zu den Zuweisungs- und Bedingungsfehlern hin (vgl. Abbildung 7-58 bis Abbildung 7-60) und stehen damit bereits während der Entwicklung einer Ursachenanalyse offen.

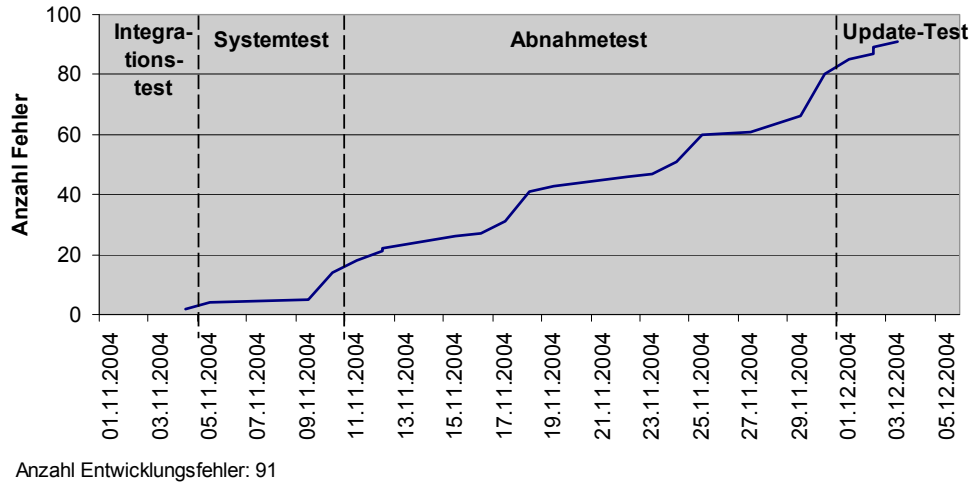


Abbildung 7-58: Kumulative Entwicklung der Entwicklungsfehler für Release 4.2

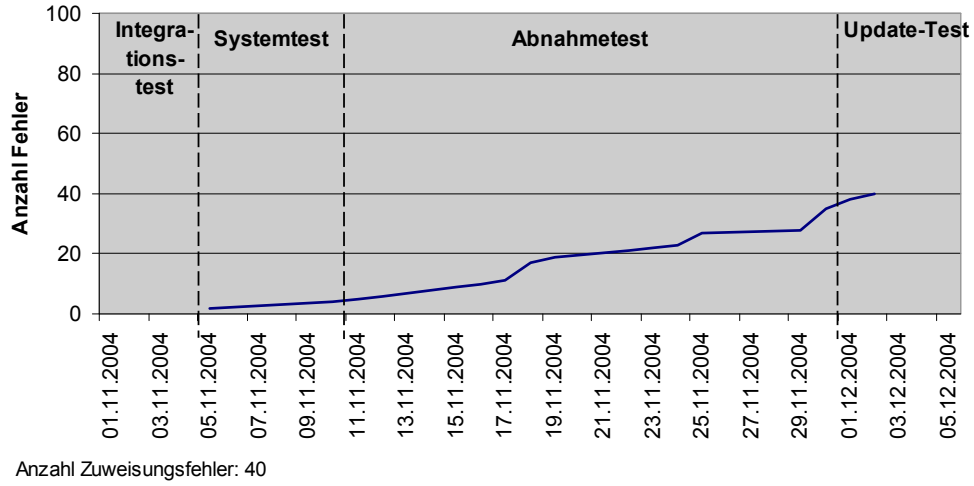


Abbildung 7-59: Kumulative Entwicklung der Zuweisungsfehler im Release 4.2

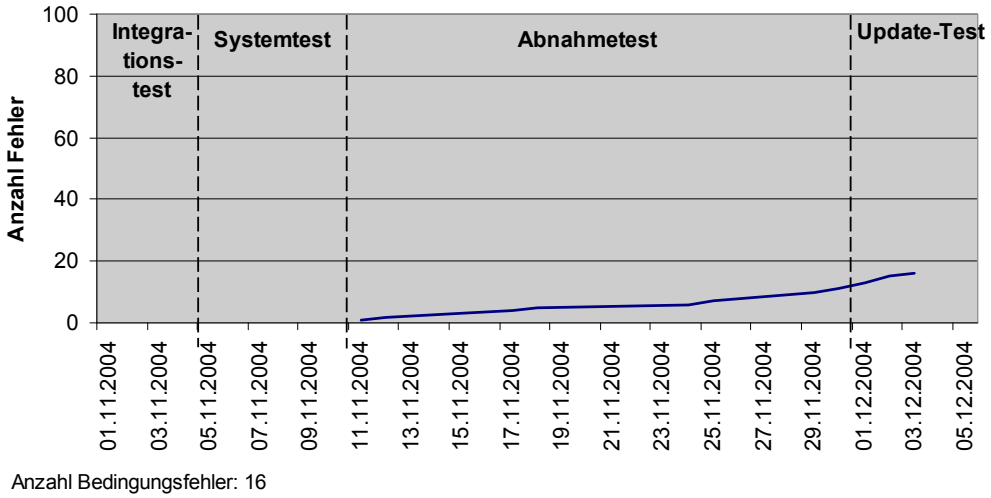


Abbildung 7-60: Kumulative Entwicklung der Bedingungsfehler im Release 4.2

Die Implementierungsfehler wurden durch den Autor der vorliegenden Arbeit nachträglich klassifiziert. Eine entsprechende Vorgehensweise verfolgen auch 11 von den 27 empirischen Untersuchungen zu ODC aus der empirischen Literaturanalyse in Kapitel 5.2.[767] Diese Vorgehensweise hat Vor- und Nachteile. Zu den Vorteilen können folgende Punkte gezählt werden:

- Sofern Fehler nachträglich klassifiziert werden und zum Zeitpunkt der Erfassung der Fehler nicht bekannt ist, dass eine empirische Untersuchung durchgeführt werden soll, erfolgt die Fehlererfassung unbeeinflußt. So kann beispielsweise kein Hawthorne-Effekt auftreten.[768] Dieser Effekt beschreibt das Phänomen, dass allein die Anwesenheit von Forschern und ihr Interesse für Untersuchungssubjekte die Leistung der Untersuchungssubjekte erhöht.
- Erfolgt die nachträgliche Klassifikation durch Forscher, entsteht hierdurch kein zusätzlicher Aufwand für die an der Untersuchung teilnehmenden Personen beziehungsweise Unternehmen.
- Erfolgt die Klassifikation durch einen Forscher oder einer kleinen Gruppe von Forschern, so ist die Wahrscheinlichkeit tendenziell höher, dass die Klassifikation für alle Fehler einheitlich durchgeführt wurde.

Nachteilig bei der Vorgehensweise sind folgende Punkte:

---

[767] Vgl. Anhang A.2. Fehler werden in folgenden empirischen Untersuchungen zu ODC nachträglich klassifiziert: Bassin, Santhanam /Software triggers/, Buczilowski /Defect reduction/, Chillarege, Bassin /Software triggers/, Chillarege, Kao, Condit /Defect type/, Chillarege, Prasad /ODC Triggers/, Damm, Lundberg /Test process improvement/, Henningsson, Wohlin /Fault classification agreement/, Lutz, Mikulski /Anomalies/, Lutz, Mikulski /Requirements/, Sullivan, Chillarege /Comparison/ und Zheng u. a. /Static analysis/.

[768] Vgl. Staehle /Management/ 34.

- Mit einer nachträglichen Fehlerklassifikation durch Forscher entzieht man sich der Möglichkeit, Erfahrungen bezüglich der Klassifikation durch Softwareentwickler zu gewinnen.
- Eine nachträgliche Fehlerklassifikation setzt voraus, dass Fehler umfangreich dokumentiert sind und relevante Informationen verfügbar sind. Andernfalls kann das Fehlerklassifikationsschema nicht vollständig angewandt werden.
- Die nachträgliche Fehlerklassifikation ist im Vergleich zur direkten aufwändiger.

Obwohl mehrere empirische Untersuchungen zu ODC vorliegen, in denen ebenfalls Fehler nachträglich klassifiziert werden, wird in keiner diese Untersuchung diskutiert, welche Herausforderungen durch die nachträgliche Klassifikation entstehen. In der vorliegenden Fallstudie wurde z. B. festgestellt, dass ein Problembericht zunächst ein Fehlverhalten der Software beschreibt. Ein Fehler kann aber mehrere Arten von Fehlverhalten verursachen und mehrere unterschiedliche Arten von Fehlverhalten können unter Umständen auf den gleichen Fehler zurückgeführt werden. So beschreiben 343 Problemberichte insgesamt 368 Fehler.[769] Von potenziellen 678 Implementierungsfehlern sind nur 54% der Fehler für die Fehlerauswertung und Ursachenanalyse relevant.

7.5.2 Beurteilung der Fallstudie

Dubé und Paré beschreiben Empfehlungen für die Durchführung von explorative Fallstudien im positivistischen Kontext.[770] Hierbei greifen sie auf Arbeiten von Benbasat u. a.[771], Yin[772], Eisenhardt[773] und Lee[774] zurück. Die im Rahmen der vorliegenden Arbeit durchgeführte Fallstudie orientiert sich an diesen Empfehlungen.

Für die Qualität einer explorativen Fallstudie folgende Aspekte relevant:[775]

- A priori Definition der Forschungsfragen, des theoretischen Bezugsrahmen und der Untersuchungseinheit
- Generalisierbarkeit der Fallstudienergebnisse
- Intersubjektive Nachvollziehbarkeit der Fallstudie

Die Fallstudie der vorliegenden Arbeit wird nachfolgend bezüglich dieser Aspekte beurteilt.

---

[769] Vgl. hierzu und zu Folgendem Kapitel 7.4.2.2.

[770] Vgl. Dubé, Paré /Case research/ 606.

[771] Vgl. Benbasat, Goldstein, Mead /Case research strategy/.

[772] Vgl. Yin /Case study research/.

[773] Vgl. Eisenhardt /Case study research/.

[774] Vgl. Lee /Case studies/.

[775] Vgl. Paré /Case study research/ 237-240 und Yin /Case study research/ 19, 31-35.

A priori Definition der Forschungsfragen, des theoretischen Bezugsrahmen und der Untersuchungseinheit

Bei explorativen Fallstudien ist es wichtig, dass vor Beginn der Fallstudiendurchführung die Forschungsfragen, der theoretische Bezugsrahmen und die Untersuchungseinheit präzise definiert sind.[776] Dadurch soll sichergestellt werden, dass im Rahmen der Fallstudie nur relevante theoretischen Kontrukte untersucht werden und diese das erfassen, was sie erfassen sollen.[777] Um dies zu erreichen, empfiehlt Yin,

- Erkenntnisse aus mehrere Informationsquellen abzuleiten,
- Beweisketten zu bilden, d. h. die Verknüpfungen von den Forschungsfragen bis zu den Fallstudienergebnissen offen zu legen, und
- wichtige Erkenntnisse aus der Datenanalyse zu dokumentieren und durch Schlüsselpersonen abnehmen zu lassen.

Der theoretische Bezugsrahmen dieser Arbeit ist das Erklärungsmodell für Anforderungsfehler in Kapitel 4. Die Untersuchungseinheit ist das eine Weiterentwicklung der Orthogonal Defect Classification (vgl. die Kapitel 5 und 6).

In der vorliegenden Fallstudie wurden mit semistrukturierten Interviews, Dokumentenanalysen und Fragebogenerhebung drei unterschiedliche Methoden zur Datenerhebung eingesetzt und damit die Möglichkeit für eine Datentriangulation eröffnet. Im Rahmen der Fallstudie gewonnene Erkenntnisse beruhen weitgehend mindestens auf zwei unterschiedlichen Informationsquellen.[778] Dies kann sich darin äußern, dass eine Aussage von zwei befragten Personen unabhängig voneinander bestätigt wird oder die Aussage einer Person durch entsprechende Projektdokumentation gestützt wird. Neben qualitativen Daten werden mit der Klassifikation von Implementierungsfehlern auch auf quantitative Daten in der Fallstudie zurückgegriffen.

In der Fallstudie wurden die Forschungsfragen explizit gemacht. Die explorative Fallstudie hatte das Ziel, die Frage zu beantworten, welche Erfahrungen bei der Konfiguration des Fehleranalyseverfahrens, der Fehlerklassifikation, der Fehlerauswertung und der Ursachenanalyse gemacht werden. Zu jedem Verfahrensschritt wurde der Ablauf der Fallstudie beschrieben und erläutert wie jeweiligen Ergebnisse zustande gekommen sind.

---

[776] Vgl. Paré /Case study research/ 239 f.

[777] Vgl. und zu Folgendem Yin /Case study research/ 34 f.

[778] Vgl. Dubé, Paré /Case research/ 615.

Für die Fallstudie wichtige Erkenntnisse aus der Datenanalyse wurden durch den Autor dokumentiert und zur Abnahme durch Schlüsselpersonen der Fallstudie vorgelegt.[779] Diese Dokumente sind in Anhang C.2 aufgeführt.

Generalisierbarkeit der Fallstudienergebnisse

Die Generalisierbarkeit beschäftigt sich mit der Fragestellung, ob Erkenntnisse der Fallstudie über den untersuchten Fall hinaus verallgemeinert werden können.[780]

Mittels Fallstudien kann keine statistische Generalisierbarkeit von Aussagen erreicht werden.[781] Bei der statistischen Generalisierbarkeit wird eine Aussage, die auf eine Stichprobe zutrifft, auf eine Grundgesamtheit, also die Menge aller potentiellen Untersuchungsobjekte für eine bestimmte Fragestellung, verallgemeinert. Eine analytische Generalisierbarkeit ist dagegen mit einer Fallstudie möglich. Dabei werden Fallstudienergebnisse zu einer Theorie oder einem gedanklichen Bezugsrahmen verallgemeinert. Eine Überprüfung der Theorie oder des gedanklichen Bezugsrahmen steht dann weiterhin aus.

Mittels der vorliegenden Fallstudie wird eine analytische Generalisierbarkeit angestrebt. Hierbei sollen neue Erkenntnisse für das hergeleitete Fehleranalysefahren als gedanklicher Bezugsrahmen mit dem Charakter eines Entscheidungsmodells gewonnen werden.

Das hergeleitete Fehleranalyseverfahren wurde nur im Rahmen eines Einzelfalls angewandt. Dieser Einzelfall ist das Softwareentwicklungsvorhaben LV-Neu. Der Grund für die Auswahl dieses Einzelfalls war, dass es im Sinne von Yin typisch[782] für die Wartung von geschäftskritischer Anwendungssoftware in Großunternehmen, insbesondere Versicherungen, ist.

Intersubjektive Nachvollziehbarkeit der Fallstudie

Bei einer explorativen Fallstudie ist zu gewährleisten, dass sie intersubjektiv nachvollziehbar ist.[783] Yin schlägt hierfür zum einen den Aufbau einer Fallstudiendatenbank[784] und zum anderen die Nutzung eines Fallstudienprotokolls[785] vor, das die Datensammlung steuert. Ein

---

[779] Vgl. Dubé, Paré /Case research/ 620.

[780] Vgl. Yin /Case study research/ 37.

[781] Vgl. zu diesem Absatz Yin /Case study research/ 31-33.

[782] Vgl. Yin /Case study research/ 41.

[783] Vgl. hierzu 37-39 Bei einer Fallstudie kann keine intersubjektive Nachprüfbarkeit (Verifizierbarkeit) erreicht werden, da eine identische Wiederholung der Untersuchungen nicht möglich ist. Deshalb wird gefordert eine intersubjektive Nachvollziehbarkeit sicherzustellen. Vgl. hierzu Steinke /Qualitative Research/ 186f.

[784] Vgl. Dubé, Paré /Case research/ 616.

[785] Vgl. Yin /Case study research/ 67 f.

Fallstudienprotokoll ist insbesondere dann erforderlich, wenn eine Untersuchungseinheit unter den gleichen Forschungsfragen bei mehreren Fällen untersucht werden soll.
Im Rahmen der vorliegenden Fallstudie wurde eine Fallstudiendatenbank angelegt, welche unter anderem folgende Elemente umfasst:

- alle Interviewleitfäden,
- die Gesprächsprotokolle aller Interviews (siehe Anhang C.3),
- alle Antworten auf die E-Mail-Fragebögen (siehe Anhang C.3),
- alle von IT-Power zur Verfügung gestellen Dokumente (siehe Anhang C.1),
- alle durch den Autor erstellten Arbeitsergebnisse (z. B. klassifizierte Fehler, Folien etc.) (siehe Anhang C.2) und
- eingesetzte Softwareprodukte (insbesondere IBM Lotus Notes Client für die Problemdatenbank und Softwarewerkzeuge zur Messung des Codeumfangs).

Um die intersubjektive Nachvollziehbarkeit zu erhöhen, wurde zum einen die Forschungsmethodik im Rahmen der Fallstudie detailliert beschrieben und zum anderen der Ablauf jedes Verfahrensschrittes der Fehleranalyse erläutert. Zugleich wurde der Kontext der Fallstudie, insbesondere das Untersuchungsobjekt ausführlich dargestellt. Diese Information sind gemäß Yin die wesentlichen Bestandteile eines Fallstudienprotokolls.[786]

Einschränkungen der Fallstudienergebnisse

Um die analystische Generalisierbarkeit der Fallstudienergebnisse zu erhöhen, könnte die Untersuchungseinheit[787] unter den gleichen Forschungsfragen bei mehreren Fällen untersucht werden. Das hergeleitete Fehleranalyseverfahren wurde jedoch nur im Rahmen eines Einzelfalls angewandt.
Das hergeleitete Fehleranalyseverfahren wurde aufgrund der Rahmenbedingungen ex post auf abgeschlossene Softwareentwicklungsprojekte angewandt. Hierdurch konnte keine Erfahrungen bzgl. der Anwendung des Verfahrens während eines Softwareentwicklungsprojekts gesammelt werden.
Aufgrund der Rahmenbedingungen der Fallstudie konnten keine Anforderungsfehler klassifiziert und ausgewertet werden. Folglich konnte das hergeleitete Fehleranalyseverfahren nicht vollständig angewandt werden. Aber selbst anhand der Analyse von Implementierungsfehlern konnten Prozessmängel in der Anforderungsanalyse identifiziert werden.

---

[786] Vgl. Yin /Case study research/ 67-69.

[787] Die Subjekte oder Objekte der Untersuchung sind nicht zu verwechseln mit der Untersuchungseinheit. Wird eine Untersuchung wiederholt, bleibt die Untersuchungseinheit bestehen, während Subjekte und Objekte der Untersuchung ersetzt werden können.

Das Fehleranalyseverfahren war in der Fallstudie effektiv. Es konnte projektspezifische Prozessmängel in der Anforderungsanalyse identifizieren. Im Gegensatz zur Effektivität konnte im Rahmen der Fallstudie keine Erkenntnisse zur Effizienz des Verfahrens gemacht werden. Wesentlicher Grund hierfür ist, dass die Klassifikation nicht durch Entwickler und Test durchgeführt wurde. Die nachträgliche Klassifikation erfolgte ausschließlich durch den Autor der vorliegenden Arbeit. Folglich liegen keine Erkenntnisse bezüglich der insgesamt erforderlichen Dauer und des erforderlichen Aufwands vor.

8 Fazit

Das Hauptziel der vorliegenden Arbeit lautet:[788]

> *Wie können Mängel im Prozess der Anforderungsanalyse bereits während der Entwicklung einer Anwendungssoftware gefunden werden, indem Fehler erfasst und ausgewertet werden?*

Aus diesem Hauptziel wurden zehn Teilziele abgeleitet:

1. *Was sind die konstituierenden Merkmale der Anforderungsanalyse und der Qualitätssicherung für die Entwicklung von Anwendungssoftware? (Grundlagen)*
2. *Was ist ein Prozess und was ein Fehler? (Grundlagen)*
3. *Welche empirischen Erkenntnisse liegen über die Häufigkeit von Fehlern vor, die auf Mängel im Prozess der Anforderungsanalyse zurückzuführen sind? (Grundlagen)*
4. *Welcher Aufwand entsteht gemäß empirischen Untersuchungen, um Anforderungsfehler zu korrigieren? (Grundlagen)*
5. *Welche fehlerbasierten Verfahren existieren, um Mängel in Prozessen der Softwareentwicklung zu finden? (Grundlagen)*
6. *Wie können Mängel im Prozess der Anforderungsanalyse beschrieben und erklärt werden? (Konzeption)*
7. *Was sind gemäß empirischen Untersuchungen bekannte Mängel im Prozess der Anforderungsanalyse? Welche Merkmale weisen Fehler auf, die durch diese Prozessmängel verursacht wurden? (Konzeption)*
8. *Wie sollte ein praktisch einsetzbares Verfahren aussehen, mit dem Fehler erfasst und ausgewertet werden, um bereits während der Entwicklung einer Anwendungssoftware Mängel im Prozess der Anforderungsanalyse zu finden? (Konzeption)*
9. *Welche Nutzenpotenziale bietet die Anwendung eines solchen Verfahrens? (Konzeption)*
10. *Welche Erfahrungen werden gemacht, wenn das entwickelte Verfahren praktisch eingesetzt wird? (Anwendung)*

[788] Vgl. Kapitel 1.2.

Mit der Kenntnis der Prozessmängel in der Anforderungsanalyse eröffnet sich die Möglichkeit, diese zu beseitigen oder, sofern dies nicht möglich ist, die Prozesse der Qualitätssicherung zu verbessern, um gezielt Fehler der Prozessmängel früher zu finden.

Hierzu wurde zunächst in Kapitel 2 ein Begriffssystem beschrieben, dass die konstituierenden Merkmale der Anforderungsanalyse und der Qualitätssicherung für die Entwicklung von Anwendungssoftware aufzeigt (Teilziel 1) und grundlegende Begriffe wie Prozess und Fehler einführt (Teilziel 2).

Das Hauptziel der Arbeit wurde damit begründet, dass

- viele Fehler auf Mängel im Prozess der Anforderungsanalyse zurückzuführen sind (*Inhalt des Praxisproblems*) und
- diese Anforderungs- und Folgefehler aufwändiger zu korrigieren sind, je später sie entdeckt und behoben werden (*Relevanz des Praxisproblems*).[789]

Beide Thesen wurden durch empirische Befunde belegt (Teilziel 3 und 4).[790]

Um das Hauptziel der vorliegenden Arbeit zu beantworten und damit einen Beitrag zur Lösung des beschriebenen Praxisproblems zu leisten, wurde in Kapitel 4 ein Erklärungsmodell für Anforderungsfehler hergeleitet. Dieses Erklärungsmodell ist ein gedanklicher Bezugsrahmen[791] und nutzt Gesetzesmäßigkeiten, um zu erklären, warum Anforderungsfehler entstehen. Aufgrund der explorativen Ausrichtung der vorliegenden Arbeit haben diese Gesetzesmäßigkeiten hypothetischen Charakter, da ihre Prüfung noch aussteht.[792]

Das Erklärungsmodell besteht aus zwei Teilen:

- Einem theoretischen Ansatz, der allgemein anhand hypothetischer Gesetzmäßigkeiten erklärt, warum Anforderungsfehler entstehen (Teilziel 6).
- Einer Menge von Erklärungsmustern für bestimmte Typen von Anforderungsfehlern. Diese Erklärungsmuster sind einzelne hypothetische Gesetzmäßigkeiten, die erklären, warum bestimmte Typen von Anforderungsfehlern entstehen (Teilziel 7). Diese Menge von Erklärungsmustern ist an die Ergebnisse der Literaturanalyse aus Kapitel 4.3.4 angelehnt.

Das Erklärungsmodell leistet als gedanklicher Bezugsrahmen bereits praktische Gestaltungshilfe, ohne das Stadium einer erklärenden Theorie erreicht zu haben (vgl. Abbildung 8-1). Es ist integraler Bestandteil des hergeleiten Verfahrens zu Fehleranalyse und stellt „Interpreta-

---

[789] Vgl. Kapitel 1.1.2.

[790] Vgl. hierzu die Kapitel 3.2 und 3.3.

[791] Vgl. hierzu Kapitel 1.3.4.1.

[792] Vgl. Gadenne /Rationalismus/ 5, 14.

tionsmuster“ zur Verfügung, die ein besseres Verständnis relevanter Ausschnitte der Wirklichkeit ermöglichen.[793]

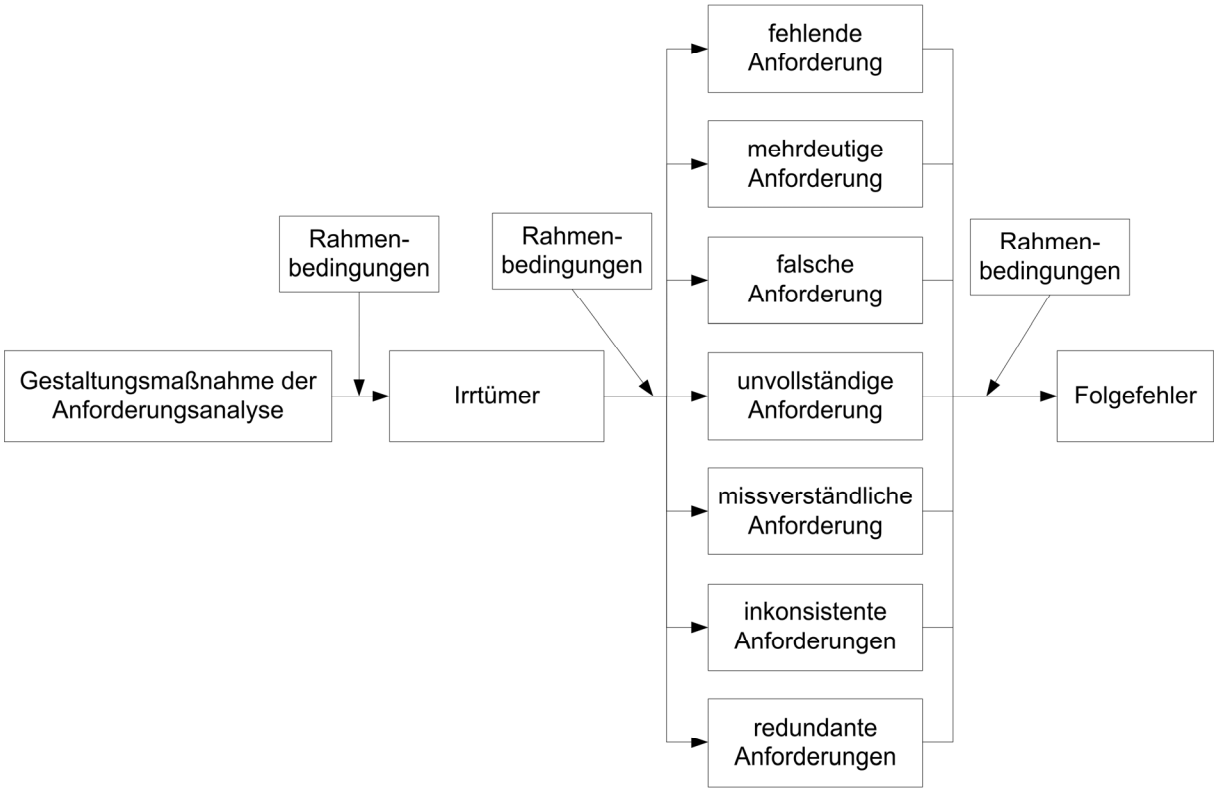


Abbildung 8-1: Theoretischer Ansatz des Erklärungsmodells für Anforderungsfehler und ihre Folgefehler

Ausgehend von dem Verständnis, dass ein Prozess ein Bündel von Gestaltungsmaßnahmen ist, ist ein Prozessmangel der Anforderungsanalyse eine Maßnahme, die *menschliche Irrtümer* begünstigt.[794] Ein Irrtum liegt vor, wenn eine geplante Abfolge von geistigen oder körperlichen Aktivitäten ohne Fremdeinwirkung nicht zum beabsichtigten Ergebnis führt. Voraussetzung für einen Irrtum ist die Absicht zur Handlung. Eine Absicht ist gegeben, wenn dem Individuum das erwartete Ergebnis und die Mittel bewusst sind, mit denen das Ergebnis erreicht werden soll.

Unterliegt ein Projektmitarbeiter einem Irrtum während er Teilaufgaben der Anforderungsanalyse ausführt, kann sein Irrtum dazuführen, dass Anforderungsfehler entstehen. Ein *Fehler* ist ein unzureichendes Merkmal oder ein erwartetes, jedoch fehlendes Merkmal eines Arbeitsergebnisses der Softwareentwicklung, sofern es eine Änderung in diesem Ergebnis notwendig macht. Fehler, die infolge eines Irrtums neu eingeführt werden, sind *originäre Fehler*.

---

[793] Vgl. Kubicek /Organisationsforschung/ 39, 45.

[794] Vgl. zum Folgenden Kapitel 4.2.

Irrtümer in der Anforderungsanalyse verursachen systematisch bestimmte Typen von Anforderungsfehlern. Wenn sich ein Anforderungsfehler auf eine Anforderung bezieht, können insgesamt fünf Fälle voneinander abgegrenzt werden:

- Für ein bestehendes Kundenbedürfnis wurde eine Anforderung gänzlich übersehen (*fehlende Anforderung*).
- Die Beschreibung der Anforderung kann auf mehrere Arten gedeutet werden (*mehrdeutige Anforderung*).
- Die Anforderung erfüllt kein Kundenbedürfnis oder erfüllt sie nicht korrekt (*falsche Anforderung*).[795]
- Die Anforderung ist unvollständig beschrieben (*unvollständige Anforderung*).
- Die Anforderung ist zwar vollständig beschrieben, jedoch fehlen Kontextinformationen zu dieser Anforderung oder Verknüpfungen zu vorhandenen Kontextinformationen, um die Anforderung zu verstehen (*missverständliche Anforderung*). Kontextinformationen sind z. B.
  - Kundenbedürfnisse, die eine Anforderung begründen,
  - für die Anforderung relevante Kundengruppen oder
  - andere abhängige Anforderungen.

Ein Anforderungsfehler kann sich auch auf eine Beziehung zwischen zwei oder mehreren Anforderungen beziehen. Zwei Fälle können dabei unterschieden werden:

- Eine Anforderung steht im Widerspruch zu einer oder mehreren anderen Anforderungen (*inkonsistente Anforderungen*).
- Eine Anforderung ist redundant zu mindestens einer anderen Anforderung (*redundante Anforderungen*).

Die 7 Typen von Anforderungsfehlern können Folgefehler in Entwurf und Implementierung verursachen, wenn sie nicht frühzeitig entdeckt werden. Diese *Folgefehler* haben in Abgrenzung zu originären Fehlern ihre Ursache in einem Fehler einer vorgelagerten Entscheidung.

Die Wirkung einzelner Gestaltungsmaßnahmen ist abhängig von den Rahmenbedingungen, unter denen sie durchgeführt werden.[796] Folglich sind auch Prozessmängel situativ.[797]

---

[795] Vgl. hierzu Kapitel 2.2.1.

[796] Vgl. hierzu Abbildung 1-3.

[797] Vgl. hierzu die Fußnote 32 und die dort aufgeführten Literaturquellen.

Das hergeleitete Verfahren zur Fehleranalyse ist eine Weiterentwicklung der Orthogonal Defect Classification (ODC). Kapitel 3.4 gibt einen Überblick über den Stand der Forschung zu fehlerbasierten Verfahren (Teilziel 5). Eine Feststellung ist, dass ODC im Vergleich zu den zwei anderen Vorschlägen ausgereifter und in Forschung und Praxis etablierter ist. Jedoch hat ODC ebenso deutliche Schwächen. So ist ODC wie die beiden anderen fehlerbasierten Verfahren nicht ausreichend theoretisch begründet und unterstützt die Identifikation und Analyse von Prozessmängeln in der Anforderungsanalyse nur rudimentär. Diesen Schwächen wird im Rahmen der Weiterentwicklung des Verfahrens durch die Integration des Erklärungsmodells begegnet.

Vor der eigentlichen Weiterentwicklung der ODC, wird in Kapitel 5 ODC ausführlich dargestellt und anschließend mittels einer systematischen Literaturanalyse kritisch gewürdigt (Teilziel 8). Letzteres erfolgt, indem empirische Befunde zu ODC zu folgenden Punkten zusammengetragen und ausgewertet werden:

- Anwendungsgebiete und Nutzen von ODC
- Voraussetzungen und Aufwand für die Einführung und Anwendung von ODC
- Probleme und Herausforderungen bei der Einführung und Anwendung von ODC

Mit der kritischen Würdigung der ODC werden zugleich die Nutzenpotenziale der ODC und damit auch einer Weiterentwicklung auf der Grundlage empirischer Befunde beleuchtet (Teilziel 6).

In Kapitel 6 wurde schließlich ein Verfahren als Entscheidungsmodell entwickelt, das auf ODC aufbaut (Teilziel 8). Dabei wurde das Erklärungsmodell integriert und die Erkenntnisse der systematischen Literaturanalyse zu ODC berücksichtigt.

Die Ergebnisse der Anwendung des entwickelten Verfahrens im Rahmen einer Fallstudie wurden anschließend in Kapitel 7 beschrieben (Teilziel 10). Ziel der explorativen Fallstudie ist die Durchführbarkeit des hergeleiteten Fehleranalyseverfahrens zu untersuchen.

Die Untersuchungseinheit der Fallstudie ist die Anwendung des Fehleranalyseverfahrens in einem realen Softwareentwicklungsvorhaben bei dem IT-Dienstleister einer großen deutschen Versicherung. Es wurden nachträglich reale Fehler von zwei Softwareentwicklungsprojekten zu einer geschäftskritischen Anwendungssoftware für Lebensversicherungen erfasst und analysiert, um Mängel im Prozess der Anforderungsanalyse dieses Entwicklungsvorhabens aufzudecken.

9 Anhang A: Empirische Literaturquellen

9.1 Anhang A.1: Empirische Literaturquellen zu Prozessmängeln und Anforderungsfehlern

Nachfolgend werden 21 empirische Quellen anhand einheitlicher Beschreibungskriterien zusammengefasst. Diese Quellen werden im Rahmen der systematischen Literaturanalyse zu Prozessmängeln und Anforderungsfehlern aus Kapitel 4.3 berücksichtigt. Die eingesetzten Beschreibungskriterien werden in Kapitel 4.3.3.2 eingeführt.

Informationen zur Literaturquelle		
Nr.: Q1	**Kurztitel:** Adelson, Soloway /Domain experience/	**Jahr der Veröffentlichung**: 1985

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 1351)

- Welche Bedeutung haben Erfahrungen in einem Anwendungsgebiet für den Softwareentwurf?[798]
- Wie lösen Softwaredesigner Aufgabenstellungen in ihnen vertrauten und nicht vertrauten Anwendungsgebieten?
- Untersuchungseinheit: Erfahrungen in einem Anwendungsgebiet.

Vorgehensweise der Untersuchung (S. 1352 f.)

- Softwaredesigner sollen den Grobentwurf für drei Entwicklungsobjekte aus verschiedenen Anwendungsgebieten durchführen, wobei jeweils Anwendungsgebiet und Entwicklungsobjekt ihnen vertraut oder nicht vertraut sind:
  - E-Mail-System (Anwendungsgebiet vertraut, Entwicklungsobjekt unvertraut)
  - Verwaltungssystem für eine Bibliothek (Anwendungsgebiet unvertraut, Entwicklungsobjekt unvertraut)
  - Interruptbehandlungsroutine (Anwendungsgebiet vertraut, Entwicklungsobjekt vertraut)
- Protokollanalyse mit Videoaufzeichnung: Untersuchungssubjekte versuchen gestellte Aufgaben zu lösen und sollen dabei ihre Gedanken laut artikulieren.

Subjekte bzw. Objekte der Untersuchung (S. 1352)

- 3 Softwaredesigner mit jeweils mindestens acht Jahren Erfahrung im Softwareentwurf, insbesondere im Anwendungsgebiet Kommunikationssysteme.
  (Experten gemäß der Untersuchung)
- 2 Softwaredesigner mit weniger als zwei Jahren Erfahrung im Softwareentwurf und mehrjähriger Erfahrung als Programmierer (Anfänger gemäß der Untersuchung)
- Alle Untersuchungssubjekte kommen aus einem Unternehmen.

Wesentliche Ergebnisse der Untersuchung (S. 1359 f.)

- Untersuchungssubjekte versuchen mentale Modelle des Entwicklungsobjekts zu bilden. Diese mentalen Modelle bilden die Abhängigkeiten der geforderten Funktionen ab und ermöglichen eine mentale Simulation des Entwicklungsobjekts.
- Die mentale Simulation eines Entwicklungsobjekts unterstützt die Untersuchungssubjekte dabei, Erfahrungen aus einem vertrauten Anwendungsgebiet auf ein unvertrautes Entwicklungsobjekt anzuwenden.
- Ein mentales Modell wird schrittweise verfeinert. Die Untersuchungssubjekte achten darauf, dass der Verfeinerungsgrad in einem Modell einheitlich bleibt. Ein einheitlicher Verfeinerungsgrad unterstützt ebenfalls die mentale Simulation.
- Die Untersuchungssubjekte versuchen neben mentalen Modellen notwendige Bedingungen für das Zusammenspiel der Funktionen zu identifizieren, um daraus weitere Schlüsse für die mentale Simulation des Entwicklungsobjekts zu folgern.
- Fehlen Erfahrungen in einem Anwendungsgebiet, bleiben mentale Modelle auf einer groben Ebene und es werden nur wenige Bedingungen für das Zusammenspiel der Funktionen identifiziert. Eine Simulation des Entwicklungsobjekts bleibt in unvertrauten Anwendungsgebieten deshalb aus.

[798] Diese Untersuchung konzentriert sich auf den Softwareentwurf. Sie wird in der Literaturanalyse trotzdem berücksichtigt, da erstens die Aufgabenstellungen der Untersuchungssubjekte erfordert, dass sie sich auch über die Softwareanforderungen Gedanken machen müssen. Die mentalen Modelle und identifizierten Bedingungen stellen bis zu einer bestimmten Ebene der Verfeinerung Softwareanforderungen dar. Zweitens werden mit Erfahrungen in einem Anwendungsgebiet implizit Erfahrungen gängiger Kundenbedürfnisse und Anforderungen einer Software für bestimmte Einsatzwecke verstanden.

Informationen zur Literaturquelle

Nr.: Q2 **Kurztitel:** Al-Rawas, Easterbrook /Communications problems/ **Jahr der Veröffentlichung**: 1996

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 48)

- Welche Probleme treten in der Kommunikation zwischen Benutzern und Softwareentwicklern im Rahmen der Anforderungsanalyse auf? Was sind ihre Ursachen?
- Untersuchungseinheit: Kommunikation zwischen Benutzern und Softwareentwicklern im Rahmen der Anforderungsanalyse.

Vorgehensweise der Untersuchung (S. 48 f.)

- Erste Stufe der Untersuchung: Interviews
  - Informelle Interviews mit Softwareentwicklern und Benutzern. Die Interviews werden in der Regel aufgezeichnet.
  - Inhaltlicher Schwerpunkt der Interviews: Über welche Wege findet die Kommunikation zwischen Benutzern und Softwareentwicklern in der Anforderungsanalyse statt? Welche Probleme treten auf, die auf ineffektive Kommunikationswege zurückzuführen sind?
- Zweite Stufe der Untersuchung: Fragebögen
  - Konzeption von zwei Fragebögen mit Likert-Skalen zur quantitativen Datensammlung: eines für Softwareentwickler und eines für Benutzer.
  - Fragebögen werden an über 50 Unternehmen in England und Oman geschickt.
  - Auswahl der Unternehmen durch persönliche Kontakte und Rückgriff auf Branchenadressverzeichnisse.
  - Die Adressaten in den Unternehmen werden gebeten, die Fragebögen an geeignete Personen im Unternehmen weiterzuleiten.
  - Geeignete Personen sind Softwareentwickler, die an einer Anforderungsanalyse mitgewirkt haben, und Benutzer, die ein neu angepasstes Anwendungssystem nutzen.
  - Berücksichtigte Projekte umfassen sowohl Neuentwicklungen als auch Wartungen von Software.

Subjekte bzw. Objekte der Untersuchung (S. 48 f.)

- In der ersten Stufe: informelle Interviews mit 6 Softwareentwicklern und 5 Benutzern
- In der zweiten Stufe:
  - 37 Entwickler beantworten jeweils einen Entwicklerfragebogen und 32 Benutzer jeweils einen Benutzerfragebogen.
  - Von den 32 Benutzen waren 18 an einer Softwareentwicklung beteiligt und 14 nicht.
  - Befragte Entwickler und Benutzer haben unterschiedliche Erfahrungen, Kenntnisse und Hintergründe.

Wesentliche Ergebnisse der Untersuchung (S. 49-57)

- 44% der Benutzer erhalten zur Validierung Softwareanforderungen in Form von *formalen Anforderungsdokumenten*. Andere Formen sind Diskussionen, informelle Anforderungsdokumente und eine Kombination aus Dokumenten und Diskussionen. (Ergebnis der Benutzerbefragung)
- 86% der Softwareentwickler meinen, dass Benutzer zusätzlich natürlich-sprachliche Erklärungen benötigen für Anforderungen, die mit grafischen Notationen beschriebenen sind. (Ergebnis der Entwicklerbefragung)
- Die Wahl von Benutzervertretern orientiert sich an erster Stelle an Empfehlungen durch Kundenverantwortliche und an zweiter Stelle am Wissen über das Anwendungsgebiet. (Ergebnis der Entwicklerbefragung)
- Zu Benutzervertretern werden häufig Kundenverantwortliche bestimmt, die sich wenig mit den Arbeitsabläufen auskennen, die durch die Software unterstützt werden sollen. (Schlussfolgerung aus mehreren Teilergebnissen)
- Softwareanforderungen werden sehr allgemein mit Benutzergruppen oder Abteilungen verknüpft. Die mangelnde Verfolgbarkeit des Ursprungs einer Anforderung erschwert es, schnell geeignete Ansprechpartner zu finden. (Ergebnis der Entwicklerbefragung)

Informationen zur Literaturquelle		
Nr.: Q3	**Kurztitel:** Bhandari u. a. /Improvement/	**Jahr der Veröffentlichung**: 1994

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 182 f.)

- Kann die Orthogonal Defect Classification (ODC) gemeinsam mit Attribute Focusing bei verschiedenen Arten von Softwareentwicklungsprojekten erfolgreich angewandt werden kann?
- Welchen Nutzen liefert ODC?
- Untersuchungseinheit: ODC in Anwendung.

Vorgehensweise der Untersuchung (S. 183, 185 f.)

- Rückblickende Darstellung der Erfahrungen und quantitativen Daten aus 7 abgeschlossenen Softwareentwicklungsprojekten, in denen ODC angewandt worden ist.
  - Fehler wurden durch Softwareentwickler während der Projekte gemäß ODC erfasst.
  - Die klassifizierten Fehler wurden automatisch mit der Methode „Attribute Focusing“ ausgewertet, um nach Mustern in der Fehlermenge zu suchen. „Attribute Focusing“ zielt darauf ab, schwer erkennbare Zusammenhänge in großen Datenbeständen aufzuspüren und ist daher eine Data-Mining-Methode[799].
- Um den Nutzen von ODC zu beschreiben werden zwei Ansätze verfolgt:
  - Der Nutzen von ODC wird für konkrete Beispiele aus den untersuchten Projekten quantifiziert.
  - Es wird untersucht, ob die Ergebnisse der ODC-Anwendung auch mit zwei anderen Verfahren hätten erlangt werden können. Zum einen ist das die Ursachenanalyse für eine Auswahl von Fehlerberichten, in denen Fehler überwiegend qualitativ beschrieben werden. Zum anderen wird ODC mit dem Einsatz zielorientierter Softwaremetriken verglichen, anhand derer überprüft wird, ob bestimmte Prozessziele erreicht werden.

Subjekte bzw. Objekte der Untersuchung (S. 183 f. und 211)

- 7 Softwareentwicklungsprojekte an sechs geographisch voneinander entfernten Standorten
- Die entwickelten Softwareprodukte zielen auf verschiedene Branchen.
- Softwareentwicklungsprojekte haben unterschiedliche Ausprägungen zu folgenden Merkmalen:
  - angestrebte Hardwareumgebung (Großrechner, Midrange-Rechner, Arbeitsplatzrechner)
  - angestrebte Softwareumgebung (Betriebssystem, Compiler, Datenbank, Anwendung)
  - Umfang des neuen und geänderten Codes (klein, mittel, groß, sehr groß),
  - Anzahl Programmierer und Tester,
  - eingesetzte Softwarewerkzeuge (Verfolgung von Fehlern aus dem Softwaretests, Verfolgung von Daten aus Inspektionen, Verfolgung von Testfällen)
  - eingesetztes Prozessmodell (sequenziell, iterativ oder Kombination aus beiden) und
  - Parallele Durchführung von unterschiedlichen Teilaufgaben für verschiedene Komponenten Software (ja, nein).

Wesentliche Ergebnisse der Untersuchung (S. 186-212)

- Durch die Anwendung von ODC
  - konnten bisher verborgen gebliebene Fehler entfernt,
  - die Einführung bestimmter Fehler vermieden und
  - kurzfristige sowie langfristige Prozessverbesserungspotenziale aufgedeckt werden.
- Durch den Einsatz von ODC in Kombination mit der Data-Mining-Methode „Attribute Focusing“ können in allen untersuchten 7 Projekten erfolgreich Prozessverbesserungspotenziale identifiziert werden.
- Wird ODC früh im Projekt eingesetzt, profitiert man sowohl von der Vermeidung als auch Entdeckung von Fehlern, während bei einem späten Einsatz der Nutzen sich auch die Entdeckung von Fehler beschränkt.

[799] Vgl. Mertens, Wieczorrek /Strategien/ 18 f.

Informationen zur Literaturquelle	
Nr.: Q4 **Kurztitel:** Briand u.a. /Change analysis process/	**Jahr der Veröffentlichung**: 1994

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 38 f.)

- Darstellung eines Verfahrens zur Beschreibung und Bewertung von Softwarewartungsprozessen und seine praktische Anwendung in einem Projekt.
- Untersuchungseinheit: das Verfahren.

Vorgehensweise der Untersuchung (S 38-48)

- Die Vorgehensweise der empirischen Untersuchung wird nicht explizit diskutiert.
- Das Verfahren wird ausführlich beschrieben und anschließend werden Erfahrungen aus einem Projekt dargestellt.

Subjekte bzw. Objekte der Untersuchung (S. 38 f. und 44)

- Wartung einer Software zur Bestimmung einer Umlaufbahn.
- Die Software wird seit 26 Jahren durch eine Abteilung der NASA gewartet.
- Sie hat einen Umfang von 250 Kloc und ist in Fortran geschrieben.

Wesentliche Ergebnisse der Untersuchung (S. 44-49)

- Das qualitative Verfahren kann erfolgreich in dem untersuchten Projekt angewandt werden.

Informationen zur Literaturquelle		
Nr.: Q5	**Kurztitel:** Crowston, Kammerer /Coordination/	**Jahr der Veröffentlichung**: 1998

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 227-230 und 232)

- Können Probleme in der Anforderungsanalyse großer Softwareentwicklungsprojekte durch die Koordinationstheorie nach Malone und Crowston[800] und durch die Collective-Mind-Theorie nach Weick und Roberts[801] erklärt werden?
- Crowston und Kammerer untersuchen in ihrem Beitrag insbesondere zwei Probleme:
  - häufige Änderungen in einer Menge von Softwareanforderungen aufgrund falscher oder fehlender Anforderungen („feature churn“) und
  - inkonsistente Anforderungen bei der Integration umfangreicher Mengen von Softwareanforderungen.
- Untersuchungseinheiten: Koordinationstheorie und Collective-Mind-Theorie.

Vorgehensweise der Untersuchung (S. 231 f. und 238)

- Theoriegeleitete Sammlung von Daten zu Prozessen der Anforderungsanalyse in zwei Projekten verschiedener Unternehmen durch semi-strukturierte Interviews, Analyse von Dokumenten und Beobachtungen von Personen bei der Arbeit.
- In Unternehmen 1 werden 14 Personen befragt, darunter Manager und Entwickler. In Unternehmen 2 werden 19 Interviews mit 22 Personen durchgeführt. Hierbei geben Manager verschiedener Ebenen und Entwickler Auskünfte.
- Auswertung der gesammelten Daten anhand der beiden Theorien.

Subjekte bzw. Objekte der Untersuchung (S. 228-230)

- Jeweils ein Projekt aus 2 verschiedenen Unternehmen. Gemeinsamkeiten dieser Projekte:
  - Entwicklung einer Echtzeit-Software (real-time software), die auf externe Ereignisse innerhalb eines festgelegten Zeitrahmens reagieren können muss.[802]
  - Weiterentwicklungsprojekte, in denen bestehende Komponenten bei Bedarf angepasst und neue Komponenten entwickelt werden.
  - Die fertigen Produkte sind ein System aus Echtzeit-Software und spezifischer Hardware.
  - Die Software muss für verschiedene Kundengruppen angepasst werden.
- Unternehmen 1 (Pseudonym im Beitrag: LGC) ist ein großer Auftragnehmer einer staatlichen Behörde und führt die Anforderungsanalyse für ein System zur Flugssicherung (air traffic control system) durch.
  - Der untersuchte Bereich des Unternehmens umfasst 300 bis 400 Mitarbeiter und betreut nur das berücksichtigte Projekt.
  - Ergebnisse der Anforderungsanalyse werden nach Abschluss einem weiteren Auftragnehmer zur Implementierung der Software übergeben.
  - Entwicklung von verschiedenen Prototypen, anhand derer Softwareanforderungen validiert und verfeinert werden.
  - Problem: Inkonsistente Anforderungen bei der Integration umfangreicher Mengen von Softwareanforderungen.
- Unternehmen 2 (Pseudonym im Beitrag: TC) ist ein multinationales Telekommunikationsunternehmen. Der untersuchte Bereich entwickelt eine Kontrollsoftware zur Steuerung der Kommunikation in Telefonnetzen.
  - Die Software umfasst 5 Millionen Codezeilen. 200 Personen wirken an der Anforderungsanalyse mit.
  - Die Anforderungsanalyse wird für jeden Abnehmer der Software einzeln durchgeführt. Ausgehend von einer bestehenden Menge von Softwareanforderungen werden Anforderungen für jeden Abnehmer angepasst.

---

[800] Vgl. Malone, Crowston /Coordination/ 88-111.

[801] Vgl. Weick, Roberts /Collective mind/ 357-378.

[802] Vgl. IEEE /Glossary/ 63.

- Problem: häufige Änderungen in einer Menge von Softwareanforderungen aufgrund falscher oder fehlender Anforderungen.

Wesentliche Ergebnisse der Untersuchung (S. 232-244, insbesondere 237 und 243 f.)

- Anhand der Koordinationstheorie von Malone und Crowston kann aufgezeigt werden, welche Abhängigkeiten zwischen Aufgaben, zwischen benötigten Ressourcen und Aufgaben sowie zwischen genutzten Ressourcen bestehen. Diese Abhängigkeiten müssen durch Koordinationsmechanismen bewusst gestaltet werden, um Koordinationsprobleme zu vermeiden.
- Mittels der Koordinationstheorie können in der Untersuchung bestimmte Probleme in der Anforderungsanalyse erklärt werden. Sie leistet jedoch keine Hilfestellung dabei, Lösungen für diese Probleme zu entwickeln.
- Die Koordinationstheorie berücksichtigt keine ebenso wichtigen Probleme, die mit der Entwicklung eines gemeinsamen Verständnisses und kollektiven Sinngebungsprozesses in der Anforderungsanalyse zusammenhängen.
- Die Collective-Mind-Theorie nach Weick und Roberts erklärt, wie Voraussetzungen geschaffen werden können, damit eine Gruppe ein gemeinsames Verständnis über die Aufgaben der Gruppe gewinnt.
- Beide Theorien können komplementär genutzt werden, um Probleme in der Anforderungsanalyse zu erklären.

Informationen zur Literaturquelle		
Nr.: Q6	**Kurztitel:** Curtis, Krasner, Iscoe /Field study/	**Jahr der Veröffentlichung**: 1988

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 1268)

- Welche Probleme treten in großen Softwareentwicklungen auf, wenn Entscheidungen in der Anforderungsanalyse und dem Entwurf getroffen, abgebildet, kommuniziert und geändert werden?
- Welche Ursachen liegen diesen Problemen zugrunde?
- Untersuchungseinheit: Probleme in der Anforderungsanalyse und dem Entwurf.

Vorgehensweise der Untersuchung (S. 1269-1271 und 1285 f.)

- Durchführung von strukturierten Interviews. Interviews werden durch zwei Personen durchgeführt und aufgezeichnet.
- Erstellung schriftlicher Protokolle der Interviews mit einem Umfang von über 3000 Seiten.
- Systematische Auswertung der Protokolle.
- Auswertung und Darstellung orientiert sich an einem Verhaltensmodell mit mehreren Ebenen: Individuum, Team, Projekt, Unternehmen, geschäftliches Umfeld.

Subjekte bzw. Objekte der Untersuchung (S. 1269 f.)

- In 17 Projekten von 9 Unternehmen werden 97 Interviews mit Softwareentwicklern, Projektmanagern und z. T. Abteilungsleitern sowie Kundenvertretern durchgeführt.
- Die Projekte befinden sich in verschiedenen Phasen: Anforderungsanalyse, Entwurf, Implementierung oder Wartung. Ein Projekt wurde abgebrochen.
- Umfang, Anwendungsgebiet und Art der entwickelten Software sind sehr unterschiedlich.

Wesentliche Ergebnisse der Untersuchung (S. 1270-1285)

- Drei Probleme treten verstärkt in großen Softwareentwicklungen auf, wenn Entscheidungen in der Anforderungsanalyse und dem Entwurf getroffen, abgebildet, kommuniziert und geändert werden:
  - Geringe Verbreitung von Wissen über das Anwendungsgebiet,
  - sich ändernde und inkonsistente Softwareanforderungen und
  - Versäumnisse in der Kommunikation und Koordination.
- Für alle Ebenen des Verhaltsmodells werden auf Grundlage der Interviewergebnisse die Ursachen dieser Probleme erörtert.

Informationen zur Literaturquelle	
Nr.: Q7 **Kurztitel:** Damian u. a. /Case study/	**Jahr der Veröffentlichung**: 2004

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 46 und 51 f.)

- Darstellung einer Prozessverbesserung der Anforderungsanalyse in einer Organisation.
- Untersuchungseinheit: Prozessverbesserung der Anforderungsanalyse.

Vorgehensweise der Untersuchung (S. 49 f.)

- Datenerhebung durch zwei Fragebögen, Inspektion von Anforderungsdokumenten, Beobachtung von Sitzungen zur Anforderungsanalyse und semi-strukturierte Interviews.
- Darstellung der Bemühungen, den Prozess der Anforderungsanalyse zu verbessern.
- Nach Abschluss der Prozessänderungen bewerten die Projektteilnehmer die Prozessmaßnahmen durch jeweils einen Fragebogen zu Beginn und nach Abschluss der Anforderungsanalyse.

Subjekte bzw. Objekte der Untersuchung (S.46 und 49 f.)

- Das Australian Center for Unisys Software (ACUS) ist ein Entwicklungslabor von Unisys. Diese Organisationseinheit entwickelt Software und ist nach ISO-9001 zertifiziert.
- 34 Personen von ACUS werden in den Fragebögen, Interviews und Sitzungen einbezogen.
- Es wurden ausschließlich Personen ausgewählt, die an vorherigen Projekten bei ACUS teilgenommen haben und zum Zeitpunkt der Untersuchung auch Mitarbeiter eines Projektes sind.

Wesentliche Ergebnisse der Untersuchung (S. 58-68)

- Die Anpassungen im Prozess der Anforderungsanalyse führen zu einem besseren Verständnis der Softwareanforderungen und einer verbesserten Fähigkeit auf Kundenbedürfnisse einzugehen.
- Zugleich verbessern die Prozessanpassungen die Zusammenarbeit der Gruppen mit Personen aus verschiedenen Funktionsbereichen.

Informationen zur Literaturquelle		
Nr.: Q8	**Kurztitel:** Damian u. a. /Requirements engineering/	**Jahr der Veröffentlichung**: 2005

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 256 f. und 261 f.)

- Stellt sich der in der Untersuchung Q7 durch die Teilnehmer wahrgenommene langfristige Nutzen einer Prozessverbesserung der Anforderungsanalyse wirklich ein?
- Fortsetzung der Untersuchung, welche in Damian u. a. /Case study/ (Q7) beschrieben ist.
- Untersuchungseinheit: langfristiger Nutzen einer Prozessverbesserung der Anforderungsanalyse.

Vorgehensweise der Untersuchung (S. 262-264)

- Datenerhebung durch einen Fragebogen, semi-strukturierten Interviews und Inspektion von Dokumenten.
- Folgende Dokumente werden u. a. berücksichtigt:
  - Dokumente zum Prozess der Anforderungsanalyse,
  - Anforderungsdokumente,
  - Änderungsanträge zu Softwareanforderungen und
  - Einträge in einer Software für das Anforderungsmanagement.

Subjekte bzw. Objekte der Untersuchung (S. 259 f. und 262)

- Das Australian Center for Unisys Software (ACUS) ist ein Entwicklungslabor von Unisys. Diese Organisationseinheit entwickelt Software und ist nach ISO-9001 zertifiziert.
- 31 Personen aus verschiedenen Bereichen von ACUS nehmen an der Untersuchung teil. Die meisten Personen sind mehrjährige Mitarbeiter der Organisation und kennen über 15 Jahre die gelebte Praxis der Anforderungsanalyse in dieser Organisation.

Wesentliche Ergebnisse der Untersuchung (S. 265-274)

Die Verbesserung des Prozesses der Anforderungsanalyse bei ACUS erzeugt einen langfristigen Nutzen. Im Einzelnen werden folgende positive Wirkungen der Prozessänderung festgestellt:

- Das Verständnis der Details, Abhängigkeiten und Komplexität von Softwareanforderungen verbessert sich.
- Der Umfang vermeidbarer Überarbeitungen (rework) nimmt ab.
- Die Kommunikation im Entwicklungsteam als auch zwischen verschiedenen Teams verbessert sich.
- Entwickler sind besser in der Lage, fundierte Entscheidungen zu treffen.
- Die Genauigkeit von Schätzungen zum Aufwand und zur Fehlerhäufigkeit erhöht sich.
- Änderungen von Softwareanforderungen werden unter Berücksichtigung ihrer Auswirkungen kontrollierter durchgeführt.

Informationen zur Literaturquelle

Nr.: Q9 **Kurztitel:** Damian, Zowghi /Requirements engineering/ **Jahr der Veröffentlichung**: 2003

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 150)

- Welche Auswirkung hat die geographische Verteilung von Projektbeteiligten auf die Aktivitäten der Anforderungsanalyse in einer weltweit verteilten Softwareentwicklung?
  - Wie werden in einer verteilten Softwareentwicklung Anforderungen erarbeitet?
  - Welche Probleme treten dabei auf?
  - Welche „Strategien" und Technologien werden eingesetzt, um diese Probleme zu bewältigen?
- Untersuchungseinheit: Prozess der Anforderungsanalyse in einer verteilten Softwareentwicklung.

Vorgehensweise der Untersuchung (S. 150)

- Einsatz der Fallstudienmethode in einem global verteilten Softwareentwicklungsprojekt.
- Datenerhebung durch Inspektion von Dokumenten, Beobachtung von Sitzungen zur Anforderungsanalyse, und semi-strukturierten Interviews.
- Interviews finden an vier Standorten in den USA, drei Standorten in Australien, und jeweils einem Standort in Neuseeland und Europa statt.
- Um die Praxis der Anforderungsanalyse in dem Softwareentwicklungsprojekt zu analysieren, setzen die Autorinnen die Soft Systems Methodology[803] (SSM) von Peter Checkland ein
- Die Ergebnisse der Interviews werden mit Techniken der Grounded Theory[804] ausgewertet.

Subjekte bzw. Objekte der Untersuchung (S. 150)

- Die Autorinnen untersuchen die geographisch verteilte Softwareentwicklung eines internationalen Unternehmens mit Sitz in den USA.
- Bei dem Softwareentwicklungsvorhaben handelt es sich um die Wartung einer Software, die stetig in Form von einzelnen Versionen weiterentwickelt wird.
- Jede neue Version der Software umfasst ungefähr 8000 KLOC und benötigt zwischen 12 bis 18 Monaten Entwicklungszeit. Circa 120 Entwickler arbeiten Vollzeit an dem Projekt.
- Die Software ist selbst ein Werkzeug, um Software zu entwickeln, und wird für den Markt produziert.

Wesentliche Ergebnisse der Untersuchung (S. 153-158)

- Die Autorinnen erarbeiten einen theoretischen Bezugsrahmen für Herausforderungen der Anforderungsanalyse, die durch geographische Verteilung von Projektbeteiligten verursacht wird.
- Der Bezugsrahmen hat drei Ebenen:
  - allgemeine Probleme, die sich aufgrund der geographischen Verteilung ergeben: unzureichende Kommunikation, unzureichendes Wissensmanagement, kulturelle Vielfalt, unterschiedliche Zeitzonen
  - Die allgemeinen Probleme führen zu spezifischen Problemen in der Anforderungsanalyse: kulturelle Vielfalt, unangemessene Beteiligung von Benutzern und lokalen Mitarbeitern zur Anwendungsberatung, Mangel an informeller Kommunikation und ein reduziertes Bewusstsein für das lokale Arbeitsumfeld, geringeres Vertrauen, ineffektive Entscheidungssitzungen, Verzögerungen.
  - Auf der dritten Ebene werden Teilaufgaben der Anforderungsanalyse genannt, die durch die Probleme der zweiten Ebene beeinträchtigt werden.

[803] Vgl. Checkland /Systems Thinking/.

[804] Vgl. Strauss, Corbin /Grounded theory/.

Informationen zur Literaturquelle	
Nr.: Q10 **Kurztitel:** Daneva /ERP/	**Jahr der Veröffentlichung**: 2004

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 26)

- Darstellung von Problemen und Lösungen im Prozess der Anforderungsanalyse bei der Entwicklung und Wartung von Enterprise-Resource-Planning-Software (ERP-Software).
- Untersuchungseinheit: Prozess der Anforderungsanalyse bei der Entwicklung und Wartung von ERP-Software.

Vorgehensweise der Untersuchung (S. 26)

- Die Erfahrungen zum Prozess der Anforderungsanalyse aus mehreren abgeschlossenen ERP-Projekten werden rückblickend dargestellt.
- Die Prozesse der Anforderungsanalyse orientieren sich an Empfehlungen des Accelerated-SAP (ASAP) und dem Requirements Engineering Good Practice Guide (REGPG) von Sommerville und Sawyer.[805]

Subjekte bzw. Objekte der Untersuchung (S. 26 f.)

- Darstellung der Erfahrungen aus 13 abgeschlossenen ERP-Projekten bei Telus Mobility, einem kanadischen Telekommunikationsunternehmen. Darunter sind 6 Neuentwicklungen.
- Jedes Projekt ist in Teilprojekte aufgegliedert, in denen jeweils separat die Anforderungsanalyse durchgeführt werden. Insgesamt werden 67 derartige Teilprojekte berücksichtigt.

Wesentliche Ergebnisse der Untersuchung (S. 28-33)

- Die Erfahrungen werden in Form von gelernten Lektionen dargestellt und nach Problemkategorien gruppiert.

[805] Vgl. Daneva /ERP/ 26 und die dort aufgeführten Literaturquellen.

Informationen zur Literaturquelle	
Nr.: Q11 **Kurztitel:** Ebert, De Man /Requirements uncertainty/	**Jahr der Veröffentlichung**: 2005

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 553 und 555)

- Was sind die Ursachen für unsichere Softwareanforderungen? (explorativer[806] Teil der Untersuchung)
- Können durch bestimmte Maßnahmen Verzögerungen in einem Projekt vermieden werden, die durch unsichere Softwareanforderungen verursacht werden? (explanatorischer[807] Teil der Untersuchung)
- Untersuchungseinheit: Ursachen für Unsicherheit bei Softwareanforderungen und Effektivität ausgewählter Maßnahmen zur Vermeidung von Projektverzögerungen.

Vorgehensweise der Untersuchung (S. 555 f.)

- Datenerhebung durch Interviews und Auswertung von Projektabschlussberichten (lessons learned reports).
- Für den explanatorischen Teil der Untersuchung werden statistische Methoden eingesetzt.

Subjekte bzw. Objekte der Untersuchung (S. 554.)

- 246 Entwicklungsprojekte bei Alcatel, die zwischen dem Januar 2002 und Oktober 2003 abgeschlossen worden sind.
- In den Entwicklungsprojekten wird überwiegend eingebettete Software entwickelt.
- Aus den 246 Entwicklungsprojekten werden jene ausgewählt, zu denen Informationen zum Projekt zur Verfügung stehen. Die Autoren lassen leider unbeantwortet, wie viele dieses Kriterium erfüllen.
- Bestimmte Auswertungen werden beispielhaft für 15 Projekte dargestellt.

Wesentliche Ergebnisse der Untersuchung (S. 555-558)

- Die Ursachen für unsichere Anforderungen sind:
  - eine vage Produktvision und –strategie,
  - Schlüsselpersonen (key stakeholders) werden nicht in das Projekt einbezogen,
  - unbekannte Projektabhängigkeiten (z. B. zu anderen Projekten),
  - die betriebswirtschaftlichen Auswirkungen eines Projektes (der „Business Case“) sind nicht gründlich ausgearbeitet und
  - Softwareanforderungen werden nicht ausreichend spezifiziert und analysiert.
- Die Autoren schlagen vier Maßnahmen vor, um Probleme mit unsicheren Anforderungen zu überwinden:
  - Aufbau eines Kernteams mit Schlüsselpersonen bestehend aus einem Produktmanager, einem Marketingmanager und einem technischen Projektmanager.
  - Produktlebenszyklus mit frühen Meilensteinen.
  - Softwareanforderungen online zugänglich und verfolgbar machen.
  - Softwareanforderungen validieren (Prüfen gegen den „Business Case“).
- Die Autoren stellen einen signifikanten Zusammenhang zwischen Projekten mit keinen oder geringen Verspätungen und dem Einsatz von drei oder vier der Maßnahmen fest.
- Werden nur eine oder zwei der Maßnahmen eingesetzt, gilt dieser Zusammenhang jedoch nicht.

[806] Vgl. zum Begriff „explorativ“ Kapitel 4.3.3.3.2.

[807] Vgl. zum Begriff „explanatorisch“ Kapitel 4.3.3.3.2.

Informationen zur Literaturquelle

Nr.: Q12 **Kurztitel:** Günther, Rombach, Ruhe /Qualitätsverbesserung/ **Jahr der Veröffentlichung**: 1996

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 166 f.)

- Beschreibung eines Projekts über GQM-basierte Messprogramme bei der Allianz Lebensversichungs-AG.
- Ziele der Untersuchung werden nicht explizit angegeben.
- Es bleibt offen, was genau die Untersuchungseinheit ist: die Einführung von GQM-basierten Meßprogrammen, die Effektivität von GQM-basierten Meßprogrammen oder bestimmte Merkmale von Softwareentwicklungsprojekte bei der Allianz Lebensversicherungs-AG?

Vorgehensweise der Untersuchung (S. 166 f.)

- In 5 Pilotprojekten werden GQM-basierte Messprogramme eingeführt.
- Die Messprogramme verfolgen jeweils verschiedene Zielsetzungen, um ausgewählte Aspekte der Softwareentwicklungsprozesse zu verstehen und zu bewerten. Zielsetzungen sind z. B.
  - Wie verteilt sich der Aufwand auf Entwicklungsphasen? Welchen Nutzen bringen eingesetzte Werkzeuge und Methoden?
  - Wie stabil sind Softwareanforderungen während Anforderungsanalyse und Entwurf?
- Projektbegleitende Erfassung der Messdaten auf der Grundlage von Messplänen.
- Analyse und Interpretation der Messergebnisse durch Interviews mit Projektbeteiligten.

Subjekte bzw. Objekte der Untersuchung (S. 166 f.)

- 5 Softwareprojekte der Allianz-Lebensversicherungs-AG, in denen jeweils Anwendungssoftware entwickelt wird.

Wesentliche Ergebnisse der Untersuchung (S. 168-170)

- Die Messprogramme führen zu zahlreichen Einzelerkenntnissen, die in der Literaturquelle nach folgenden Bereichen gruppiert werden:
  - verbessertes Know-how,
  - verbessertes Verständnis,
  - Bestätigung, Widerlegung und Neuaufstellen von Hypothesen,
  - Identifikation von Problembereichen und
  - Erkennen von Ansätzen zur Verbesserung.
- Fachliche Erfahrungen über projektrelevante Geschäftsprozesse haben einen entscheidenden Einfluss auf die Stabilität von Softwareanforderungen.
- Vollständige, eindeutige und durchführbare Zielvorgaben für eine Softwareentwicklung fördern die Stabilität von Softwareanforderungen.
- „Kopfmonopole“ an kritischen Stellen führen zur Instabilität von Softwareanforderungen.
- Beeinflussung durch parallele Projekte führen ebenfalls zur Instabilität von Softwareanforderungen.

Anmerkungen

Diese Literaturquelle referenziert mehrere nicht veröffentlichte, interne Arbeitspapiere, die vermutlich die Untersuchung detailliert beschreiben.

Informationen zur Literaturquelle		
Nr.: Q13	**Kurztitel:** Hofmann, Lehner /Requirements engineering/	**Jahr der Veröffentlichung**: 2001

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 58)

- Welche Maßnahmen der Anforderungsanalyse tragen zum Erfolg der Anforderungsanalyse bei?
- Untersuchungseinheit: Einfluss einzelner Maßnahmen der Anforderungsanalyse auf den Erfolg.

Vorgehensweise der Untersuchung (S. 60)

- Datenerhebung durch einen Fragebogen mit Likert-Skalen und anschließenden Interviews.
- Das Maß für den Erfolg der Anforderungsanalyse ist angelehnt an eine Untersuchung von El Emam und Madhavji. Es berücksichtigt drei Aspekte:[808]
  - die Qualität des Prozesses der Anforderungsanalyse,
  - die Qualität der Produkte der Anforderungsanalyse und
  - die Prozesskontrolle (Grad der Übereinstimmung der tatsächlichen Werte für Dauer, Aufwand und Kosten mit geplanten Werten)

Subjekte bzw. Objekte der Untersuchung (S. 58, 60)

- 15 Teams der Anforderungsanalyse wurden untersucht. 6 der Teams entwickeln Standardsoftware und 9 Individualsoftware. Es handelt sich jeweils um geschäftskritische, betriebliche Anwendungssoftware.
- Durchschnittlich dauern die untersuchten Softwareprojekte 16,5 Monate und haben einen Aufwand von ungefähr 120 Personenmonaten.
- Die Teams stammen aus 9 Organisationen aus der Telekommunikations- oder Bankenbranche.
- Insgesamt wurden 76 Personen befragt. Darunter 15 Projektmanager, 34 Teammitglieder, 27 sonstige Projektbeteiligte (z. B. Kunden, Management, Mitarbeiter für die Qualitätssicherung).

Wesentliche Ergebnisse der Untersuchung (S. 61-65)

Erfolgreiche Teams

- haben weitgehendes Wissen bzgl. des Anwendungsgebiets der Software, der Technik und des eingesetzten Prozesses der Anforderungsanalyse,
- binden Kunden in die Anforderungsanalyse ein,
- investieren mehr Ressourcen in die Anforderungsanalyse,
- spezifizieren Anforderungen durch mehrere Modelle und setzen Prototypen ein,
- stellen die Verfolgbarkeit vom Ursprung einer Anforderung bis zu ihrer Implementierung sicher.

[808] Vgl. Hofmann, Lehner /Requirements engineering/ 63 und die dort aufgeführte Literaturquelle.

Informationen zur Literaturquelle	
Nr.: Q14 **Kurztitel:** Kauppinen u. a. /Cultural change/	**Jahr der Veröffentlichung**: 2002

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 43)

- Die Autoren gehen von der Annahme aus, dass eine grundlegende Prozessverbesserung der Anforderungsanalyse mit einem kulturellen Wandel der Entwickler einhergehen muss.
- Auf dieser Annahme aufbauend führen sie eine Untersuchung unter folgender Fragestellung durch: Welche Faktoren unterstützen einen entsprechenden kulturellen Wandel und welche verhindern ihn?
- Untersuchungseinheit: kultureller Wandel im Kontext der Anforderungsanalyse.

Vorgehensweise der Untersuchung (S. 43 f.)

- Datenerhebung durch Beobachtungen, informellen Gesprächen, formellen Interviews, offiziellen Sitzungen und Dokumentenanalyse.
- Zur Bewertung der Prozesse der Anforderungsanalyse wurde der Ansatz des „Requirements Engineering Good Practice Guide“ von Sommerville und Sawyer gewählt.[809]

Subjekte bzw. Objekte der Untersuchung (S. 44)

- 4 softwareentwickelnde Organisationseinheiten mittelgroßer und großer Unternehmen aus Finnland.
- Diese Organisationseinheiten haben 25 bis 160 Mitarbeiter und entwickeln Software überwiegend für den Markt.
- Das Anwendungsgebiet der Software ist in den vier Organisationseinheiten jeweils unterschiedlich.

Wesentliche Ergebnisse der Untersuchung (S. 45-49)

- Faktoren, die einen kulturellen Wandel unterstützen:
  - Die Verknüpfung von Unternehmenszielen mit technischen Anforderungen über Kundenbedürfnisse und Softwareanforderungen.
  - Integration eines einfachen Prozesses der Anforderungsanalyse in den Entwicklungsprozess.
  - Aktive Erhebung von Kundenbedürfnissen.
  - Systematische Darstellung von Softwareanforderungen in Form von Anwendungsfällen („use cases“).
  - Das Bewusstsein der Projektbeteiligten für die Anforderungsanalyse erhöhen.
- Faktoren, die einen kulturellen Wandel verhindern:
  - Überzeugung, dass es sich nicht lohnt, die Bedürfnisse direkt von den Benutzern zu ermitteln.
  - Überzeugung, dass es schwierig ist, die Bedürfnisse direkt von den Benutzern zu ermitteln.
  - Überzeugung, dass es riskant ist, die Bedürfnisse direkt von den Benutzern zu ermitteln.
  - Überzeugung, dass es sich nicht lohnt, Softwareanforderungen systematisch zu dokumentieren.

[809] Vgl. Kauppinen u. a. /Cultural change/ 44 und die dort aufgeführte Literatur.

Informationen zur Literaturquelle	
Nr.: Q15 **Kurztitel:** Lauesen, Vinter /Requirement defects/	**Jahr der Veröffentlichung**: 2001

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 38)

- Autoren beschreiben eine Untersuchung, die das Ziel hat, Maßnahmen zu finden, anhand derer Anforderungsfehler mit geringen Kosten effektiv vermieden werden sollen.
- Es bleibt offen, was genau die Untersuchungseinheit ist: die Vorgehensweise, entsprechende Maßnahmen zu finden, oder die Maßnahmen selbst.

Vorgehensweise der Untersuchung (S. 38)

- Autoren werten Fehlerberichte einer eingebetteten Software aus. In diesen Fehlerberichten sind Fehler gemäß einem Klassifikationsschema nach Beizer erfasst.[810]
- Für ausgewählte Maßnahmen zur Vermeidung von Anforderungsfehlern wird bestimmt,
  - wie viele Fehler sie hätten verhindern können,
  - wie viele Personenstunden dadurch eingespart worden wären und
  - welche Kosten sie verursacht hätten.
- Zwei Maßnahmen werden in zwei Pilotprojekten angewandt: Beschreibung von Szenarios und Benutzbarkeitstest der Benutzeroberfläche anhand von Prototypen.

Subjekte bzw. Objekte der Untersuchung (S. 40-42.)

- Die Untersuchung wird in dem dänischen Unternehmen Brüel & Kjaer (B&K) durchgeführt, das Ausrüstungen zur Messung von Tönen und Schwingungen herstellt. Die Hälfte der Mitarbeiter sind Softwareentwickler.
- 200 von 800 Fehlerberichten einer eingebetteten Software werden ausgewertet. 107 von den 200 Fehlerberichten haben Probleme mit Anforderungen zum Gegenstand. Ungefähr 70% der 107 Fehlerberichte betreffen Probleme der Benutzbarkeit.
- Die eingebettete Software ist in C++ entwickelt und umfasst 90 000 Codezeilen. Der Entwicklungsaufwand beträgt 77 Personenmonate.
- Die zwei Verbesserungsmaßnahmen (s. o.) werden in zwei anderen Projekten des Unternehmens eingesetzt.

Wesentliche Ergebnisse der Untersuchung (S. 47-48)

- Die Anzahl der Benutzbarkeitsfehler kann durch die zwei Maßnahmen um 70% reduziert werden.

[810] Vgl. Lauesen, Vinter /Requirement defects/ 40 f. und die dort aufgeführte Literatur.

Informationen zur Literaturquelle	
Nr.: Q16	**Kurztitel:** Lubars, Potts, Richter /Requirements modeling/ **Jahr der Veröffentlichung**: 1993

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 2)

- Wie wird die Anforderungsanalyse in der Praxis durchgeführt?
- Untersuchungseinheit: Stand der Anforderungsanalyse in der Praxis.

Vorgehensweise der Untersuchung (S. 2 f.)

- Strukturierte Interviews mit Softwareentwicklern.

Subjekte bzw. Objekte der Untersuchung (S. 2-4)

- Zu insgesamt 23 Projekten aus 10 US-amerikanischen Firmen werden 35 Interviews durchgeführt. Insgesamt werden 87 Projektmitglieder befragt.
- 6 Projekte sind interne Entwicklungen oder Marktentwicklungen. Die restlichen 17 Projekte, sind Kundenprojekte. Aus letzteren sollen bei 8 Projekten zukünftig Produkte für den Markt entstehen (Pilotkundenprojekte). Deshalb erfolgt in der Untersuchung eine Unterteilung in 14 marktspezifische Projekte und 9 kundenspezifische Projekte.
- In den Projekten wird jeweils Software für sehr unterschiedliche Anwendungsgebiete entwickelt: Avionik-Software, Software für den öffentlichen Bereich, Telekommunikationssoftware, etc.

Wesentliche Ergebnisse der Untersuchung (S. 4-11)

- Die qualitativen Ergebnisse geben einen Überblick über den Stand der Anforderungsanalyse in folgenden Bereichen:
  - Erhebung von Anforderungen,
  - Dokumentation von Anforderungen,
  - Validierung von Anforderungen,
  - Systemevolution und
  - Projektmanagement und –planung.

Informationen zur Literaturquelle

Nr.: Q17 **Kurztitel:** Morris, Masera, Wilikens /Requirements Engineering/ **Jahr der Veröffentlichung**: 1998

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 130)

- Welche Probleme treten auf, wenn aus Forschungs- und Entwicklungsprojekten zur Anforderungsanalyse gewonnene Erkenntnisse in die Praxis übertragen werden sollen? Welche Priorität haben diese Probleme jeweils?
- Welche Maßnahmen können die Übertragung dieser Erkenntnisse in die Praxis verbessern? Welche Priorität haben diese Maßnahmen jeweils?
- Untersuchungseinheit: Übertragung von Erkenntnissen zur Anforderungsanalyse aus der Forschung in die Praxis.

Vorgehensweise der Untersuchung (S. 130)

Arbeitskreis mit vier Sitzungen. Die Sitzungen haben jeweils folgende Schwerpunkte:

- 1. Sitzung: Forscher und Praktiker erheben Probleme, die auftreten, wenn Forschungserkenntnisse zur Anforderungsanalyse in die Praxis übertragen werden sollen, und Maßnahme zur Verbesserung der Übertragung.
- 2. Sitzung: Konsolidierung der Liste mit Problemen und Maßnahmen.
- 3. Sitzung: Teilnehmer, die auf den Transfer von Erkenntnissen von der Forschung in die Praxis spezialisiert sind, schlagen aus ihrer Perspektive Verbesserungsmaßnahmen vor.
- 4. Sitzung: Bewertung der Probleme und Maßnahmen hinsichtlich ihrer Wichtigkeit.

Subjekte bzw. Objekte der Untersuchung (S. 130)

Der Arbeitskreis besteht aus

- 10 Teilnehmern, die an früheren europäischen oder nationalen Forschungsprojekten zur Anforderungsanalyse mitgewirkt haben,
- 3 Teilnehmern, die auf den Transfer von Erkenntnissen aus der Forschung in die Praxis spezialisiert sind, und
- 10 Teilnehmern, die Praktiker und potenzielle Kunden für Forschungserkenntnisse zur Anforderungsanalyse sind.

Wesentliche Ergebnisse der Untersuchung (S. 131-137)

- Die vier wichtigsten Probleme bei der Übertragung von Forschungserkenntnissen zur Anforderungsanalyse in die Praxis sind:
  - Unzureichendes Training beim Einsatz neuer Techniken zur Anforderungsanalyse,
  - inhärente Komplexität der Anforderungsanalyse,
  - mangelnde Integration der Anforderungsanalyse in die Unternehmensorganisation und
  - eine Unternehmenskultur, die der Anforderungsanalyse nicht die notwendige Bedeutung beimisst.
- Die vier wichtigsten Maßnahmen zur Verbesserung der Übertragung von Forschungsergebnissen in die Praxis sind:
  - Stärkung der Anforderungsanalyse als Disziplin durch Schulungen, Pilotprojekten und Kontaktnetzwerken,
  - Erhöhung der Flexibilität von Forschungsprojekten,
  - deutlichere Ausrichtung der Forschungsprojekte auf Praxisprobleme in der Anforderungsanalyse und Berücksichtigung dieser Ausrichtung bei der Bewilligung von Forschungsprojekten und
  - Wahrung der Relevanz des Forschungsprojekts während der Durchführung.

Informationen zur Literaturquelle	
Nr.: Q18 **Kurztitel:** Nakajo, Kume /Case history analysis/	**Jahr der Veröffentlichung**: 1991

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 830)

- Autoren beschreiben eine Untersuchung, in der Fehler anhand eines beschriebenen Verfahrens analysiert werden, um ihre Ursache im Prozess zu identifizieren.
- Es bleibt offen, was genau die Untersuchungseinheit ist: das fehlerbasierte Verfahren oder die Ergebnisse ihrer Anwendung, d. h. die Ursache-Wirkungszusammenhänge zwischen Prozessmaßnahmen und Fehlern.

Vorgehensweise der Untersuchung (S. 831)

- Das fehlerbasierte Verfahren unterscheidet drei Elemente: Softwarefehler, menschliche Irrtümer und Prozessmängel.
- Zu vier kommerziellen Softwareprodukten werden Daten zu diesen Elementen durch Dokumentenanalysen erhoben (Softwarecode und Entwicklungsdokumente). Interviews mit den Softwareentwicklern, die einen Fehler jeweils eingeführt haben, vervollständigen die Datenerhebung.

Subjekte bzw. Objekte der Untersuchung (S. 831)

- Das fehlerbasierte Verfahren wird auf 670 Fehler angewandt, die während der Softwareentwicklung von vier kommerziellen Softwareprodukten aufgetreten sind.
- Es handelt sich jeweils um eingebettete Software.

Wesentliche Ergebnisse der Untersuchung (S. 831-837)

- Mit der Anwendung ihres Verfahrens können die Autoren vier Ursache-Wirkungs-Mechanismen zwischen Prozessmaßnahmen und Fehlern in den untersuchten Softwareprodukten ermitteln.
- Die Autoren stellen zusätzlich fest, dass Methoden der Strukturierten Analyse und des Strukturierten Entwurfs Fehler reduzieren können.

Informationen zur Literaturquelle

Nr.: Q19 **Kurztitel:** Sutcliffe, Economou, Markis /Requirements errors/ **Jahr der Veröffentlichung**: 1999

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 135)

- Autoren untersuchen eine Software zur Entscheidungsunterstützung für das griechische Gesundheitsministerium unter folgenden Fragestellungen:
  - Warum sind die Kunden unter funktionalen und qualitativen Gesichtspunkten mit der entwickelten Software unzufrieden?
  - Was sind die Ursachen im Prozess der Anforderungsanalyse, die zu den funktionalen und qualitativen Unzulänglichkeiten der Software geführt haben?
- Untersuchungseinheit: Ursache-Wirkungsmechanismen zwischen Prozessmängeln in der Anforderungsanalyse und Fehlern in der Software.

Vorgehensweise der Untersuchung (S. 136 f.)

- Datenerhebung durch eine Analyse der Benutzbarkeit der Software, semi-strukturierten Interviews und der Analyse von verschiedenen Entwicklungsdokumenten.

Subjekte bzw. Objekte der Untersuchung (S. 135 f.)

- Eine neu entwickelte Software zur Entscheidungsunterstützung für das griechische Gesundheitsministerium wird untersucht. Sie ersetzt ein bestehendes Altsystem.
- Interessensgruppen dieser Software sind:
  - 6 primäre Endbenutzer („health planners“) im griechischen Gesundheitsministerium,
  - ungefähr 25 Abteilungsleiter und Mitarbeiter der Verwaltung, die ausgewählte Informationen abfragen,
  - Privatunternehmen der Gesundheitsbranche (insbesondere Hersteller von Arzneimitteln und medizinischen Geräten),
  - sonstigen Interessenten wie Beraternund Forscher,
  - der Gesundheitsrat des Ministeriums, der dafür verantwortlich ist, Informationen zu Gesundheitsfragen zu beantworten, um die Bedürfnisse der primären und sekundären Benutzer zufriedenzustellen, und
  - der Informatik-Abteilung des Gesundheitsrats.
- Die Software wird durch den Gesundheitsrat des Ministeriums finanziert und durch einem externen Auftragnehmer entwickelt.

Wesentliche Ergebnisse der Untersuchung (S. 831-837)

- Die entwickelte Software soll ein Altsystem ersetzen und dabei im Wesentlichen die Benutzbarkeit verbessern.
- Aufgrund verschiedener Prozessmängel der Anforderungsanalyse wird diese Bedeutung der Benutzbarkeit durch die Entwickler zu spät wahrgenommen.

Informationen zur Literaturquelle	
Nr.: Q20 **Kurztitel:** Takahashi u. a. /Methodologies/	**Jahr der Veröffentlichung**: 1995

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 69)

- Sind Strukturierte Methoden effektiver und effizienter als die natürlich-sprachliche Beschreibung von Anforderungen und Entwurfsentscheidungen?
- Untersuchungseinheit: Effektivität von Methoden zur Beschreibung von Anforderungen und Entwurfsentscheidungen.

Vorgehensweise der Untersuchung (S. 70, 72)

Autoren beschreiben zwei Experimente:

- Experiment 1:
  - Zwei verschiedene Gruppen erhalten die Anforderungen von vier Beispielen einer Anwendungssoftware.
  - Für die erste Gruppe werden die Anforderungen mit der Strukturierten Analyse in einem CASE-Tool aufbereitet, für die zweite Gruppe erfolgt dies in natürlich-sprachlicher Form.
  - Die zwei Gruppen werden gebeten drei Aufgaben zu bewältigen: die Anforderungsspezifikation verstehen, Anforderungsfehler identifizieren und die Auswirkungen von Änderungen bestimmen.
- Experiment 2:
  - Die zwei Gruppen aus dem ersten Experiment erhalten eine natürlich-sprachliche Problembeschreibung.
  - Die Gruppen werden jeweils gebeten, eine Anforderungsspezifikation zu erstellen, diese Spezifikation zu prüfen und etwaige Fehler zu korrigieren, Entwurfsentscheidungen zu spezifizieren und diese ebenfalls zu prüfen.
  - Die erste Gruppe wird gebeten, diese Aufgaben in einem CASE-Tool mit Strukturierten Methoden zu bewältigen, während die zweite Gruppe diese Aufgabe mit einer Textverarbeitung natürlich-sprachlich lösen soll.

Subjekte bzw. Objekte der Untersuchung (S. 70 f.)

- Die erste Gruppe besteht aus drei Studenten, die in einer dreitägigen Schulung in die Strukturierten Methoden und einem CASE-Tool eingeführt werden.
- Die zweite Gruppe besteht ebenfalls aus drei Studenten. Sie besitzen praktische Erfahrungen mit der natürlich-sprachlichen Beschreibung von Anforderungen und Entwurfsentscheidungen.

Wesentliche Ergebnisse der Untersuchung (S. 71-75)

- Die Strukturierte Analyse ist bei der Definition von Daten und Schnittstellen effektiver als die natürlich-sprachliche Methode. Mit der Strukturierten Analyse erstellte Anforderungsspezifikationen erschweren das Verständnis der Anforderungen, wenn abhängige Bestandteile auf mehrere Dokumente verteilt sind.
- Strukturierte Methoden mit der Unterstützung eines CASE-Tools sind effizienter als die natürlich-sprachliche Methode.

Informationen zur Literaturquelle

Nr.: Q21 **Kurztitel:** Walz, Elam, Curtis /Softtware design team/ **Jahr der Veröffentlichung**: 1993

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 63)

- Wie eignen sich Softwareentwickler projektrelevantes Wissen an? Wie nutzen und integrieren sie gemeinsam projektrelevantes Wissen?
- Bestehen zwischen Softwareentwicklern Unterschiede in dem Ausmaß, wie sie an diesen drei Aktivitäten mitwirken?
- Untersuchungseinheit: Aneignung, Nutzung und Integration von projektrelevantem Wissen in einem Softwareentwicklungsteam.

Vorgehensweise der Untersuchung (S. 63 f., 67, 75)

- Autoren zeichnen Sitzungen eines Softwareentwicklungsteams auf Video auf. Diese Videoaufzeichnungen werden schriftlich dokumentiert.
- Die schriftlichen Dokumente der Aufzeichnungen werden in zwei Stufen ausgewertet:
  - Zunächst wird qualitativ bewertet, auf welche Art und in welchem Ausmaß Wissen angeeignet, genutzt und integriert wird.
  - Im zweiten Schritt werden die Dokumente strukturiert ausgewertet, um damit die qualitativen Bewertungen des ersten Schritts zu stützen oder zu entkräften.

Subjekte bzw. Objekte der Untersuchung (S. 63 f.)

- 19 Sitzungen im Rahmen eines Softwareentwicklungsprojekts werden im Zeitraum von August bis November 1986 in dem Unternehmen Microelectronics and Computer Technology Corporation (MCC) aufgezeichnet.
- An diesen Sitzungen haben insgesamt 8 Entwickler, ein Projektmanager und ein Kundenvertreter teilgenommen.

Wesentliche Ergebnisse der Untersuchung (S. 67-75)

- Bedürfnisse der Kunden wurden nicht während der Projektlaufzeit zufrieden gestellt.
- Das Entwicklungsteam wurde mit vielen Informationen konfrontiert, welche nie festgehalten worden sind.
- Kontextabhängiges Lernen war für die Softwareentwicklung wichtig.
- Die Anforderungsanalyse wurde nicht abgeschlossen, nachdem ein vollständiges Verständnis der erforderlichen Softwareanforderungen vorlag, sondern nachdem die Hälfte der geplanten Projektdauer abgelaufen war.

9.2 Anhang A.2: Empirische Literaturquellen zu ODC

Nachfolgend werden 27 empirische Quellen zu ODC anhand einheitlicher Beschreibungskriterien dargestellt. Diese Quellen werden im Rahmen der systematischen Literaturanalyse zu ODC aus Kapitel 5.2 berücksichtigt. Die eingesetzten Beschreibungskriterien werden in Kapitel 5.2.2.2 vorgestellt. In der nachfolgenden Tabelle 9-1 ist aufgeführt, welche empirische ODC-Quelle welche ODC-Attribute thematisiert (Hinweis: k. A. steht für „keine Angabe“).

Quelle	**Aktivität (activity)**	**Auslöser (trigger)**	**Auswirkung (impact)**	**Ziel (target)**	**Fehlertyp (defect type)**	**Kennzeichner (qualifier)**	**Ursprung (source)**	**Alter (age)**
ODC01	ja	ja	ja	ja	ja	ja	nein	nein
ODC02	ja	ja	nein	nein	nein	nein	nein	nein
ODC03	ja	ja	nein	nein	ja	nein	nein	nein
ODC04	nein	ja	nein	nein	nein	nein	ja	ja
ODC05	ja	ja	nein	nein	nein	nein	nein	nein
ODC06	ja	nein	nein	nein	ja	nein	nein	nein
ODC07	nein	ja	ja	nein	ja	ja	nein	ja
ODC08	nein	ja	ja	nein	ja	ja	nein	ja
ODC09	k. A.	k. A.	k. A.	k. A.	k. A.	k. A.	k. A.	k. A.
ODC10	ja	ja	nein	nein	ja	ja	nein	ja
ODC11	ja	ja	nein	nein	ja	ja	nein	nein
ODC12	nein	ja	nein	nein	ja	nein	nein	nein
ODC13	nein	ja	nein	nein	ja	ja	nein	nein
ODC14	nein	ja	nein	nein	nein	nein	nein	nein
ODC15	nein	nein	nein	nein	ja	nein	nein	nein
ODC16	nein	nein	nein	nein	ja	nein	nein	nein
ODC17	nein	ja	nein	nein	nein	nein	nein	nein
ODC18	nein	ja	nein	nein	ja	nein	nein	nein
ODC19	ja	ja	ja	nein	ja	ja	nein	ja
ODC20	ja	ja	nein	nein	nein	nein	nein	nein
ODC21	nein	nein	nein	nein	ja	nein	nein	nein
ODC22	nein	nein	nein	nein	ja	nein	nein	nein
ODC23	ja	ja	nein	ja	ja	nein	nein	nein
ODC24	ja	ja	nein	nein	ja	nein	nein	nein
ODC25	nein	ja	nein	nein	ja	nein	nein	nein
ODC26	k. A.	k. A.	k. A.	k. A.	k. A.	k. A.	k. A.	k. A.
ODC27	nein	nein	nein	nein	ja	nein	nein	nein

Tabelle 9-1: Überblick über die in den ODC-Untersuchungen eingesetzten ODC-Attribute

Informationen zur Literaturquelle

Nr.: ODC01 **Kurztitel:** Bassin, Biyani, Santhanam /Metrics/ **Jahr der Veröffentlichung**: 2002

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 12 f.)

- Darstellung von ausgewählten Kennzahlen zu Testfällen, zur Ausführung von Testfällen und zu Fehlern im Kontext einer ausgelagerten, inkrementellen Softwareentwicklung. Diese Kennzahlen sollen dem Abnehmer der Software die Möglichkeit geben, den Grad der Fertigstellung der Software, die Qualität der Software, den Fortschritt der eigenen Testaktivitäten und ihre Effektivität zu bestimmen.
- Untersuchungseinheit: Kennzahlen zu Testfällen, zur Ausführung von Testfällen und zu Fehlern.

Vorgehensweise der Untersuchung (S. 16-19, 26 f.)

- Bestehende Testfälle werden nachträglich gemäß dem ODC-Attribut Auslöser (trigger) klassifiziert. Alle Fehler, die durch einen Testfall entdeckt werden, erhalten die gleiche Ausprägung für dieses Attribut wie der Testfall.
- Neben Daten zur Ausführung der Testfälle werden insbesondere auf der Grundlage von ODC auch Daten zu gefundenen Fehlern erfasst.
- Manche Testfälle werden in mehreren Teststufen eingesetzt.
- Die verfügbaren Daten werden eingesetzt, um Werte für bestimmte Kennzahlen (s.u.) zu ermitteln.
- Die Korrektheit und Vollständigkeit der erhobenen Daten wird überprüft.

Subjekte bzw. Objekte der Untersuchung (S. 14-16)

- Gegenstand der Untersuchung ist eine Software für die Olympischen Spiele 2000 in Sydney. Die Software erfasst und sammelt die Ergebnisse von jedem sportlichen Ereignis während der Olympiade und berichtet die Ergebnisse 15000 akkreditierten Medienvertretern, Sportoffiziellen, Trainern, Sportlern, Teilnehmern und 3,7 Milliarden Zuschauern, die die Olympiade von der Ferne verfolgen.
- Ein externes Unternehmen ist für den Entwurf, die Implementierung und bestimmte Qualitätssicherungsmaßnahmen (Code-Inspektionen, Modultest, einfache funktionale Tests) verantwortlich. Die Softwareentwicklung erfolgt inkrementell und der Abnehmer bekommt während der Entwicklung Zwischenstände von Komponenten der Software zum Testen.
- Eine Einheit von IBM übernimmt für den Abnehmer der Software mehrere funktionale Tests, einen Performanztest und den Abnahmetest.

Wesentliche Ergebnisse der Untersuchung (S. 21-28)

- Folgende Kennzahlen werden neben ODC-Kennzahlen eingesetzt:
  - prozentualer Anteil ausgeführter Testfälle (Indikator für den Fortschritt der Testbemühungen),
  - Anzahl Fehler pro ausgeführtem Testfall (Indikator für die Qualität der Software),
  - Anzahl fehlgeschlagener Testfälle ohne Fehlerbericht (Indikator für den Grad der Vollständigkeit der Fehlerverfolgung),
  - prozentualer Anteil der Testfälle, die bei ihrer letzten Ausführung erfolgreich abgeschlossen werden konnten (Indikator für die Qualität und Stabilität der Software),
  - prozentualer Anteil der Testfälle, die beständig fehlschlagen (Indikator für die Qualität der Software),
  - prozentualer Anteil der Testfälle, deren Ausführung fehlschlagen, nachdem sie zuvor erfolgreich abgeschlossen werden konnten (Indikator dafür, in welchem Umfang fehlerhafte Codeänderungen durchgeführt worden sind).
- Die Kennzahlen können erfolgreich eingesetzt werden, um den Grad der Fertigstellung der Software, die Qualität der Software, den Fortschritt der eigenen Testbemühungen und ihre Effektivität zu bestimmen.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC01 **Kurztitel:** Bassin, Biyani, Santhanam /Metrics/		**Jahr der Veröffentlichung**: 2002	
Darstellung der Literaturquelle			
Softwareart	Anwendungssoftware		
Eingesetzte ODC-Attribute für Fehlerfindung	Aktivität (activity), Auslöser (trigger), Auswirkung (impact)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Ziel (target), Fehlertyp (defect type), Kennzeichner (qualifier)		
Zusätzliche Attribute für Fehler	Verknüpfung zum Testfall, Verknüpfung zur Historie der Testfallausführung, Fehlerbeschreibung, Fehlerschwere (severity), Priorität (priority), Bearbeitungsstatus (status). (S.19)		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	Statt Fehler gemäß dem Attribut *Auslöser (trigger)* zu klassifizieren, werden alle Testfälle gemäß diesem Attribut klassifiziert. Alle Fehler, die durch einen Testfall entdeckt werden, erhalten die gleiche Ausprägung für dieses Attribut wie der Testfall. (S. 18)		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	nein		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	ja	Produktionsfehler	nein
Vorgehensweise bei Fehler-klassifikation	direkt		

Informationen zur Literaturquelle
Nr.: ODC02 **Kurztitel:** Bassin, Biyani, Santhanam: /Sydney Olympics/ **Jahr der Veröffentlichung**: 2001
Darstellung der Literaturquelle

Ziele der Untersuchung (S. 264-268)

- Darstellung einer angepassten Anwendung von ODC auf Anwendungssoftware für die Olympischen Spiele 2000 in Sydney. Anhand von ODC-Kennzahlen sowie weiteren ausgewählten Kennzahlen werden die Einhaltung der Teststrategie in jeder Testaktivität, der Umfang und die Effizienz jeder Testaktivität und der Fortschritt der Testaktivitäten bestimmt.
- Untersuchungseinheit: angepasste Anwendung von ODC und der Nutzen dieser Anwendung.
- Vgl. auch Bassin, Biyani, Santhanam /Metrics/. Beide Beiträge beschreiben das gleiche Projekt, setzen jedoch unterschiedliche inhaltliche Schwerpunkte.

Vorgehensweise der Untersuchung (S. 265-268)

- Für eine Stichprobe von 500 aus insgesamt 38000 Testfällen wird geprüft,
  - ob Testfälle gemäß dem ODC-Attribut Auslöser (trigger) klassifiziert werden können und
  - ob es inhaltlich korrekt ist, wenn durch einen Testfall entdeckte Fehler die Ausprägung dieses Attributs vom Testfall übernehmen.
- Nach positiven Ergebnissen werden alle bestehenden Testfälle nachträglich gemäß dem ODC-Attribut Auslöser (trigger) klassifiziert.

Subjekte bzw. Objekte der Untersuchung (S 264 f.)

- Gegenstand der Untersuchung ist eine Anwendungssoftware für die Olympischen Spiele 2000 in Sydney.
- Die Entwicklung einzelner Komponenten der Software wird ausgelagert.
- Einheiten von IBM führen mehrere funktionale Tests durch.

Wesentliche Ergebnisse der Untersuchung (S. 267-273)

- Testfälle können gemäß dem ODC-Attribut Auslöser klassifiziert werden.
- Diese Veränderung von ODC bringt folgende Vorteile:
  - Tester müssen für die Erfassung von Fehlern in ODC nicht geschult werden, da die Ausprägung für das Attribut Aktivität über das Datum der Fehlerentdeckung automatisch bestimmt und die Ausprägung für das Attribut Auslöser vom entsprechenden Testfall übernommen wird.
  - Tester müssen aus den gleichen Gründen keine ODC-Informationen erfassen.
  - Die Klassifikation von Fehlern muss nur bei 9% der Fehler überarbeitet werden.
  - In Verbindung mit weiteren ausgewählten Kennzahlen können die ODC-basierten Auswertungen die Einhaltung der Teststrategie in jeder Testaktivität, den Umfang und die Effizienz jeder Testaktivität und den Fortschritt der Testaktivitäten bestimmen.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC02 **Kurztitel:** Bassin, Biyani, Santhanam: /Sydney Olympics/ **Jahr der Veröffentlichung**: 2001			
Darstellung der Literaturquelle			
Softwareart	Anwendungssoftware		
Eingesetzte ODC-Attribute für Fehlerfindung	Aktivität (activity), Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	nein		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	Statt Fehler gemäß dem Attribut *Auslöser (trigger)* zu klassifizieren, werden alle Testfälle gemäß diesem Attribut klassifiziert. Alle Fehler, die durch einen Testfall entdeckt werden, erhalten die gleiche Ausprägung für dieses Attribut wie der Testfall. (S. 266)		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	Auslöser (trigger) für Testfälle x Aktivität (activity) (S. 269) Auslöser (trigger) für Testfälle x Sportart (S. 270 f.)		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	ja	Produktionsfehler	nein
Vorgehensweise bei Fehler-klassifikation	direkt		

Informationen zur Literaturquelle

Nr.: ODC03 **Kurztitel:** Bassin, Kratschmer, Santhanam /Software development/
Jahr der Veröffentlichung: 1998

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 66)

- Einführende Darstellung der ODC und ihrer Nutzenpotenziale anhand dreier Beispiele aus der Praxis.
- Untersuchungseinheit: ODC.

Vorgehensweise der Untersuchung (S. 67-73)

- Darstellung der ODC und einer Erweiterung namens „butterfly model“ zur Auswertung von ODC-klassifizierten Fehlern.
- Darstellung drei verschiedener ODC-Anwendungen als Beispiel.

Subjekte bzw. Objekte der Untersuchung (S 67, 70, 72 f.)

- Untersuchung 1: Risikobewertung einer betrieblichen Anwendungssoftware mit ODC. Die Anwendungssoftware wurde extern entwickelt und in C++ programmiert.
- Untersuchung 2: Senkung der Garantiekosten eines Softwareproduktes mit ODC.
- Untersuchung 3: Verbesserung der Qualitätssicherungsprozesse einer Altsoftware mit ODC.

Wesentliche Ergebnisse der Untersuchung (S. 67-74)

- In der ersten Untersuchung können mit ODC folgende Fragen beantwortet werden:
  - Ist die Software ausreichend reif für einen produktiven Einsatz?
  - Welche Maßnahmen sind erforderlich, um die Software in einen reifen Zustand zu bringen?
  - Wie sollten Prozesse zukünftig gestaltet werden, um Mängel zu vermeiden?
- In der zweiten Untersuchung wird für ein Softwareprodukt das Nutzungsprofil der Kunden ermittelt, indem durch Kunden entdeckten Fehler analysiert werden. Um die Garantiekosten zu reduzieren, werden Maßnahmen definiert, um Fehler zu vermeiden und früher zu finden, die insbesondere das Nutzungsprofil der Kunden berücksichtigen.
- In der dritten Untersuchung wird ODC erfolgreich eingesetzt, um Verbesserungspotenziale in der Qualitätssicherung einer Altsoftware aufzudecken.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC03 **Kurztitel:** Bassin, Kratschmer, Santhanam /Software development/ **Jahr der Veröffentlichung**: 1998			
Darstellung der Literaturquelle			
Softwareart	Untersuchung 1: Anwendungssoftware Untersuchung 2 und 3: keine Angaben		
Eingesetzte ODC-Attribute für Fehlerfindung	Aktivität (activity), Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type)		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	„butterfly model“ zur Fehlerauswertung (S. 72 f.)		
Quantitative ODC-Daten	Auslöser (trigger) S. 68 Aktivität (activity) S. 70 Aktivität (activity) x Auslöser (trigger) S. 72 Zeitraum x Auslöser (trigger) S. 73		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	ja	Produktionsfehler	ja
Vorgehensweise bei Fehlerklassifikation	keine Angaben		

Informationen zur Literaturquelle	
Nr.: ODC04 **Kurztitel:** Bassin, Santhanam /Maintenance/	**Jahr der Veröffentlichung**: 2001

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 726 f.)

- Wie können Fehler bei der Wartung portierter, ausgelagerter und alter Software mit ODC vermieden oder früher gefunden werden?
- Untersuchungseinheit: ODC in der Wartung von Software.

Vorgehensweise der Untersuchung (S. 727)

- Darstellung der ODC-Attribute Ursprung (source) und Alter (age).
- Darstellung der Erfahrungen aus drei Wartungsprojekten, in denen anhand von ODC Maßnahmen bestimmt werden, um Fehler zu vermeiden und früher zu finden.

Subjekte bzw. Objekte der Untersuchung (S. 729-731)

- Softwareprodukt A ist eine Anwendungssoftware für Großrechner. Der Code einer Komponente dieser Software wird von einer bestehenden Software für eine andere Plattform portiert.
- Zum Softwareprodukt B werden abgesehen vom Hinweis, dass ein Teil der Entwicklung ausgelagert wird, keine Angaben gemacht.
- Zum Softwareprodukt C werden keine Angaben gemacht.

Wesentliche Ergebnisse der Untersuchung (S. 729-733)

- ODC kann erfolgreich in der Wartung von drei Softwareprodukten eingesetzt werden, um Fehler zu vermeiden und früher zu finden.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC04 **Kurztitel:** Bassin, Santhanam /Maintenance/		**Jahr der Veröffentlichung**: 2001	
Darstellung der Literaturquelle			
Softwareart	Produkt A: Anwendungssoftware für Großrechner, Produkt B: keine Angaben, Produkt C: keine Angaben.		
Eingesetzte ODC-Attribute für Fehlerfindung	Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Alter (age), Ursprung (source)		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	„butterfly model“ zur Fehlerauswertung (S. 728)		
Quantitative ODC-Daten	Ursprung (source) x Auslöser (trigger) (S. 730 f.) Zeitraum x Alter (age) (S. 731)		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	ja	Produktionsfehler	ja
Vorgehensweise bei Fehler-klassifikation	keine Angaben		

Informationen zur Literaturquelle	
Nr.: ODC05 **Kurztitel:** Bassin, Santhanam /Software triggers/	**Jahr der Veröffentlichung**: 1997

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 103-105)

- Wie können Prozesse der Entwicklung und Qualitätssicherung verbessert und Kundennutzungsprofile einer Software erstellt werden, indem das ODC-Attribut Auslöser (trigger) ausgewertet wird?
- Untersuchungseinheit: Anwendung des ODC-Attributs Auslöser (trigger).

Vorgehensweise der Untersuchung (S. 103)

- Darstellung des ODC-Attributs Auslöser (trigger).
- Darstellung der Erfahrungen bzgl. der Auswertung dieses Attributs bei zwei verschiedenen Softwareprodukten.

Subjekte bzw. Objekte der Untersuchung (S.105 f., 111)

- Softwareprodukt A ist eine Software für Großrechner. Es handelt sich vermutlich um die gleiche Software, wie in Bassin, Santhanam /Maintenance/ (ODC04), die dort ebenfalls als Softwareprodukt A bezeichnet wird.
- Softwareprodukt B ist eine Software für leistungsfähige Arbeitsplatzrechner (Workstation).

Wesentliche Ergebnisse der Untersuchung (S. 106-112)

- Mit der Verteilung der Ausprägungen des Attributs Auslöser (trigger) für Entwicklungsfehler einerseits und Produktionsfehler andererseits können Verbesserungspotenziale in der Entwicklung und Qualitätssicherung von Softwareprodukt A aufgezeigt werden.
- Eine Verbesserung der Prozesse wird erreicht, wenn sich die Verteilungen der Ausprägungen des Attributs Auslöser (trigger) für Entwicklungsfehler und Produktionsfehler ähneln.
- Solange sich die Funktionalität einer Software nicht grundlegend ändert, ändert sich die Verteilung der Ausprägungen des Attributs Auslöser (trigger) für Produktionsfehler über mehrere Versionen nur geringfügig.
- Werden die Verteilungen der Ausprägungen des Attributs Auslöser (trigger) für die Produktionsfehler zweier unterschiedlicher Softwareprodukte verglichen, können Unterschiede in den Nutzungsprofilen der jeweiligen Kundengruppen aufgezeigt werden.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC05 **Kurztitel:** Bassin, Santhanam /Software triggers/		**Jahr der Veröffentlichung**: 1997	
Darstellung der Literaturquelle			
Softwareart	Produkte A und B: keine Angaben		
Eingesetzte ODC-Attribute für Fehlerfindung	Aktivität (activity), Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	keine		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	Aktivität (activity) (S. 106, 108) Aktivität (activity) x Auslöser (trigger) (S. 106, 108) Zeitraum x Auslöser (trigger) (S. 109-111)		
Anzahl Fehler	30000		
Entwicklungsfehler	ja	Produktionsfehler	ja
Vorgehensweise bei Fehlerklassifikation	indirekt		

Informationen zur Literaturquelle

Nr.: ODC06 **Kurztitel:** Berling, Thelin /Case study/ **Jahr der Veröffentlichung**: 2003

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 227)

- Wie können Prüf- und Testprozesse hinsichtlich der gefundenen Fehler und eingesetzten Ressourcen optimal aufeinander abgestimmt werden? Diese Hauptfrage zerlegen die Autoren in drei Teilfragen:
  - Wie können die Prüf- und Testprozesse einer großen Softwareentwicklung methodisch bewertet werden?
  - Welche Kosten und welcher Nutzen entstehen, wenn Fehler in verschiedenen Phasen entdeckt werden?
  - Wie effizient sind Prüf- und Testprozesse?
- Untersuchungseinheit: Bewertung von Prüf- und Testprozessen.

Vorgehensweise der Untersuchung (S. 228-236)

- Die Autoren führen ein Gütekriterium für gefundene Fehler ein, das den Zeitpunkt der Fehlerfindung mit dem frühesten Zeitpunkt vergleicht, zu dem ein Fehler hätte gefunden werden können.
- Sie bestimmen anschließend die Schwere jedes Fehlers einer Softwareentwicklung, welche über den Korrekturaufwand eines Fehlers und den Umstand definiert wird, ob es die Ausführung von Tests behindert.
- Implementierungsfehler werden nach ODC klassifiziert.
- Schließlich wird der Aufwand für die Prüf- und Testprozesse bestimmt.

Subjekte bzw. Objekte der Untersuchung (S. 227-229)

- Untersucht wird ein nicht näher beschriebenes Softwareentwicklungsprojekt bei Ericsson Microwave Systems AB. Dieses Unternehmen stellt Radarsysteme her.
- Die Softwareentwicklung erfolgt inkrementell und es werden während der Entwicklung Inkremente zum Testen zur Verfügung gestellt.

Wesentliche Ergebnisse der Untersuchung (S. 231-237)

- Mittels des eingeführten Gütekriteriums für gefundene Fehler können die Prüf- und Testprozesse bewertet werden und die Effektivität von Verbesserungsmaßnahmen während des Softwareentwicklungsprojekts bestimmt werden.
- Die Anwendung von ODC mit dem Gütekriterium zeigt Verbesserungspotenziale im Modultest und Feinentwurf auf.
- Zwischen den Fehlern, die bei der Inspektion von Anforderungen gefunden werden, und denen, die beim Testen der Software gefunden werden, gibt es wenige Überlappungen. Die Autoren folgern daraus, dass sowohl das Prüfen als auch das Testen während einer Softwareentwicklung erforderlich sind, um die Qualität einer Software zu verbessern.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC06 **Kurztitel:** Berling, Thelin /Case study/		**Jahr der Veröffentlichung**: 2003	
Darstellung der Literaturquelle			
Softwareart	keine Angaben		
Eingesetzte ODC-Attribute für Fehlerfindung	Aktivität (activity; in der Quelle als „phase“ bezeichnet)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type)		
Zusätzliche Attribute für Fehler	Fehlerschwere (severity), die definiert wird über den Korrekturaufwand eines Fehlers und den Umstand, ob es die Ausführung von Tests behindert. (S. 229)		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	Gütekriterium für gefundene Fehler, das den Zeitpunkt der Fehlerfindung mit dem frühesten Zeitpunkt vergleicht, zu dem ein Fehler hätte gefunden werden können.		
Quantitative ODC-Daten	Aktivität (activity; in der Quelle als „phase“ bezeichnet) (S. 231 f.) Fehlertyp (defect type) (S. 234 f.)		
Anzahl Fehler	49 Fehler		
Entwicklungsfehler	ja	Produktionsfehler	nein
Vorgehensweise bei Fehlerklassifikation	keine Angaben		

Informationen zur Literaturquelle	
Nr.: ODC07 **Kurztitel:** Bhandari u. a. /Case study/	**Jahr der Veröffentlichung**: 1993

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 1157)

- Wie kann der Prozess der Softwareentwicklung bereits während der Softwareentwicklung verbessert werden?
- Untersuchungseinheit: ODC und Attribute Focusing in Anwendung

Vorgehensweise der Untersuchung (S. 1159 f.)

- Darstellung der Methoden ODC und Attribute Focusing.
- Die Autoren beschreiben Erfahrungen aus einem Softwareentwicklungsprojekt, bei dem ODC und Attribute Focusing angewandt worden sind.
- Fehler wurden während der Softwareentwicklung gemäß ODC klassifiziert.
- Die klassifizierten Fehler wurden automatisch mit der Methode Attribute Focusing ausgewertet, um nach Mustern in der Fehlermenge zu suchen. Attribute Focusing zielt darauf ab, schwer erkennbare Zusammenhänge in großen Datenbeständen aufzuspüren und ist daher eine Data-Mining-Methode[811].

Subjekte bzw. Objekte der Untersuchung (S. 1157-1160)

- Untersucht wird ein Projekt zur Entwicklung eines Betriebssystems. Das Entwicklungsteam besteht aus 25 Personen.
- Im Rahmen des Projekts werden CASE-Werkzeuge und weitere Entwicklungswerkzeuge erstmalig eingesetzt. Damit das Entwicklungsteam Erfahrungen mit den Werkzeugen sammeln kann, wird das Projekt in zwei Phasen aufgeteilt, die nahezu gleiche Entwicklungs- und Qualitätsicherungsaufgaben aufweisen.
- Es werden alle gefundenen Entwurfs- und Implementierungsfehler nach ODC erfasst.

Wesentliche Ergebnisse der Untersuchung (S. 1167 f.)

- ODC und automatisierte Fehlerauswertung nach Attribute Focusing können erfolgreich eingesetzt werden, um den Prozess der Softwareentwicklung während der Softwareentwicklung zu verbessern und die Effektivität der Verbesserungsmaßnahmen zu bestimmen.

[811] Vgl. Mertens, Wieczorrek /Strategien/ 18 f.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC07 **Kurztitel:** Bhandari u. a. /Case study/		**Jahr der Veröffentlichung**: 1993	
Darstellung der Literaturquelle			
Softwareart	Systemsoftware (Betriebssystem)		
Eingesetzte ODC-Attribute für Fehlerfindung	Auslöser (trigger), Auswirkung (impact)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type), Kennzeichner (qualifier[812]), Alter (age[813])		
Zusätzliche Attribute für Fehler	Phase gefunden (phase found) (S. 1159), Phase eingeführt (phase introduced) (S. 1159), Komponente (component) (S. 1159).		
Fehlerauswertung mit Attribute Focusing	ja		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	Kennzeichner (qualifier) (S. 1161) Fehlertyp (defect type) x Phase gefunden (phase found) (S. 1162, 1169) Alter (age) x Phase gefunden (phase found) (S. 1163) Fehlertyp (defect type) x Kennzeichner (qualifier) (S. 1163) Fehlertyp (defect type) x Komponente (component) (S. 1163) Kennzeichner (qualifier) x Phase gefunden (phase found) (S. 1165) Phase eingeführt (phase introduced) x Phase gefunden (phase found) (S. 1166) Auswirkung (impact) (S. 1166) Komponente (component) x Auslöser (trigger) (S. 1167) Auswirkung (impact) x Phase gefunden (phase found) (S. 1169)		
Anzahl Fehler	ungefähr 400 Fehler		
Entwicklungsfehler	ja	Produktionsfehler	nein
Vorgehensweise bei Fehlerklassifikation	direkt		

[812] In Bhandari u. a. /Case study/ wird das Attribut „qualifier“ nicht namentlich eingeführt, sondern über die Ausprägungen mit „missing/incorrect“ beschrieben.

[813] In Bhandari u. a. /Case study/ wird gemäß ODC 5.11 das Attribute „age“ verwendet, in dem Beitrag jedoch als „source“ bezeichnet.

Informationen zur Literaturquelle

Nr.: ODC08 **Kurztitel:** Bhandari u. a. /Improvement/ **Jahr der Veröffentlichung**: 1994

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 182 f.)

- Kann die Orthogonal Defect Classification (ODC) gemeinsam mit Attribute Focusing bei verschiedenen Arten von Softwareentwicklungsprojekten erfolgreich angewandt werden?
- Welchen Nutzen liefert ODC?
- Untersuchungseinheit: ODC in Anwendung.

Vorgehensweise der Untersuchung (S. 183, 185 f.)

- Rückblickende Darstellung der Erfahrungen und quantitativen Daten aus 7 abgeschlossenen Softwareentwicklungsprojekten, in denen ODC angewandt worden ist.
  - Fehler wurden während der Projekte durch Softwareentwickler gemäß ODC erfasst.
  - Die klassifizierten Fehler wurden automatisch mit der Methode „Attribute Focusing“ ausgewertet, um nach Mustern in der Fehlermenge zu suchen. „Attribute Focusing“ zielt darauf ab, schwer erkennbare Zusammenhänge in großen Datenbeständen aufzuspüren und ist daher eine Data-Mining-Methode[814].
- Um den Nutzen von ODC zu beschreiben werden zwei Ansätze verfolgt:
  - Der Nutzen von ODC wird für konkrete Beispiele aus den untersuchten Projekten quantifiziert.
  - Es wird untersucht, ob die Ergebnisse der ODC-Anwendung auch mit zwei anderen Verfahren hätten erlangt werden können. Zum einen ist das die Ursachenanalyse für eine Auswahl von Fehlerberichten, in denen Fehler überwiegend qualitativ beschrieben werden. Zum anderen wird ODC mit dem Einsatz zielorientierter Softwaremetriken verglichen, anhand derer überprüft wird, ob bestimmte Prozessziele erreicht werden.

Subjekte bzw. Objekte der Untersuchung (S. 183 f. und 211)

- 7 Softwareentwicklungsprojekte an sechs geographisch voneinander entfernten Standorten.[815]
- Die entwickelten Softwareprodukte zielen auf verschiedene Branchen.
- Die Softwareentwicklungsprojekte haben unterschiedliche Ausprägungen zu folgenden Merkmalen:
  - angestrebte Hardwareumgebung (Großrechner, Midrange-Rechner, Arbeitsplatzrechner)
  - angestrebte Softwareumgebung (Betriebssystem, Compiler, Datenbank, Anwendung)
  - Umfang des neuen und geänderten Codes (klein, mittel, groß, sehr groß),
  - Anzahl Programmierer und Tester,
  - eingesetzte Softwarewerkzeuge (Verfolgung von Fehlern aus den Softwaretests, Verfolgung von Daten aus Inspektionen, Verfolgung von Testfällen),
  - eingesetztes Prozessmodell (sequenziell, iterativ oder Kombination aus beiden) und
  - parallele Durchführung von unterschiedlichen Teilaufgaben für verschiedene Komponenten der Software (ja, nein).

Wesentliche Ergebnisse der Untersuchung (S. 186-212)

- Durch die Anwendung von ODC konnten latente Fehler entfernt, die Einführung bestimmter Fehler vermieden und kurzfristige sowie langfristige Prozessverbesserungspotenziale aufgedeckt werden.
- Durch den Einsatz von ODC in Kombination mit der Data-Mining-Methode „Attribute Focusing“ können in allen untersuchten 7 Projekten erfolgreich Prozessverbesserungspotenziale identifiziert werden.
- Wird ODC früh im Projekt eingesetzt, profitiert man sowohl von der Vermeidung als auch Entdeckung von Fehlern, während bei einem späten Einsatz der Nutzen sich auch die Entdeckung von Fehler beschränkt.

---

[814] Vgl. Mertens, Wieczorrek /Strategien/ 18 f.

[815] Das als „Project A“ bezeichnete Projekt in Bhandari u. a. /Improvement/ ist mit großer Wahrscheinlichkeit das in Bhandari u. a. /Case study/ behandelte Projekt.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC08 **Kurztitel:** Bhandari u. a. /Improvement/		**Jahr der Veröffentlichung**: 1994	
Darstellung der Literaturquelle			
Softwareart	Anwendungssoftware, systemnahe Software und Systemsoftware: 7 verschiedene Projekte, in denen unterschiedliche Softwarearten entwickelt werden: 2 Betriebssysteme, 2 Datenbanken, 1 Compiler und eine Anwendungssoftware.		
Eingesetzte ODC-Attribute bei Fehlerfindung	Auslöser (trigger), Auswirkung (impact)		
Eingesetzte ODC-Attribute bei Fehlerkorrektur	Fehlertyp (defect type), Kennzeichner (qualifier[816]), Alter (age[817])		
Zusätzlich eingesetzte Attribute für Fehler	Phase gefunden (phase found) (S. 185), Phase eingeführt (phase introduced) (S. 185), Komponente (component) (S. 185).		
Fehlerauswertung mit Attribute Focusing	ja		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	Ursachenanalyse mit einem Model zur Interpretation von Fehlerdaten (S. 190, 211 f.)		
Quantitative ODC-Daten	Kennzeichner (qualifier) (S. 188) Fehlertyp (defect type) (S. 189, 204) Phase eingeführt (phase found) (S. 189) Fehlertyp (defect type) x Phase eingeführt (Phase introduced) (S. 189) Auslöser (trigger) (S. 193) Alter (age) x Auslöser (trigger) (S. 194) Fehlertyp (defect type) x Auslöser (trigger) (S. 196) Alter (age) x Auswirkung (impact) (S. 197) Kennzeichner (qualifier) x Auswirkung (impact) (S. 198) Auswirkung (impact) x Auslöser (trigger) (S. 199) Auslöser (trigger) x Komponente (component) (S. 202) Fehlertyp (defect type) x Kennzeichner (qualifier) (S. 203) Fehlertyp (defect type) x Phase gefunden (phase found) (S. 206) Kennzeichner (qualifier) x Phase eingeführt (phase introduced) (S. 208) Phase eingeführt (phase introduced) (S. 207) Auslöser (trigger) (S. 210)		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	ja	Produktionsfehler	nein
Vorgehensweise bei Fehlerklassifikation	keine Angaben		

[816] In Bhandari u. a. /Improvement/ wird das Attribut „qualifier“ nicht namentlich eingeführt, sondern über die Ausprägungen „missing/incorrect“ beschrieben.

[817] In Bhandari u. a. /Improvement/ wird gemäß ODC 5.11 das Attribute „age“ verwendet, in dem Beitrag jedoch als „source“ bezeichnet.

Informationen zur Literaturquelle	
Nr.: ODC09 **Kurztitel:** Buczilowski /Defect reduction/	**Jahr der Veröffentlichung**: 1995

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 274)

- Der Autor stellt dar, wie die Anzahl der Problemberichte für eine ständig weiterentwickelte Software reduziert werden konnte.
- Untersuchungseinheit: Bemühungen zur Verbesserung der Fehleranalyse, -vermeidung und Qualitätsverfolgung.

Vorgehensweise der Untersuchung (S. 274-277)

- Ausgelieferte Fehler einer Software aus einem Zeitraum von 10 Jahren werden nachträglich nach ODC klassifiziert.
- Der Autor beschreibt, welche Verbesserungsmaßnahmen nach der Anwendung von ODC eingeleitet worden sind und die Wirkungen dieser Maßnahmen.
- Ziel der Maßnahmen ist die Reduzierung der Fehlerdichte nach Auslieferung der Software. Als Fehlerdichte wird die Anzahl ausgelieferte Fehler pro Millionen Codeanweisungen definiert.

Subjekte bzw. Objekte der Untersuchung (S. 274)

- Die untersuchte Software ist VSE, ein Betriebssystem für Großrechner von IBM.
- Ausgelieferte Fehler dieser Software, welche in besonderen Problemberichten dokumentiert sind, werden nachträglich nach ODC erfasst und ausgewertet.

Wesentliche Ergebnisse der Untersuchung (S. 1167 f.)

- Die Fehlerdichte nach Auslieferung einer neuen Version des Betriebssystems kann nach 6 Jahren um 94% reduziert werden.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC09 **Kurztitel:** Buczilowski /Defect reduction/		**Jahr der Veröffentlichung**: 1995	
Darstellung der Literaturquelle			
Softwareart	Systemsoftware (Betriebssystem)		
Eingesetzte ODC-Attribute für Fehlerfindung	keine Angaben		
Eingesetzte ODC-Attribute für Fehlerkorrektur	keine Angaben		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	nein		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	nein	Produktionsfehler	ja
Vorgehensweise bei Fehlerklassifikation	indirekt		

Informationen zur Literaturquelle

Nr.: ODC10 **Kurztitel:** Butcher, Munro, Kratschmer /Software testing/ **Jahr der Veröffentlichung**: 2002

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 31)

- Die Autoren stellen anhand von drei Untersuchungen den Nutzen von ODC dar.
- Untersuchungseinheit: ODC in Anwendung.

Vorgehensweise der Untersuchung (S. 32 f.)

- Einführende Darstellung von ODC.
- Für jede der drei Untersuchungen wird jeweils angegeben,
  - welche Rahmenbedingungen durch das untersuchte Produkt und Projekt vorgegeben waren,
  - wie ODC jeweils eingeführt wurde und
  - welche Verbesserungspotenziale durch den Einsatz von ODC aufgedeckt werden konnten.

Subjekte bzw. Objekte der Untersuchung (S. 33-41)

- Untersuchung 1:
  - Zur untersuchten Software werden keine genauen Angaben gemacht. Es gibt aber Hinweise, dass es sich um das gleiche Produkt handelt wie in Bassin, Kratschmer, Santhanam /Software development/ beschrieben (siehe Untersuchung 2 in ODC03).
  - Klassifikation von ausgelieferten und durch Kunden entdeckten Fehlern über einen Zeitraum von 12 Monaten.
- Untersuchung 2:
  - Es werden nach ODC klassifizierte Fehler einer Middleware-Software von IBM untersucht, die während der Softwareentwicklung entdeckt worden sind.
- Untersuchung 3:
  - Es wird ein Software-Werkzeug untersucht, das die Auswertung von mit ODC klassifizierten Fehlern in IBM-Projekten unterstützen soll.
  - Die Entwicklungsfehler dieses Software-Werkzeugs werden ebenfalls nach ODC klassifiziert und ausgewertet.

Wesentliche Ergebnisse der Untersuchung (S. 33-42)

- Untersuchung 1:
  - Mittels ODC wird ein Nutzungsprofil ermittelt, das beschreibt wie die Kunden die Software einsetzen.
  - Ausgehend vom Nutzungsprofil werden die Testprozesse verbessert.
- Untersuchung 2:
  - Mittels ODC wird quantitativ die Entscheidung unterstützt, ob eine Software reif für den Systemtest ist, und die Effektivität von Testfallsequenzen ermittelt.
- Untersuchung 3:
  - Die Auswertung von Entwicklungsfehlern nach ODC ergibt, dass eine für die Auslieferung geplante Version des Software-Werkzeugs nicht reif für den produktiven Einsatz ist.
  - Mit gezielten zusätzlichen Qualitätssicherungsmaßnahmen werden zahlreiche Fehler vor Auslieferung der Software entdeckt und behoben.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC10 **Kurztitel:** Butcher, Munro, Kratschmer /Software testing/ **Jahr der Veröffentlichung**: 2002			
Darstellung der Literaturquelle			
Softwareart	Untersuchung 1: keine Angaben (vermutlich Anwendungssoftware) Untersuchung 2: Systemsoftware Untersuchung 3: Anwendungssoftware		
Eingesetzte ODC-Attribute für Fehlerfindung	Aktivität (activity), Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type), Kennzeichner (qualifier), Alter (age)		
Zusätzliche Attribute für Fehler	Komponente (component) (S. 34), Fehlerschwere (severity) (S. 39)		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	Alter (age) x Aktivität (activity) (S. 34) Aktivität (activity) x Komponente (component) (S. 34) Zeitraum x Auslöser (trigger) (S. 35) Aktivität (activity) x Auslöser (trigger) (S. 37 und 40) Fehlertyp (defect type) x Kennzeichner (qualifier) (S. 38) Fehlertyp (defect type) x Auslöser (trigger) (S. 38) Alter (age) (S. 40)		
Anzahl Fehler	Untersuchung 1 und 2: keine Angaben Untersuchung 3: 138 Fehler		
Entwicklungsfehler	Untersuchung 1: nein Untersuchung 2: ja Untersuchung 3: ja	Produktionsfehler	Untersuchung 1: ja Untersuchung 2: nein Untersuchung 3: nein
Vorgehensweise bei Fehlerklassifikation	direkt		

Informationen zur Literaturquelle

Nr.: ODC11 **Kurztitel:** Chaar u. a. /In-process evaluation/ **Jahr der Veröffentlichung**: 1993

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 1058)

- Die Autoren stellen ODC als ein Instrument vor, um zu bewerten, wie effektiv Entwurfs- und Codeinspektionen sowie Testaktivitäten sind.
- Untersuchungseinheit: ODC in Anwendung.

Vorgehensweise der Untersuchung (S. 1055)

- Einführende Darstellung von ODC.
- Anhand klassifizierter Fehler aus Entwurfsinspektionen, Codeinspektionen und Testaktivitäten wird der Nutzen von ODC zur Steigerung der Fehlerfindungseffektivität aufgezeigt.

Subjekte bzw. Objekte der Untersuchung (S. 1061-1065)

- Klassifizierte Fehler eines nicht näher beschriebenen Softwareprodukts, die während Entwurfsinspektionen, Codeinspektionen und Tests entdeckt werden.

Wesentliche Ergebnisse der Untersuchung (S. 1060-1069)

- In Inspektionen können die Ausprägungen des ODC-Attributs Auslöser (trigger) mit erforderlichen Fähigkeiten der Inspektionsteilnehmer verknüpft werden. Die angemessene Besetzung eines Inspektionsteams erhöht die Fehlerfindungseffektivität.
- In Tests können die Ausprägungen des ODC-Attributs Auslöser (trigger) mit erforderlichen Fähigkeiten zur Erstellung von Testfällen verknüpft werden.
- In Tests erlaubt die Verteilung der Ausprägungen des ODC-Attributs Auslöser (trigger) für Fehler Rückschlüsse darüber, wie effektiv und vollständig eine Menge von Testfällen ist.
- Fehlerauswertungen mit den ODC-Attributen Fehlertyp (defect type) und Auslöser (trigger) geben Rückschlüsse über die Qualität einer in Entwicklung befindlichen Software und damit den Fortschritt der QS-Maßnahmen.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC11 **Kurztitel:** Chaar u. a. /In-process evaluation/		**Jahr der Veröffentlichung**: 1993	
Darstellung der Literaturquelle			
Softwareart	keine Angaben		
Eingesetzte ODC-Attribute für Fehlerfindung	Aktivität (activity), Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type), Kennzeichner (qualifier)		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	ja (S. 1058)[818]		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	Fehlertyp (defect type) x Kennzeichner (qualifier) (S. 1060, 1062, 1066) Auslöser (trigger) x Fehlertyp (defect type) (S. 1061, 1063, 1065)		
Anzahl Fehler	Inspektionen des dokumentierten Ergebnisses für den Grobentwurf (design specification document) ergeben 255 Fehler. Inspektionen des dokumentierten Ergebnisses für den Feinentwurf (design structures document) ergeben 222 Fehler. Inspektionen eines Softwarecodes ergeben 333 Fehler. erster Funktionstest: 148 Fehler. zweiter Funktionstest: 253 Fehler.		
Entwicklungsfehler	ja	Produktionsfehler	nein
Vorgehensweise bei Fehlerklassifikation	keine Angaben		

[818] Die Methode „Attribute Focusing“ wird im Text nicht namentlich explizit genannt, sondern nur beschrieben. Literaturverweise auf Quellen zu „Attribute Focusing“ zeigen jedoch, dass eindeutig „Attribute Focusing“ eingesetzt wurde.

Informationen zur Literaturquelle	
Nr.: ODC12 **Kurztitel:** Chernak /Statistical approach/	**Jahr der Veröffentlichung**: 1996

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 866)

- Der Autor macht einen Vorschlag, wie Prüflisten systematisch erstellt werden können, anhand derer Inspektionen effektiv durchgeführt werden sollen. Dieser Vorschlag wird exemplarisch anhand einer Fallstudie demonstriert.
- Untersuchungseinheit: Vorschlag zur Erstellung von Prüflisten für Inspektionen.

Vorgehensweise der Untersuchung (S. 866-872)

- Darstellung von Anforderungen an Prüflisten für Checklisten.
- Darstellung des Vorschlags zur Erstellung von Prüflisten für Inspektionen.
- Darstellung der exemplarischen Anwendung des Vorschlags in einer Fallstudie.

Subjekte bzw. Objekte der Untersuchung (S. 870)

- Vorschlag wird für die Inspektion einer Spezifikation einer grafischen Benutzeroberfläche angewandt.
- Die untersuchte Software ist eine Finanzanwendungssoftware, dessen Zielsystem das Betriebssystem MS Windows ist. Als Programmiersprache wird MS Visual C++ eingesetzt.
- Die Softwareentwicklung erfolgt sequenziell.

Wesentliche Ergebnisse der Untersuchung (S. 873)

- In Anlehnung an ODC wird ein Vorschlag gemacht, wie Prüflisten für Inspektionen systematisch erstellt werden können.
- Dieser Vorschlag wird erfolgreich für die Inspektion einer Spezifikation einer grafischen Benutzer-oberfläche angewandt.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC12 **Kurztitel:** Chernak /Statistical approach/		**Jahr der Veröffentlichung**: 1996	
Darstellung der Literaturquelle			
Softwareart	Anwendungssoftware		
Eingesetzte ODC-Attribute für Fehlerfindung	Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type)		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	Gemäß dem Autor muss eine Prüfliste für eine Inspektion zwei Fragen beantworten: ▪ Worauf muss man beim Prüfobjekt besonders achten? ▪ Wie kann man einen Fehler entdecken? Die erste Frage lehnt er an dem ODC-Attribut Fehlertyp (defect typ) und die zweite Frage an dem ODC-Attribute Auslöser (trigger) an. Im Gegensatz zu ODC sind die Ausprägungen der Attribute für ein Prüfobjekt neu herzuleiten. Diese Attribute werden zudem nicht ausschließlich auf Code und Entwurfsergebnisse angewandt..		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	nein		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	keine Angaben	Produktionsfehler	keine Angaben
Vorgehensweise bei Fehler-klassifikation	keine Angaben		

Informationen zur Literaturquelle

Nr.: ODC13 **Kurztitel:** Chillarege u. a. /In-process measurements/ **Jahr der Veröffentlichung**: 1992

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 943)

- Die Autoren stellen ODC als einen Vorschlag vor, anhand dessen während eines laufenden Softwareentwicklungsprojekts der Fortschritt der in Entwicklung befindlichen Software und die Effektivität und Vollständigkeit von Qualitätssicherungsmaßnahmen bestimmt werden kann.
- ODC wird mit exemplarischen Fehlerdaten aus Pilotprojekten bei IBM verdeutlicht.
- Untersuchungseinheit: ODC in Anwendung.

Vorgehensweise der Untersuchung (S. 943)

- Definition von ODC und Darstellung der Anwendung von ODC.
- Darstellung von ODC-Fehlerdaten aus Pilotprojekten bei IBM.

Subjekte bzw. Objekte der Untersuchung (S. 947-954)

- Zu den untersuchten Softwareprodukten werden weitgehend keine Angaben gemacht.

Wesentliche Ergebnisse der Untersuchung (S. 954)

- Messungen mit ODC können Softwareentwicklern während eines laufenden Projekts entscheidungsrelevante Informationen bereitstellen.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC13 **Kurztitel:** Chillarege u. a. /In-process measurements/ **Jahr der Veröffentlichung**: 1992			
Darstellung der Literaturquelle			
Softwareart	keine Angaben		
Eingesetzte ODC-Attribute für Fehlerfindung	Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type), Kennzeichner (qualifier[819])		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	Fehlertyp (defect type) x Zeitraum (S. 948) Fehlertyp (defect type) x Kennzeichner (qualifier) (S. 949 f.) Fehlertyp (defect type) (S. 951) Auslöser (trigger) (S. 952) Auslöser (trigger) x Fehlertyp (defect type) (S. 953 f.)		
Anzahl Fehler	Inspektion Grobentwurf: 153 Fehler[820] Inspektion Feinentwurf: 222 Fehler Inspektion Softwarecode: 333 Fehler Funktionstest: 253 Fehler Systemtest: 325 Fehler		
Entwicklungsfehler	ja	Produktionsfehler	nein
Vorgehensweise bei Fehlerklassifikation	keine Angaben		

[819] In Chillarege u. a. /In-process measurements/ wird das Attribut „qualifier“ nicht namentlich eingeführt, sondern über die Ausprägungen „missing/incorrect“ beschrieben.

[820] Die absoluten Fehlerzahlen entsprechenden denen in Chaar u. a. /In-process evaluation/. Folglich ist es sehr wahrscheinlich, dass jeweils das gleiche Projekt dargestellt wird.

Informationen zur Literaturquelle	
Nr.: ODC14 **Kurztitel:** Chillarege, Bassin /Software triggers/	**Jahr der Veröffentlichung**: 1995

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 327 f.)

- Welche Bedingungen führen bei einer ausgelieferten Software dazu, dass latente Fehler aktiv werden und damit Fehlverhalten der Software auslösen?
- Wie verteilen sich diese Bedingungen über die Zeit?
- Untersuchungseinheit: Bedingungen, die Fehlverhalten auslösen.

Vorgehensweise der Untersuchung (S. 327)

- Einführung des ODC-Konzepts der Auslöser (trigger).
- Auswertung der Verteilung des Attributs Auslöser (trigger) für Produktionsfehler.

Subjekte bzw. Objekte der Untersuchung (S. 328)

- Alle Produktionsfehler eines Betriebssystems, die im Zeitraum von 2 Jahren nach Auslieferung entdeckt worden sind.[821]
- Betriebssystem umfasst mehrere Millionen Codezeilen und hat einen großen Kundenstamm.

Wesentliche Ergebnisse der Untersuchung (S. 341)

- Jeder Wert des Attributs Auslöser (trigger) hat eine unterschiedliche Verteilung über die Zeit.

[821] Es handelt sich mit großer Wahrscheinlichkeit um das gleiche Untersuchungsobjekt wie in Bassin, Kratschmer, Santhanam /Software development/, wo es als dritte Untersuchung („case study 3“) vorgestellt wird.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC14 **Kurztitel:** Chillarege, Bassin /Software triggers/		**Jahr der Veröffentlichung**: 1995	
Darstellung der Literaturquelle			
Softwareart	Systemsoftware (Betriebssystem)		
Eingesetzte ODC-Attribute für Fehlerfindung	Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	keine		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	Auslöser (trigger) (S. 336, 341) Auslöser (trigger) x Zeitraum (S. 338-340)		
Anzahl Fehler	2770 Fehler		
Entwicklungsfehler	Nein	Produktionsfehler	ja
Vorgehensweise bei Fehler-klassifikation	indirekt		

Informationen zur Literaturquelle	
Nr.: ODC15 **Kurztitel:** Chillarege, Biyani /Risk/	**Jahr der Veröffentlichung**: 1994

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 282)

- Wie können ODC und Wachstumsmodelle gemeinsam eingesetzt werden, um in einem laufenden Projekt die Steuerung des Projekts zu unterstützen?
- Untersuchungseinheit: ODC und Wachstumsmodelle

Vorgehensweise der Untersuchung (S. 327)

- Darstellung von ODC und eines Wachstumsmodells für Implementierungsfehler.
- Anwendung von ODC gemeinsam mit dem Wachstumsmodell auf reale Fehler aus einem Softwareentwicklungsprojekt.

Subjekte bzw. Objekte der Untersuchung (S. 286)

- Es werden die Implementierungsfehler aus einer nicht näher beschriebenen Softwareentwicklung untersucht. Die Fehler sind nach ODC klassifiziert.
- Der berücksichtigte Zeitraum beginnt mit dem Funktionstest, deckt den Systemtest der Software ab und endet wenige Monate vor dem geplanten Auslieferungstermin der Software.

Wesentliche Ergebnisse der Untersuchung (S. 288 f.)

Werden Wachstumskurven für Gruppen von Werten des ODC-Attributs Fehlertyp erstellt und deren zukünftiger Verlauf prognostiziert, erlaubt dies

- die Entwicklung und den aktuellen Stand der Qualität einer Software zu bestimmen,
- eine Risikobewertung hinsichtlich der Einhaltung der Projekttermine und der Anzahl ausgelieferter Fehler, und
- zielgerichtete Maßnahmen zur Steuerung des Projekts einzuleiten, um entsprechende Risiken zu reduzieren (u. a. durch eine für spezifische Probleme geeignete Teambesetzung).

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC15 **Kurztitel:** Chillarege, Biyani /Risk/		**Jahr der Veröffentlichung**: 1994	
Darstellung der Literaturquelle			
Softwareart	keine Angaben		
Eingesetzte ODC-Attribute für Fehlerfindung	keine		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type)		
Zusätzliche Attribute für Fehler	Fehlerschwere (severity) (S. 287)		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	Wachstumsmodelle zur Progrose der Fehlerentwicklung		
Quantitative ODC-Daten	Fehlertyp (defect type) x Zeitraum (S. 286 f.)		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	ja	Produktionsfehler	nein
Vorgehensweise bei Fehler-klassifikation	keine Angaben		

Informationen zur Literaturquelle

Nr.: ODC16 **Kurztitel:** Chillarege, Kao, Condit /Defect type/ **Jahr der Veröffentlichung**: 1991

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 246, 254)

- Existieren messbare Ursache-Wirkungszusammenhänge zwischen Typen von Fehlern und dem Zuverlässigkeitswachstum (reliability growth) der Software?
- Untersuchungseinheit: Ursache-Wirkungszusammenhänge zwischen Fehlertypen und dem Zuverlässigkeitswachstum.

Vorgehensweise der Untersuchung (S. 247-254)

- Jeder Fehler wird dahingehend klassifiziert, welche Kategorie von Fehlverhalten er verursacht. Die Fehler werden nach der Kategorie des Fehlverhaltens gruppiert und für jede entstehende Teilmenge von Fehlern wird die Zuverlässigkeitswachstumskurve ermittelt. Eine Wachstumskurve beschreibt jeweils den Verlauf der kumulativen Anzahl der Fehler über die Zeit (gemessen in Kalendertagen).
- Fehlerteilmengen, deren Zuverlässigkeitswachstumskurven sich ähneln, werden zu Teilpopulationen vereint. Als Ergebnis entstehen insgesamt vier verschiedene Teilpopulationen von Fehlern, die unterschiedliche Zuverlässigkeitswachstumskurven aufweisen.
- Für jede Teilpopulation wird eine repräsentative Anzahl von Fehlern nachträglich nach dem ODC-Attribut Fehlertyp klassifiziert. Schließlich kann für jede Teilpopulation von Fehlern die Verteilung des ODC-Attributs Fehlertyp bestimmt werden.

Subjekte bzw. Objekte der Untersuchung (S. 247)

- Es werden die Implementierungsfehler eines nicht näher beschriebenen Betriebssystems untersucht. Die Software ist überwiegend mit der Programmiersprache C und in geringen Anteilen mit Assembler entwickelt.
- Die berücksichtigten Fehler wurden im Funktionstest, dem Systemtest und kurze Zeit nach der Auslieferung der Software entdeckt.
- Am Projekt sind mehrere Hundert Entwickler und Tester beteiligt (genauere Angaben fehlen).

Wesentliche Ergebnisse der Untersuchung (S. 252-254)

- Teilpopulationen von Fehlern, deren Zuverlässigkeitswachstumskurve eine starke Beugung aufweisen, sind in hohem Maße mit dem Fehlertyp „Initialisierung“ verknüpft. Die Autoren begründen dies damit, dass Initialisierungsfehler früh in einem Codeausführungspfad auftreten und deshalb weitere, abhängige Fehler maskiert werden.
- Die Verteilung des ODC-Attributs Fehlertyp zeigt Verbesserungspotenziale in der Anforderungsanalyse und dem Entwurf auf.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC16 **Kurztitel:** Chillarege, Kao, Condit /Defect type/		**Jahr der Veröffentlichung**: 1991	
Darstellung der Literaturquelle			
Softwareart	Systemsoftware (Betriebssystem)		
Eingesetzte ODC-Attribute für Fehlerfindung	keine		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type)		
Zusätzliche Attribute für Fehler	Fehlverhaltenskategorie (symptom) (S. 247)		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	Fehlertyp (defect type) (S. 252 f.)		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	ja	Produktionsfehler	ja
Vorgehensweise bei Fehler-klassifikation	indirekt		

Informationen zur Literaturquelle

Nr.: ODC17 **Kurztitel:** Chillarege, Prasad /ODC Triggers/ **Jahr der Veröffentlichung**: 2002

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 669)

- Wie kann das ODC-Attribut Auslöser (trigger) eingesetzt werden, um abgeschlossene Softwareentwicklungsprojekte quantitativ zu bewerten und Verbesserungspotenziale für zukünftige Projekte aufzuzeigen?
- Untersuchungseinheit: ODC in Anwendung

Vorgehensweise der Untersuchung (S. 670, 674-676)

- Auf der Grundlage von Fehlerberichten wird für jeden Fehler die Ausprägung des ODC-Attributs Auslöser (trigger) bestimmt.
- Die Fehlerdaten werden ausgewertet, um ausgewählte Fragen zu beantworten.

Subjekte bzw. Objekte der Untersuchung (S. 670 f., 674)

- Untersucht werden die Implementierungsfehler einer Software aus einem abgeschlossenen Weiterentwicklungsprojekt.
- Die entwickelte Software ist eine Web-Anwendungssoftware mit 3-Schichten-Architektur, die stetig weiterentwickelt wird.
- Die Software hat einen Umfang von ca. 1000 Function Points. In Tests wurden etwas über 1000 Fehler entdeckt. Folglich ist die im Test entdeckte Fehlerdichte ca. 1 Fehler pro Function Point.
- Nach Schätzung der Autoren kann das Softwareentwicklungsteam zwischen dem CMM Level 2 und 3 eingeordnet werden.

Wesentliche Ergebnisse der Untersuchung (S. 677)

Durch die Anwendung von ODC können folgende Punkte erreicht werden:

- Aufzeigen von Verbesserungspotenzialen in den Prüf- und Testprozessen.
- Aufzeigen von Verbesserungspotenzialen im Entwurf und der Implementierung.
- Bewertung der Effektivität durchgeführter Verbesserungsmaßnahmen.
- Bewertung der Entwicklung der Softwarequalität.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC17 **Kurztitel:** Chillarege, Prasad /ODC Triggers/		**Jahr der Veröffentlichung**: 2002	
Darstellung der Literaturquelle			
Softwareart	Anwendungssoftware		
Eingesetzte ODC-Attribute für Fehlerfindung	Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	keine		
Zusätzliche Attribute für Fehler	Priorität (priority) (S. 673), Komponente (component) (S. 676)		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	Auslöser (trigger) (S. 675) Auslöser (trigger) x Zeitraum (S. 676) Auslöser (trigger) x Komponente (component) (S. 676)		
Anzahl Fehler	1096 Fehler		
Entwicklungsfehler	ja	Produktionsfehler	nein
Vorgehensweise bei Fehler-klassifikation	indirekt		

Informationen zur Literaturquelle	
Nr.: ODC18 **Kurztitel:** Christmansson, Santhanam /Error injection/	**Jahr der Veröffentlichung**: 1996

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 175 f.)

- In welche ungültigen Zustände muss eine Software in Tests künstlich versetzt werden, um Fehler in den Fehlertoleranzmechanismen[822] der Software aufzudecken?
- Untersuchungseinheit: Auswahl ungültiger Zustände einer Software

Vorgehensweise der Untersuchung (S. 177-181)

- ODC wird um einen Ansatz erweitert, um ungültige Zustände einer Software zu bestimmen, um die Fehlertoleranz der Software zu bewerten.
- Der Vorschlag wird auf Implementierungsfehler eines Betriebssystems angewandt, die nach der Auslieferung der Software entdeckt und nach ODC klassifiziert worden sind.

Subjekte bzw. Objekte der Untersuchung (S. 177)

- Untersucht werden alle Produktionsfehler eines Betriebssystems, die im Zeitraum von 2 Jahren nach Auslieferung durch Kunden der Software entdeckt worden sind.

Wesentliche Ergebnisse der Untersuchung (S. 182)

Folgende Auswahlkriterien zur Bestimmung der ungültigen Zustände einer Software sind zweckmäßig:

- Ungültige Zustände, die Fehler simulieren, deren Fehlverhalten zur erheblichen Einschränkung bei der Nutzung der Software seitens der Kunden führen.
- Ungültige Zustände, die Fehler simulieren, welche die Ausprägung „Recovery/Exception“ des ODC-Attributs Auslöser (trigger) haben.

[822] Unter der Fehlertoleranz einer Software wird die Eigenschaft verstanden, dass die Software ihre Funktionen trotz der Existenz von Fehlern erfüllt. Vgl. hierzu IEEE /Glossary/ 31.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC18 **Kurztitel:** Christmansson, Santhanam /Error injection/		**Jahr der Veröffentlichung**: 1996	
Darstellung der Literaturquelle			
Softwareart	Systemsoftware (Betriebssystem)		
Eingesetzte ODC-Attribute für Fehlerfindung	Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type)		
Zusätzliche Attribute für Fehler	Komponente (component) (S. 179)		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	Kategorien für ungültige Zustände einer Software		
Quantitative ODC-Daten	Auslöser (trigger) (S. 176) Fehlertyp (defect typ) (S. 178) Komponente (component) x Auslöser (trigger) (S. 181)		
Anzahl Fehler	408 Fehler		
Entwicklungsfehler	nein	Produktionsfehler	ja
Vorgehensweise bei Fehlerklassifikation	keine Angaben		

Informationen zur Literaturquelle

Nr.: ODC19 **Kurztitel:** Dalal u. a. /Defect patterns/ **Jahr der Veröffentlichung**: 1999

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 1)

- Die Autoren berichten über ihre Erfahrungen, die sie im Unternehmen Bellcore gemacht haben, um ODC einzuführen und unternehmensweit einzusetzen.
- Untersuchungseinheit: ODC in Anwendung.

Vorgehensweise der Untersuchung (S. 1)

- Darstellung der Ausgangssituation bei Bellcore.
- Darstellung einer beispielhaften Anwendung von ODC.

Subjekte bzw. Objekte der Untersuchung (S. 5)

- Es werden die klassifizierten Fehlerdaten von verschiedenen Versionen von drei nicht näher beschriebenen Softwareprodukten untersucht: Produkt A (Version 2 und 3), Produkt B (Version 3) und Produkt C (Version 3).
- Dabei werden zum einen die Fehler zweier Versionen des gleichen Softwareprodukts und zum anderen die Fehler von Versionen unterschiedlicher Softwareprodukte miteinander verglichen.
- Es handelt sich ausschließlich um Fehler, die jeweils nach Auslieferung der Softwareprodukte durch die Kunden entdeckt worden sind.

Wesentliche Ergebnisse der Untersuchung (S. 2-10)

- ODC kann bei Bellcore unternehmensweit unterstützt durch eine Web-Anwendungssoftware erfolgreich eingesetzt werden.
- Durch die Auswertung und den Vergleich der Fehlerdaten verschiedener Projekte werden Verbesserungspotenziale in den Entwicklungsaufgaben und der Qualitätssicherung aufgezeigt.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC19 **Kurztitel:** Dalal u. a. /Defect patterns/		**Jahr der Veröffentlichung**: 1999	
Darstellung der Literaturquelle			
Softwareart	keine Angaben		
Eingesetzte ODC-Attribute für Fehlerfindung	Aktivität (activity[823]), Auslöser (Trigger), Auswirkung (impact)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type), Kennzeichner (qualifier[824]), Alter (age[825])		
Zusätzliche Attribute für Fehler	Fehlerschwere (severity) (S. 2), Komponente (defect domain) (S. 2), Phase eingeführt (fault origin) (S. 2)		
Fehlerauswertung mit Attribute Focusing	ja[826]		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	keine		
Anzahl Fehler	Produkt A, Version 2: 31 Fehler Produkt A, Version 3: 50 Fehler Produkt A, Version 2 + 3: 81 Fehler (davon 10 Fehler mit Fehlerschwere 2) Produkt B, Version 3: 38 Fehler (ausschließlich Fehler mit Fehlerschwere 2) Produkt C, Version 3: 309 Fehler (davon 134 Fehler mit Fehlerschwere 2 und 175 Fehler mit Fehlerschwere 3)		
Entwicklungsfehler	nein	Produktionsfehler	ja
Vorgehensweise bei Fehlerklassifikation	keine Angaben		

[823] In Dalal u. a. /Defect patterns/ wurde das ODC-Attribut „activity“ in „lifecycle“ umbenannt.

[824] In Dalal u. a. /Defect patterns/ wurde das ODC-Attribut „qualifier“ in „modifier“ umbenannt.

[825] In Dalal u. a. /Defect patterns/ wird gemäß ODC 5.11 das Attribute „age“ verwendet, in dem Beitrag jedoch als „source“ bezeichnet.

[826] Die Methode „Attribute Focusing“ wird im Text nicht namentlich explizit genannt, sondern nur beschrieben. Literaturverweise auf Quellen zu „Attribute Focusing“ zeigen jedoch, dass eindeutig „Attribute Focusing“ eingesetzt wurde.

Informationen zur Literaturquelle

Nr.: ODC20 **Kurztitel:** Damm, Lundberg /Test process improvement/ **Jahr der Veröffentlichung**: 2005

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 152 f.)

- Wie kann ODC eingesetzt werden, um Fehler früher zu finden?
- Untersuchungseinheit: ODC in Anwendung

Vorgehensweise der Untersuchung (S. 155 f.)

- Auswertung des ODC-Attributs Auslöser zusammen mit dem wirtschaftlichsten Soll-Entdeckungszeitpunkt eines Fehlers.

Subjekte bzw. Objekte der Untersuchung (S. 156)

- ODC wird in einem Softwareentwicklungsprojekt bei Ericsson AB angewandt.
- Die Softwareentwicklungsabteilung von Ericsson AB betreut mehrere parallel laufende Projekte.
- In der Regel sind es Weiterentwicklungsprojekte, d. h. zu bestehenden Softwareprodukten werden neue Versionen entwickelt.
- Die Softwareentwicklungsprojekte dauern durchschnittlich 1 bis 1,5 Jahre und es wirken jeweils ungefähr 50 Personen mit.
- Die Softwareprodukte werden überwiegend in C++ entwickelt.

Wesentliche Ergebnisse der Untersuchung (S. 157-160)

- Das (angepasste) ODC-Attribut Auslöser und die Bestimmung des wirtschaftlichsten Soll-Entdeckungs-zeitpunks eines Fehlers können erfolgreich angewandt werden,
  - um Verbesserungspotenziale in den Testaktivitäten aufzuzeigen und
  - um zu bewerten, wie konsequent eine definierte Teststrategie umgesetzt wird.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC20 **Kurztitel:** Damm, Lundberg /Test process improvement/ **Jahr der Veröffentlichung**: 2005			
Darstellung der Literaturquelle			
Softwareart	systemnahe Software (für die Telekommunikationsbranche)		
Eingesetzte ODC-Attribute für Fehlerfindung	Auslöser (trigger), Aktivität (activity, in der Quelle als „phase found“ bezeichnet)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	keine		
Zusätzliche Attribute für Fehler	wirtschaftlichster Soll-Entdeckungszeit (phase belonging) (S. 154)		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	Die Ausprägungen des ODC-Attributs Auslöser (trigger) werden neu bestimmt. (S. 155)		
Erweiterungen des ODC-Verfahrens	Fault-Slip-Through (FST): Vorschlag zur Bestimmung des Zeitpunkts, an dem es am wirtschaftlichten ist, einen Fehler zu entdecken. (S. 152 f.)		
Quantitative ODC-Daten	Phase gefunden (phase found) x Auslöser (trigger) (S. 157) wirtschaftlichster Soll-Entdeckungszeit (phase belonging) x Auslöser (trigger) (S. 157)		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	ja	Produktionsfehler	nein
Vorgehensweise bei Fehlerklassifikation	indirekt		

Informationen zur Literaturquelle

Nr.: ODC21 **Kurztitel:** El Emam, Wieczorek /Repeatability/ **Jahr der Veröffentlichung**: 1998

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 322)

- Ist die Klassifikation von Fehlern nach dem ODC-Attribut Fehlertyp (defect type) wiederholbar, d. h. unabhängig von einzelnen Personen?
- Untersuchungseinheit: Wiederholbarkeit der Fehlerklassifikation nach ODC

Vorgehensweise der Untersuchung (S. 323-328)

- Jeweils zwei Personen klassifizieren unabhängig voneinander als Vorbereitung auf Codeinspektionen Fehler nach dem ODC-Attribut Fehlertyp (defect type).
- Anhand statistischer Methoden wird überprüft, inwieweit die Klassifikation der Fehler übereinstimmt, die beide Personen gefunden haben.

Subjekte bzw. Objekte der Untersuchung (S. 323)

- Fehlerdaten stammen von einem Softwareentwicklungsprojekt eines deutschen Unternehmens.
- Die entwickelte Software ist eine Anwendungssoftware zur Datenanalyse und hat einen Umfang von 30000 Codezeilen.
- Es arbeiten bis zu 5 Personen an diesem Softwareentwicklungsprojekt.
- Fehler stammen aus zwei Codeinspektionen der Anwendungssoftware.

Wesentliche Ergebnisse der Untersuchung (S. 330)

- Die durch zwei Personen unabhängig durchgeführten Fehlerklassifikationen nach dem ODC-Attribut Fehlertyp (defect type) stimmen in ausreichendem Maße überein. Folglich ist eine gute Wiederholbarkeit des Fehlerklassifikationsschemas gegeben und deshalb für den praktischen Einsatz nutzbar.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC21 **Kurztitel:** El Emam, Wieczorek /Repeatability/		**Jahr der Veröffentlichung**: 1998	
Darstellung der Literaturquelle			
Softwareart	Anwendungssoftware		
Eingesetzte ODC-Attribute für Fehlerfindung	keine		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type)		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nicht relevant		
Veränderungen des ODC-Verfahrens	Bestimmte Ausprägungen des ODC-Attributs Fehlertyp (defect type) werden ignoriert und andere wiederum hinzugefügt. (S. 324)		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	Fehlertyp (defect type) (S. 329)		
Anzahl Fehler	605 Fehler (aus zwei Codeinspektionen)		
Entwicklungsfehler	ja	Produktionsfehler	nein
Vorgehensweise bei Fehler-klassifikation	direkt		

Informationen zur Literaturquelle

Nr.: ODC22 **Kurztitel:** Henningsson, Wohlin /Fault classification agreement/

Jahr der Veröffentlichung: 2004

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 95)

- Ist die Klassifikation von Fehlern nach dem ODC-Attribut Fehlertyp (defect type) wiederholbar, d. h. unabhängig von einzelnen Personen?
- Untersuchungseinheit: Wiederholbarkeit der Fehlerklassifikation nach ODC.

Vorgehensweise der Untersuchung (S. 96 f.)

- Teilnehmer eines Experiments klassifizieren unabhängig voneinander 30 Fehler nach dem ODC-Attribut Fehlertyp (defect type) auf der Grundlage von Fehlerberichten.
- Anhand statistischer Methoden wird überprüft, inwieweit die Klassifikation der Fehler übereinstimmt.
- Die Teilnehmer werden zusätzlich nach Abschluss der Fehlerklassifikation gebeten, einen Fragebogen auszufüllen. Die Auswertung der Fragebögen soll Erkenntnisse darüber liefern, wie unter anderem die nachträgliche Klassifikation von Fehlern verbessert werden kann.

Subjekte bzw. Objekte der Untersuchung (S. 97 f.)

- Die Fehlerberichte zu 30 Fehlern sind fiktiv und stammen nicht aus einem realen Softwareentwicklungsprojekt, sondern sind Bestandteil einer Lehrveranstaltung.
- Die acht Teilnehmer des Experiments sind Mitarbeiter eines Instituts einer schwedischen Universität.

Wesentliche Ergebnisse der Untersuchung (S. 100-104)

- Die 30 Fehler werden durch die acht Personen nur mit einer sehr geringen Übereinstimmung klassifiziert.
- Nach Auswertung der Fragebögen schlussfolgern die Autoren, dass die in den Fehlerberichten enthaltenen Informationen für eine Fehlerklassifikation unzureichend sind und dies die wahrscheinlichste Ursache für die geringe Übereinstimmung ist.
- Zusätzlich trägt nach ihrer Ansicht die zu kurze Einweisung der Teilnehmer in das ODC-Klassifikationsschema zur geringen Übereinstimmung bei.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC22 **Kurztitel:** Henningsson, Wohlin /Fault classification agreement/ **Jahr der Veröffentlichung**: 2004			
Darstellung der Literaturquelle			
Softwareart	keine Angaben		
Eingesetzte ODC-Attribute für Fehlerfindung	keine		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type)		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nicht relevant		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	keine		
Anzahl Fehler	30 fiktive Fehler		
Entwicklungsfehler	keine Angaben	Produktionsfehler	keine Angaben
Vorgehensweise bei Fehler-klassifikation	indirekt		

Informationen zur Literaturquelle

Nr.: ODC23 **Kurztitel:** Lutz, Mikulski /Anomalies/ **Jahr der Veröffentlichung**: 2004

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 172 f.)

- Die Autorinnen untersuchen Problemberichte gemäß ODC, die nach dem Start von 7 Raumsonden angelegt und als sicherheitskritisch eingestuft worden sind.
- Untersuchungseinheit: ODC im Kontext von unbemannten Raumfahrtsmissionen.

Vorgehensweise der Untersuchung (S. 174)

- Nach Anpassung des ODC-Klassifikationsschemas auf das spezifische Anwendungsgebiet werden Abweichungen (in der Quelle als Anomalien bezeichnet) nach dem angepassten ODC-Klassifikationsschema klassifiziert und anschließend ausgewertet.

Subjekte bzw. Objekte der Untersuchung (S. 172 f.)

- Untersucht werden 199 Problemberichte von insgesamt 7 Raumsondenmissionen.
- Die Raumsonden sind alle im Zeitraum von 1989 bis 1999 gestartet.
- Die in den Problemberichten beschriebenen Abweichungen wurden stets nach dem Start der jeweiligen Raumsonde entdeckt und werden alle als sicherheitskritisch eingestuft.
- Die Problemberichte beinhalten nicht nur Fehler in der Software, sondern auch weitere Ursachen für Abweichungen.

Wesentliche Ergebnisse der Untersuchung (S. 179)

- Die Autorinnen decken verschiedene Muster in der Menge klassifizierter Abweichungen auf.
- Auf der Grundlage dieser Abweichungsmuster schlagen sie Verbesserungen für zukünftige unbemannte Raumfahrtsmissionen vor, um die Sicherheit zu erhöhen.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC23 **Kurztitel:** Lutz, Mikulski /Anomalies/		**Jahr der Veröffentlichung**: 2004	
Darstellung der Literaturquelle			
Softwareart	Systemsoftware (eingebettete Software)		
Eingesetzte ODC-Attribute für Fehlerfindung	Aktivität (activity), Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type), Ziel (target)		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	Die Ausprägungen der Attribute Aktivität (activity), Auslöser (trigger), Ziel (target) und Fehlertyp (defect type) werden für das spezifische Anwendungsgebiet angepasst. (S. 175)		
Erweiterungen des ODC-Verfahrens	Die Anwendung von ODC wird nicht nur auf Fehler in einer Software beschränkt, sondern auch auf andere Arten von Abweichungen ausgeweitet. (S. 172-175)		
Quantitative ODC-Daten	Aktivität (activity) (S. 175) Auslöser (trigger) (S. 176) Ziel (target) (S.176) Auslöser (trigger) x Ziel (target) (S. 177)		
Anzahl Fehler	Untersucht werden 199 Problemberichte, die nicht nur ausschließlich Fehler beschreiben.		
Entwicklungsfehler	nein	Produktionsfehler	ja
Vorgehensweise bei Fehler-klassifikation	indirekt		

Informationen zur Literaturquelle

Nr.: ODC24 **Kurztitel:** Lutz, Mikulski /Requirements/ **Jahr der Veröffentlichung**: 2004

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 19 f.)

- Die Autorinnen untersuchen Änderungen von Softwareanforderungen, die durch Abweichungen in der Software für Raumsonden hervorgerufen werden. Diese Abweichungen werden während der Tests oder des operativen Betriebs der Software festgestellt.
- Untersuchungseinheit: unvorhergesehene, durch Abweichungen in der Software verursachte Änderungen von Softwareanforderungen.

Vorgehensweise der Untersuchung (S. 20)

- Anhand eines für das spezifische Anwendungsgebiet angepasste ODC-Klassifikationsschemas werden Abweichungen (in der Quelle als Anomalien bezeichnet) klassifiziert und anschließend ausgewertet.

Subjekte bzw. Objekte der Untersuchung (S. 20)

- Untersucht werden 436 Problemberichte der Software für die US-amerikanische Raumsonde Mars Exploration Rover. Die Problemberichte stammen aus dem Integrations- und Systemtest der Software.
- Zusätzlich wurden ca. 200 Problemberichte von 7 weiteren Raumsondenmissionen untersucht. Die in den Problemberichten beschriebenen Abweichungen wurden stets nach dem Start der jeweiligen Raumsonde entdeckt und werden alle als sicherheitskritisch eingestuft.
- Alle untersuchten Problemberichte beinhalten nicht nur Fehler in der Software, sondern auch weitere Ursachen für Abweichungen.

Wesentliche Ergebnisse der Untersuchung (S. 21-24)

- Die Autorinnen identifizieren vier Mechanismen, wie mit unvorhergesehenen Änderungen von Softwareanforderungen umgegangen wird:
  - Problemberichte weisen auf eine fehlende oder unvollständige Softwareanforderung hin. Der Sachverhalt wird durch eine Änderung der Software gelöst.
  - Problemberichte weisen auf eine fehlende oder unvollständige Softwareanforderung hin. Der Sachverhalt wird durch eine Änderung des operativen Arbeitsablaufs abgefangen. Eine Änderung der Software wird umgangen.
  - Problemberichte weisen auf eine (korrekte) Softwareanforderung hin, die jedoch von Verantwortlichen für Tests oder den Betrieb der Software missverstanden wird. Durch eine Änderung der Dokumentation der Software sollen entsprechende Missverständnisse unterbunden werden.
  - Problemberichte weisen auf eine (korrekte) Softwareanforderung hin, die jedoch von Verantwortlichen für Tests oder den Betrieb der Software missverstanden wird. Auf diesen Sachverhalt wird durch keine Maßnahme reagiert (z. B. weil die für die beschriebene Abweichung erforderlichen Voraussetzungen nicht wieder auftreten werden).
- Die Autorinnen beschreiben die mittels des angepassten ODC-Klassifikationsschemas festgestellten Muster in den Abweichungen und machen Vorschläge zum besseren Umgang mit unvorhergesehenen Änderungen von Softwareanforderungen.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC24 **Kurztitel:** Lutz, Mikulski /Requirements/		**Jahr der Veröffentlichung**: 2004	
Darstellung der Literaturquelle			
Softwareart	eingebettete Software		
Eingesetzte ODC-Attribute für Fehlerfindung	Aktivität (activity), Auslöser (trigger)		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type)		
Zusätzliche Attribute für Fehler	nein		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	Die Ausprägungen der Attribute Aktivität (activity), Auslöser (trigger), Ziel (target) und Fehlertyp (defect type) werden für das spezifische Anwendungsgebiet angepasst. (S. 20 f.)		
Erweiterungen des ODC-Verfahrens	Die Anwendung von ODC wird nicht nur auf Fehler in einer Software beschränkt, sondern auch auf andere Arten von Abweichungen ausgeweitet. (S. 20 f.)		
Quantitative ODC-Daten	keine		
Anzahl Fehler	436 Problemberichte aus dem Integrations- und Systemtest der Software für die Raumsonde Mars Exploration Rover. 200 sicherheitskritische Problemberichte von 7 weiteren Raumsondenmissionen, die nach dem Start der jeweiligen Raumsonde entdeckt werden. Die Problemberichte beschreiben nicht Fehler in der jeweiligen Software, insofern ist die genaue Anzahl der Fehler nicht angegeben.		
Entwicklungsfehler	ja	Produktionsfehler	ja
Vorgehensweise bei Fehlerklassifikation	indirekt		

Informationen zur Literaturquelle	
Nr.: ODC25 **Kurztitel:** Sullivan, Chillarege /Comparison/	**Jahr der Veröffentlichung**: 1992

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 475 f.)

- Die Autoren vergleichen die Ergebnisse der Fehleranalyse von drei IBM-Produkten (DB2, IMS und MVS).
- Untersuchungseinheit: Verfahren zur Fehleranalyse

Vorgehensweise der Untersuchung (S. 476-478)

- Strichproben von ausgelieferten Fehlern der drei Softwareprodukte werden nachträglich anhand eines Klassifikationsschemas klassifiziert und ausgewertet.

Subjekte bzw. Objekte der Untersuchung (S. 476 f.)

- Untersucht werden ausgelieferte Fehler der zwei Datenbankmanagementsysteme DB2 und IMS sowie des Betriebssystems MVS.

Wesentliche Ergebnisse der Untersuchung (S. 484)

- Die drei Softwareprodukte weisen jeweils eine unterschiedliche Verteilung bzgl. der Codefehler auf.
- Das Datenbankmanagementsystem IMS hat im Vergleich zu DB2 weniger Softwarefehler, die durch eine Auslastung der Software ausgelöst werden.
- Softwarefehler in DB2 haben größere Auswirkungen auf die Kunden als bei IMS und MVS.
- Die Fehleranalyse der drei Softwareprodukte ergibt, dass jeweils bei einem hohen Anteil der Fehler, die Software in einen undefinierten Zustand wechselt. Mittels der Fehlerauswertungen werden die Ursachen für entsprechende Fehler untersucht.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC25 **Kurztitel:** Sullivan, Chillarege /Comparison/		**Jahr der Veröffentlichung**: 1992	
Darstellung der Literaturquelle			
Softwareart	Systemnahe Software und Systemsoftware (die zwei Datenbankmanagement-systeme DB2 und IMS und das Betriebssystem MVS)		
Eingesetzte ODC-Attribute für Fehlerfindung	Auslöser (trigger)[827]		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type)		
Zusätzliche Attribute für Fehler	Fehlerschwere (severity) (S. 477), Codefehlertyp (error type) (S. 477)		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	keine		
Anzahl Fehler	DB2: 222 Fehler IMS: 201 Fehler MVS: 150 Fehler		
Entwicklungsfehler	nein	Produktionsfehler	ja
Vorgehensweise bei Fehler-klassifikation	indirekt		

[827] Wird in der Quelle als „error trigger“ bezeichnet und kann als Vorläufer des aus ODC bekannten „defect trigger“-Attributs betrachtet werden. Die Ausprägungen der beiden Attribute unterscheiden sich jedoch deutlich. Vgl. Sullivan, Chillarege /Comparison/ 478.

Informationen zur Literaturquelle

Nr.: ODC26 **Kurztitel:** Halliday u. a. /Experiences/ **Jahr der Veröffentlichung**: 1994

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 61)

- Die Autoren beschreiben ihre Erfahrungen, die sie bei der Einführung und Umsetzung von ODC in verschiedenen Entwicklungszentren von IBM gemacht haben.
- Untersuchungseinheit: Einführung und Umsetzung von ODC.

Vorgehensweise der Untersuchung (S. 61)

- Die Autoren stellen die Faktoren dar, die nach ihrer Ansicht eine erfolgreiche Einführung und Umsetzung von ODC in Softwareentwicklungsprojekten unterstützen oder hindern.

Subjekte bzw. Objekte der Untersuchung (S. 61)

- ODC wird in verschiedenen nicht näher beschriebenen Entwicklungszentren von IBM eingeführt.

Wesentliche Ergebnisse der Untersuchung (S. 62-67)

- Pilotprojekte gewährleisten alleine nicht eine erfolgreiche Umsetzung von ODC in anderen Softwareentwicklungsprojekten.
- Die Klassifikation von Fehlern muss nicht immer durch Werkzeuge automatisiert werden.
- Der Technologietransfer zwischen Forschung und Praxis erfolgt wechselseitig.
- Für eine erfolgreiche Einführung von ODC müssen sowohl das Management als auch Mitarbeiter auf Projektebene überzeugt werden.
- Der Zeitpunkt der Einführung von ODC in einem Softwareentwicklungsprojekt ist idealerweise zu Beginn des Projekts.
- Werkzeuge zur Unterstützung der Fehlerklassifikation müssen einfach gehalten werden.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC26 **Kurztitel:** Halliday u. a. /Experiences/		**Jahr der Veröffentlichung**: 1994	
Darstellung der Literaturquelle			
Softwareart	keine Angaben		
Eingesetzte ODC-Attribute für Fehlerfindung	keine Angaben		
Eingesetzte ODC-Attribute für Fehlerkorrektur	keine Angaben		
Zusätzliche Attribute für Fehler	keine		
Fehlerauswertung mit Attribute Focusing	ja (S. 62, 67)		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	keine		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	keine Angaben	Produktionsfehler	keine Angaben
Vorgehensweise bei Fehlerklassifikation	keine Angaben		

Informationen zur Literaturquelle

Nr.: ODC27 **Kurztitel:** Zheng u. a. /Static analysis/ **Jahr der Veröffentlichung**: 2006

Darstellung der Literaturquelle

Ziele der Untersuchung (S. 240 f.)

- Kann eine automatische Codeprüfung mittels Software-Werkzeugen eine Organisation dabei unterstützen, die Qualität von Softwareprodukten wirtschaftlich zu verbessern?
- Untersuchungseinheit: automatische Prüfung von Softwarecode mittels Software-Werkzeugen.

Vorgehensweise der Untersuchung (S. 243-250)

- Ausgehend von dem Hauptziel der Untersuchung leiten die Autoren Unterfragen ab.
- Um die Unterfragen zu beantworten, werden Kennzahlen definiert und für die untersuchten Softwareprodukte erhoben.

Subjekte bzw. Objekte der Untersuchung (S. 242 f.)

- Untersucht werden die Fehlerberichte von Kunden und über 200 Inspektoren und Testern zu drei Softwareprodukten für Netzwerke von Nortel Networks.
- Die Produkte umfassen mehr als 3 Millionen Codezeilen, die in C/C++ geschrieben worden sind.

Wesentliche Ergebnisse der Untersuchung (S. 243-251)

- Die Kosten für einen gefundenen Fehler liegen bei der automatischen Codeprüfung mittels Software-Werkzeugen in der gleichen Größenordnungen wie bei Codeinspektionen.
- Die Fehlerfindungseffektivität einer automatischen Codeprüfung mittels Software-Werkzeugen unterscheidet sich nicht signifikant von der einer Inspektion. Ein Softwaretest ist zwei- bis dreimal effektiver als die automatische Codeprüfung.
- Die Anzahl der Fehler, die in einem Softwaremodul durch eine automatische Codeprüfung gefunden werden, ist für den Test ein geeigneter Indikator für fehlerbehaftete Softwaremodule.
- Mittels einer automatischen Codeprüfung werden gemäß dem ODC-Attribut Fehlertyp (defect type) insbesondere Bedingungs- und Zuweisungsfehler gefunden.
- 90% der mittels Codeinspektionen entdeckten Fehler sind gemäß dem ODC-Attribut Fehlertyp (defect type) insbesondere Algorithmus-, Dokumentations- und Bedingungsfehler.
- Die große Mehrheit der im Test und durch Kunden entdeckten Fehler sind gemäß dem ODC-Attribut Fehlertyp (defect type) Funktions- und Algorithmusfehler.
- Die meisten Fehler sind auf wenige Programmierer-Irrtümer zurückzuführen.
- Ein prozentual großer Anteil der durch eine automatische Codeprüfung gefundener Fehler hat das Potenzial Sicherheitslücken in der Software zu verursachen.

Informationen zur Literaturquelle (Fortsetzung)			
Nr.: ODC27 **Kurztitel:** Zheng u. a. /Static analysis/		**Jahr der Veröffentlichung**: 2006	
Darstellung der Literaturquelle			
Softwareart	systemnahe Software		
Eingesetzte ODC-Attribute für Fehlerfindung	keine Angaben		
Eingesetzte ODC-Attribute für Fehlerkorrektur	Fehlertyp (defect type)		
Zusätzliche Attribute für Fehler	keine		
Fehlerauswertung mit Attribute Focusing	nein		
Veränderungen des ODC-Verfahrens	nein		
Erweiterungen des ODC-Verfahrens	nein		
Quantitative ODC-Daten	Fehlertyp (defect type) (S. 247-249)		
Anzahl Fehler	keine Angaben		
Entwicklungsfehler	ja	Produktionsfehler	ja
Vorgehensweise bei Fehler-klassifikation	indirekt		

10 Anhang B: Mindmaps zu Prozessmängeln und Anforderungsfehlern

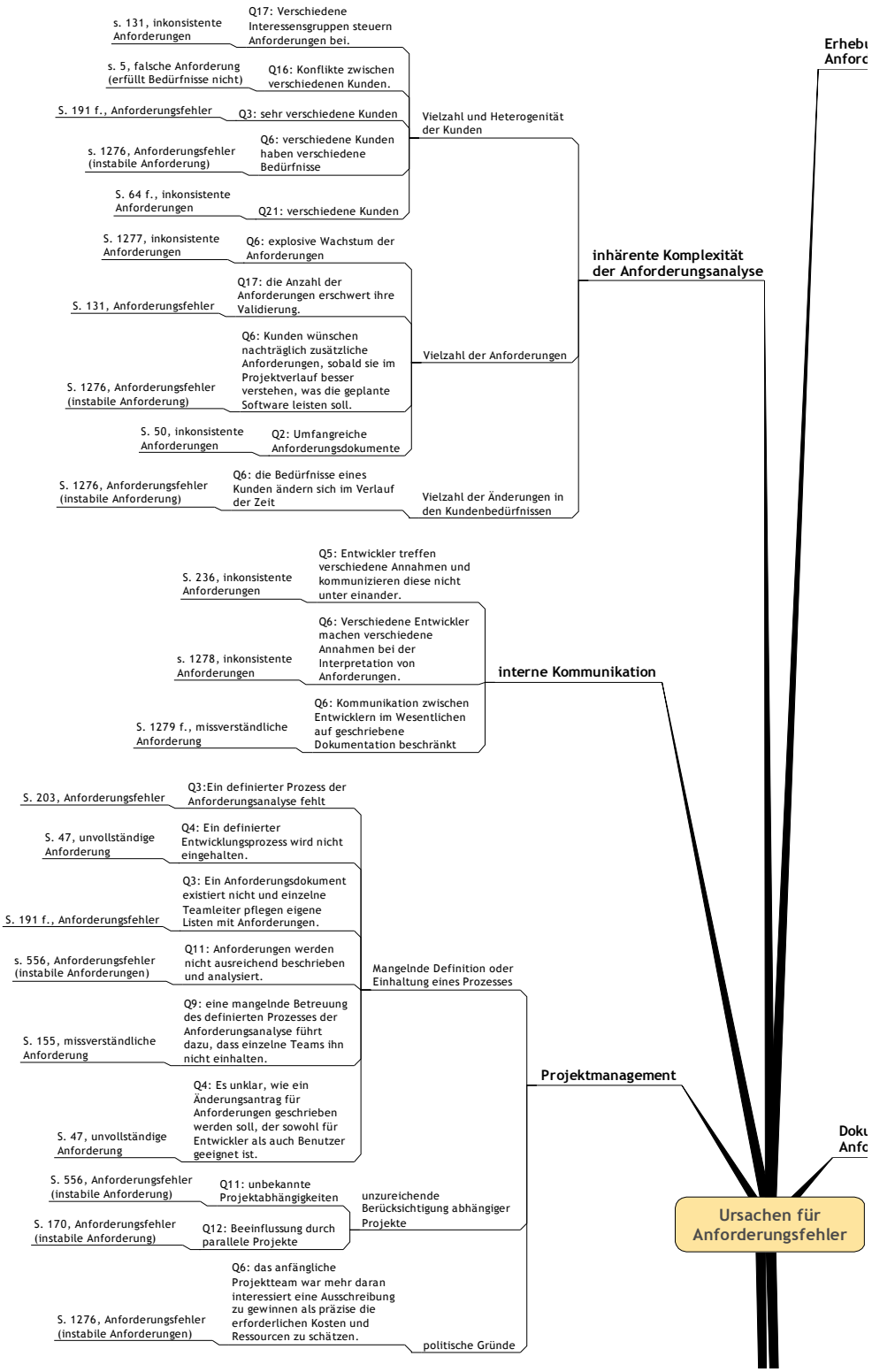


Abbildung 10-1: Mindmap – Prozessmängel und ihre Wirkungen (Teil 1 von 4)

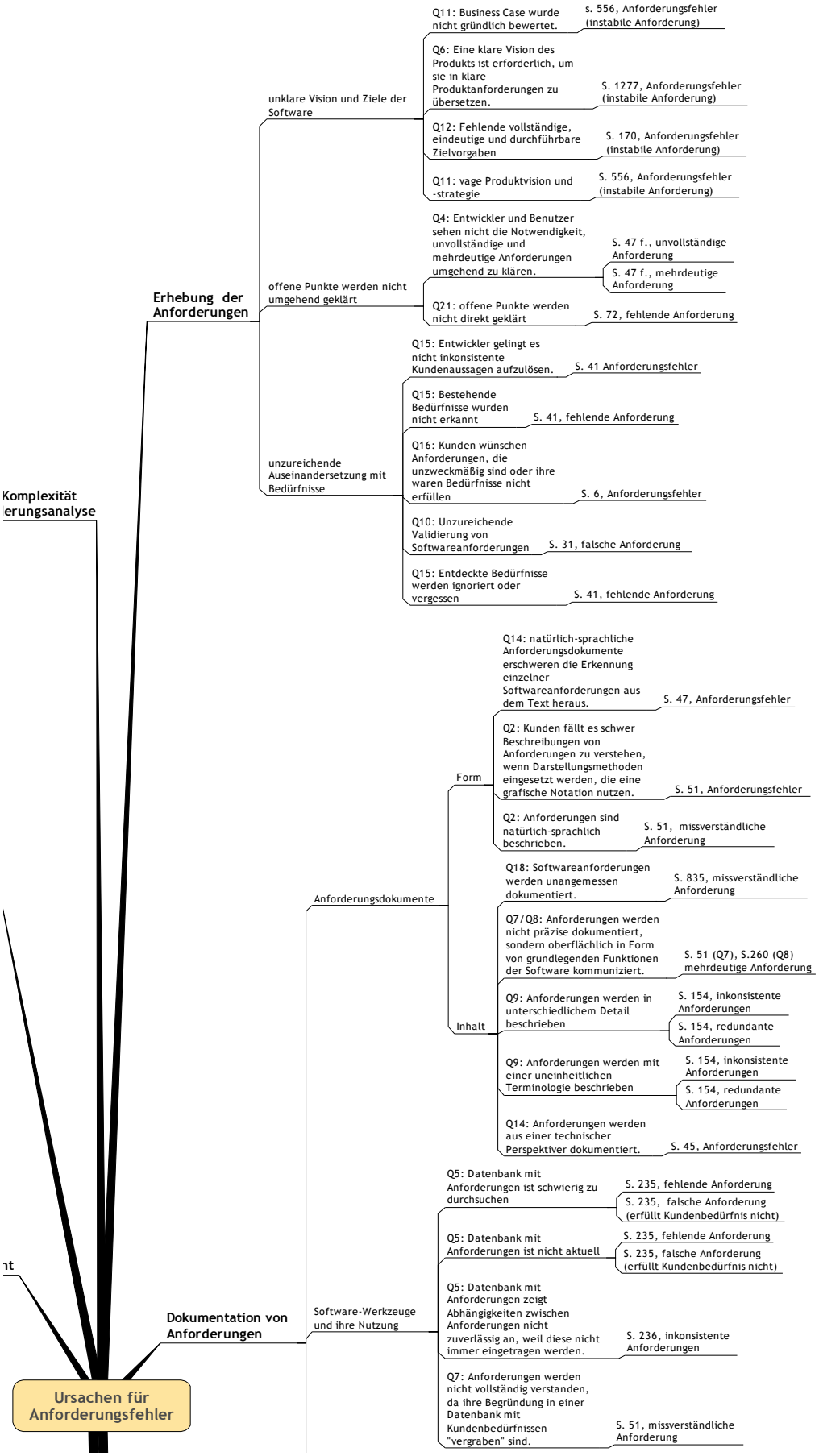


Abbildung 10-2: Mindmap – Prozessmängel und ihre Wirkungen (Teil 2 von 4)

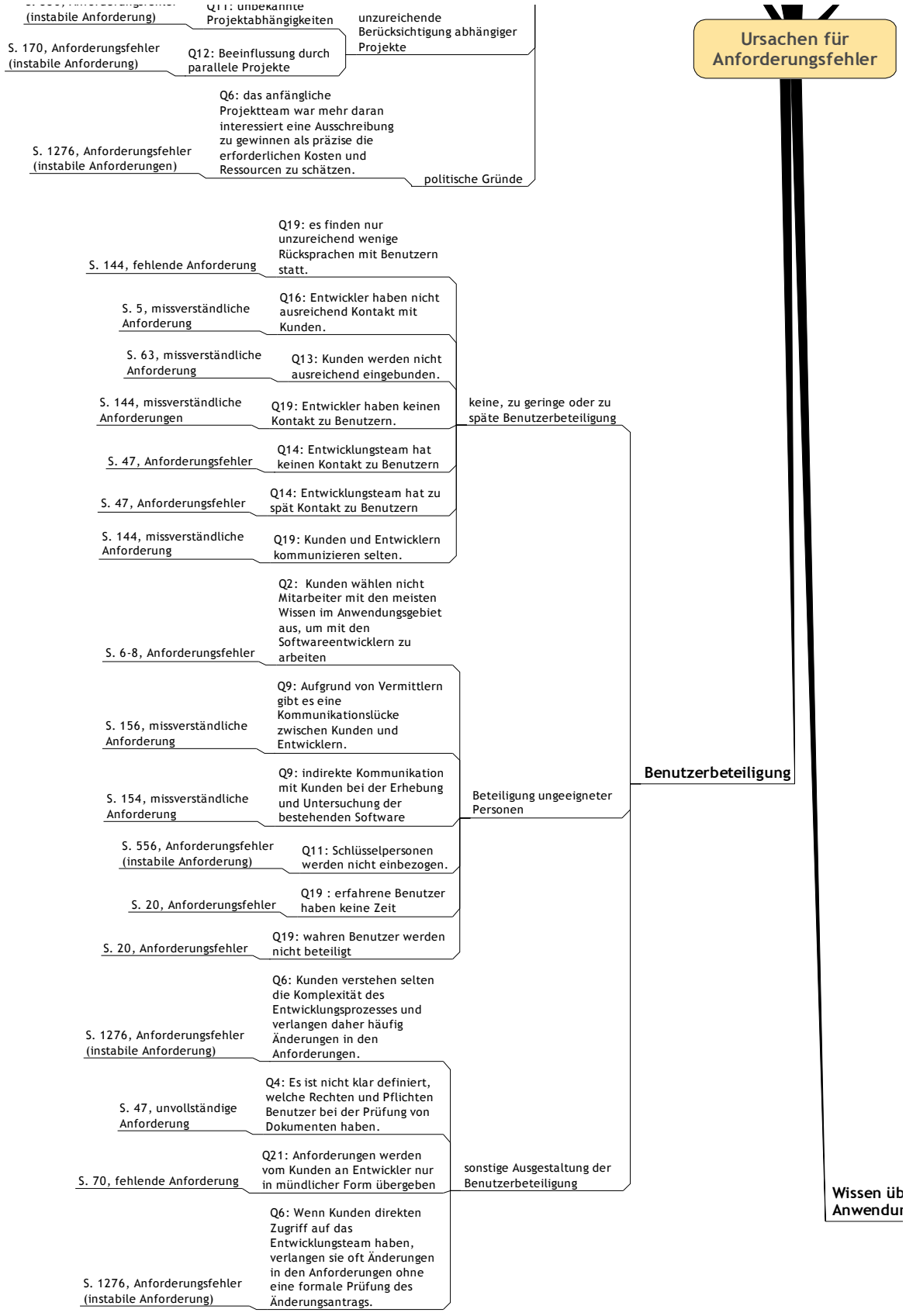


Abbildung 10-3: Mindmap – Prozessmängel und ihre Wirkungen (Teil 3 von 4)

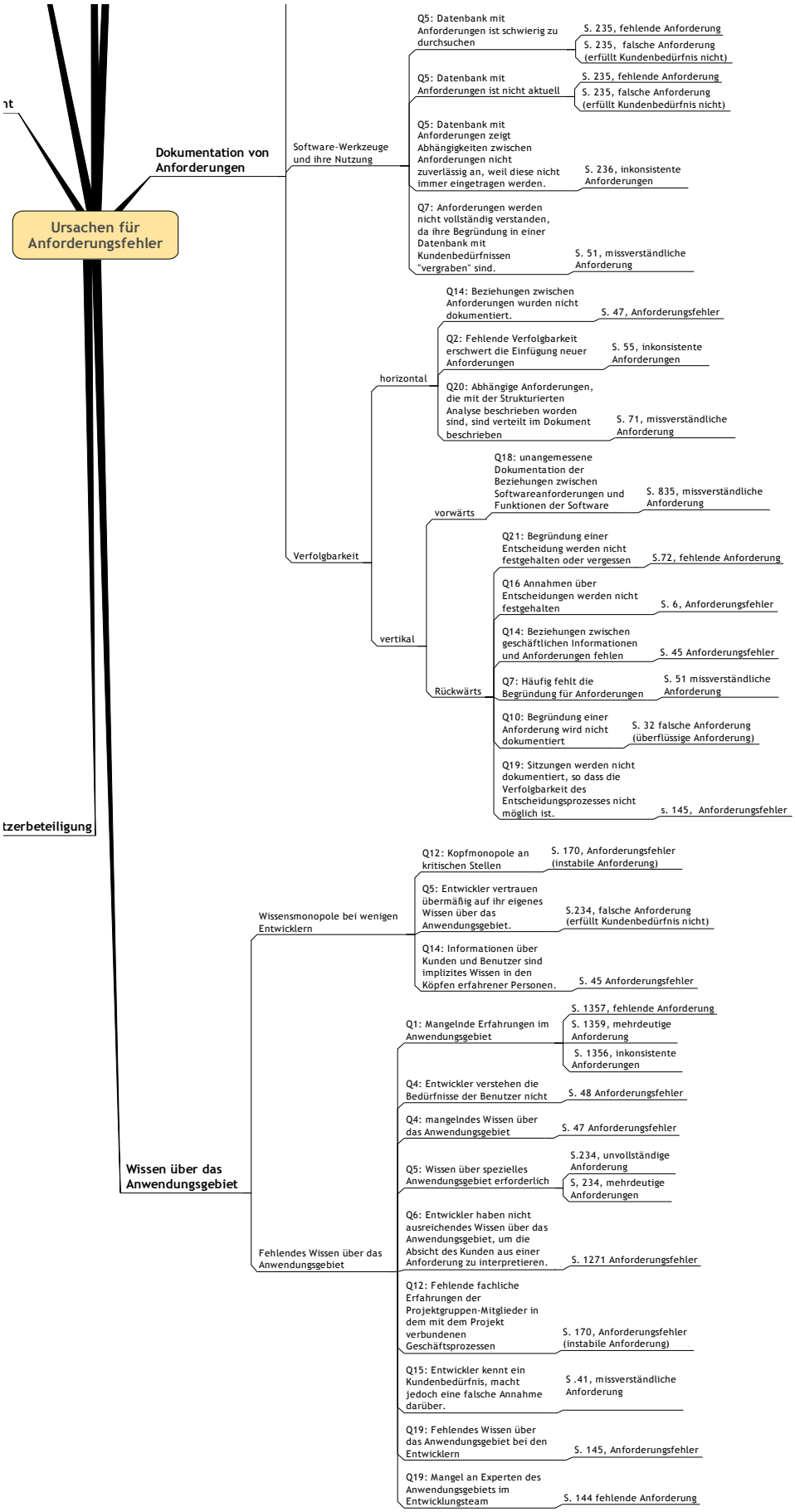


Abbildung 10-4: Mindmap – Prozessmängel und ihre Wirkungen (Teil 4 von 4)

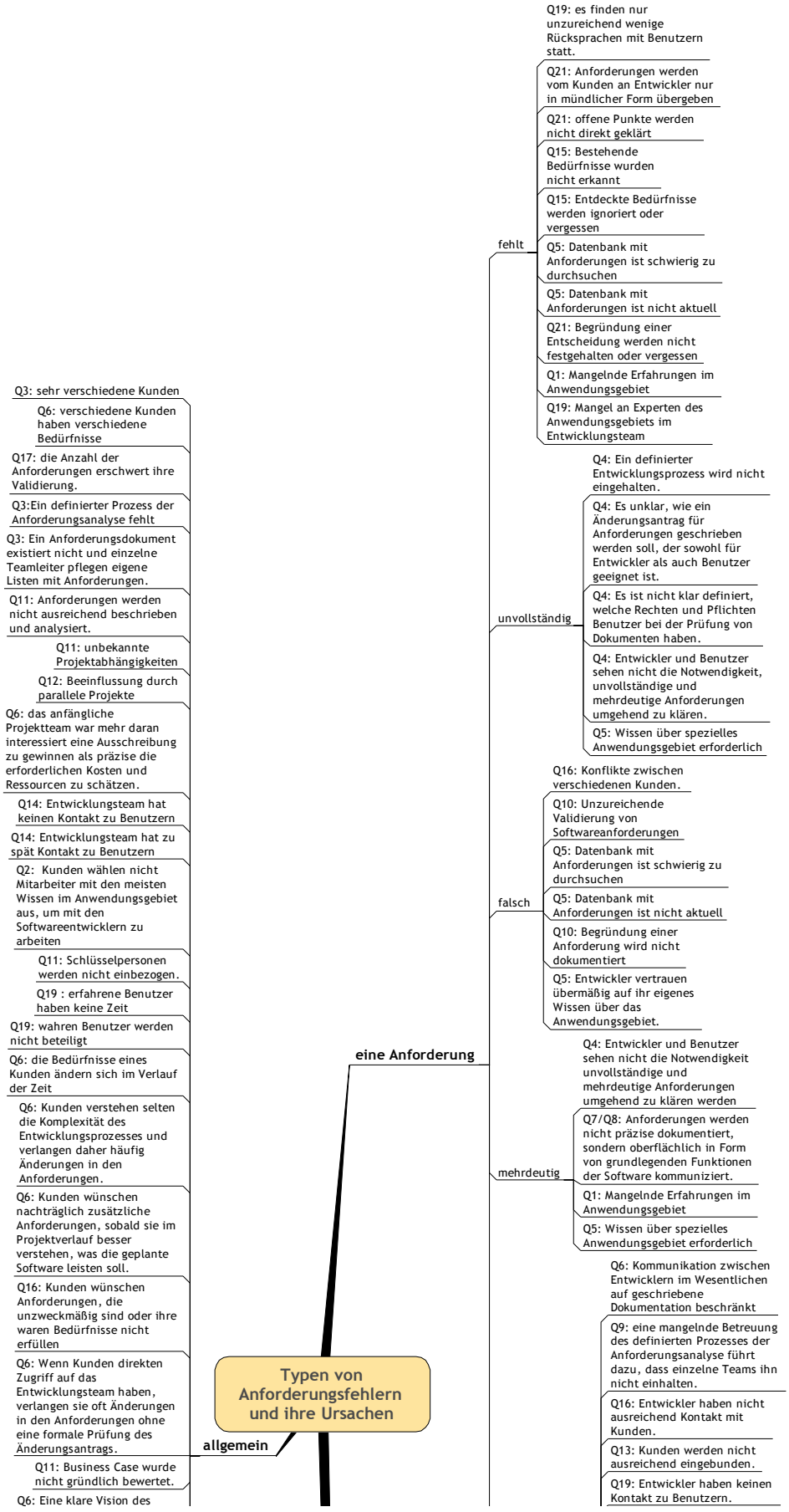


Abbildung 10-5: Mindmap – Anforderungsfehler und ihre Ursachen (Teil 1 von 2)

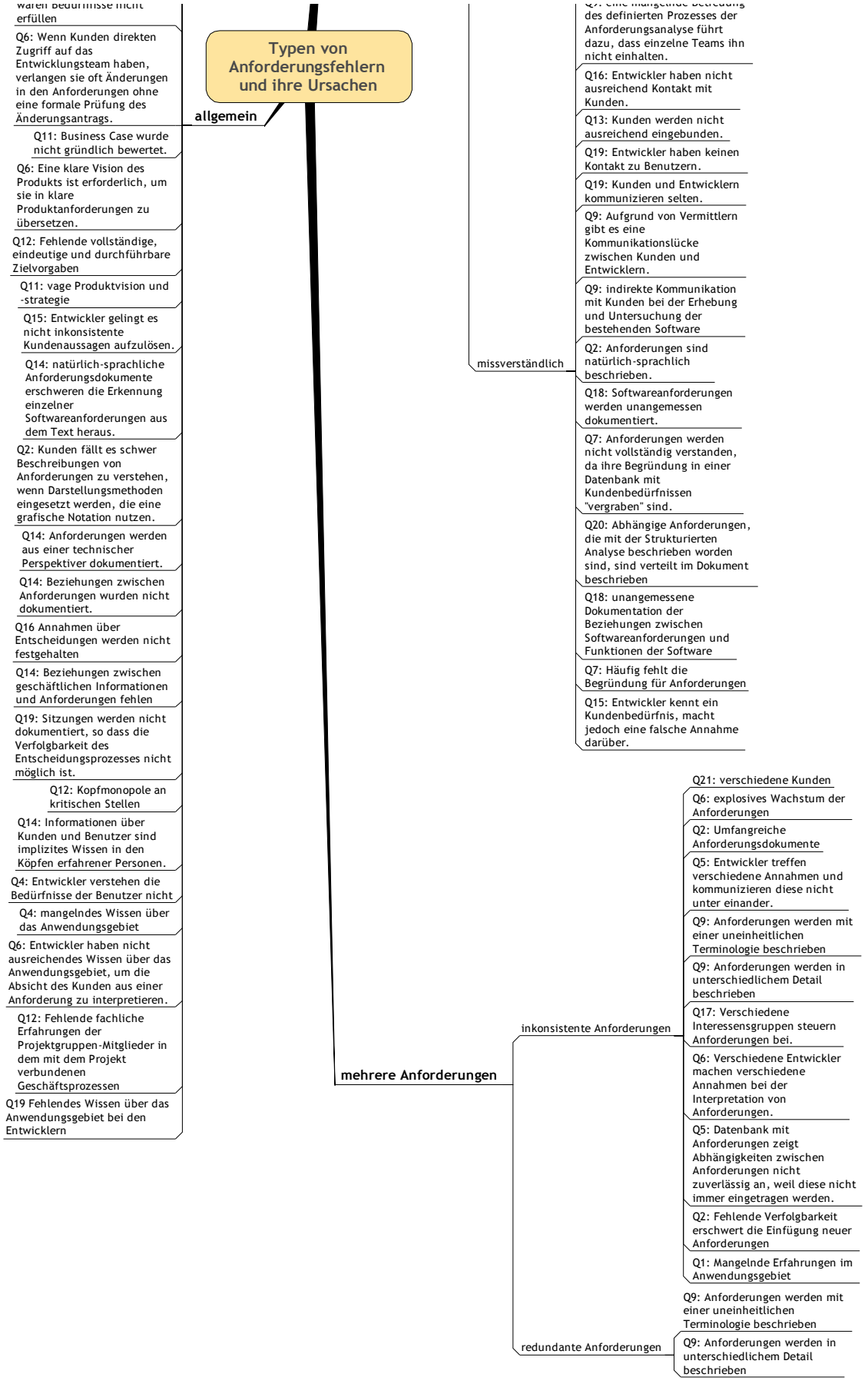


Abbildung 10-6: Mindmap – Anforderungsfehler und ihre Ursachen (Teil 2 von 2)

11 Anhang C: Fallstudie

11.1 Anhang C.1: Durch das Unternehmen bereitgestellte Dokumente

Die Tabelle 11-1 und Tabelle 11-2 listen die Dokumente auf, die durch das Unternehmen der Fallstudie bereitgestellt worden sind. Neben einer identifizierenden Nummer ist das Datum aufgeführt, an dem das Dokument zur Verfügung gestellt worden ist.

Nr.	Wann erhalten?	Inhalt des Dokuments	Anmerkungen zum Dokument
1	17.02.05	Bildschirmfotos (Screenshots) von zwei unterschiedlichen Softwareprodukten zur Fehlerverfolgung, die im Unternehmen eingesetzt werden.	Bei den Softwareprodukten handelt es sich zum einen um eine kommerzielle Fehlerverfolgungssoftware und zum anderen um eine Eigenentwicklung als Lotus-Notes-Datenbank.
2	18.04.05	Vertraulichkeitsvereinbarung	
3	04.05.05	Problemdatenbank zum Projekt LV-Neu, Stand 04.05.05.	In der Problemdatenbank werden entdeckte Fehler in LV-Neu verfolgt und Änderungsaufträge verwaltet.
4	04.05.05	Foliensatz zum überarbeiteten Softwareentwicklungsprozess, der für die Releaseentwicklung von LV-Neu ab Anfang 2006 umgesetzt werden soll. Kernpunkte: überarbeitetes Releasemanagement, abgestimmtes Konfigurations- und Changemanagement. Umfang: 24 Folien.	
5	04.05.05	Folie zur Aufteilung der Aufgaben im Rahmen der Releaseentwicklung für LV-Neu zwischen IT-Power und GDV.	7 unterschiedliche Bereiche aus IT-Power und GDV tragen zu Arbeitsergebnissen bei.
6	23.05.05	Steckbrief für die Anwendungssoftware LV-Neu. Umfang: 204 Seiten.	Das Dokument gibt unter anderem einen Überblick über die Aufgaben und Ziele der Anwendungssoftware LV-Neu, ihren technischen Aufbau sowie Schnittstellen zu Randsystemen.
7	23.05.05	Beispiel 1 für ein Fachkonzeptdokument aus dem Projekt LV-Neu. Umfang: 175 Seiten.	„Fachkonzept“ im Sinne von IT-Power: Das Dokument ist eine Mischung aus fachlichen Anforderungen als auch Entscheidungen aus dem Feinentwurf.
8	23.05.05	Beispiel 1 für ein Dokument zum Feinentwurf: Schnittstelle LV-Neu zum Randsystem Bankverbindung Umfang: 11 Seiten	
9	23.05.05	Beispiel 2 für ein Dokument zum Feinentwurf: Schnittstelle LV-Neu zu den Randsystemen Exkasso und Finanzbuchhaltung Umfang: 11 Seiten	
10	23.05.05	Beispiel 3 für ein Dokument zum Feinentwurf: Schnittstelle LV-Neu zu dem Randsystem Inkasso Umfang: 12 Seiten	

Tabelle 11-1: Durch das Unternehmen der Fallstudie bereitgestellte Dokumente

Nr.	Wann erhalten?	Inhalt des Dokuments	Anmerkungen zum Dokument
11	23.05.05	Beispiel für ein Dokument zum Grobentwurf: softwaretechnische Architektur von LV-Neu. Umfang: 53 Seiten.	
12	03.06.05	Projektplan für die Entwicklung der Releases 4.3 und 4.4 von LV-Neu. Umfang: 5 Seiten.	
13	24.06.05	Leitfaden zur Nutzung der Problemdatenbank. Umfang: 18 Seiten.	
14	12.07.05	Projektplan für Release 4.3 von LV-Neu inklusive QS-Prozesse und Testkriterien.	
15	12.07.05	Definition der QS-Prozesse im Projekt LV-Neu. Umfang: 8 Seiten.	
16	13.07.05	Projektplan für die Entwicklung der Releases 4.0 bis 4.2 von LV-Neu. Umfang: 9 Seiten.	
17	19.07.05	Patchplanung für Release 4.2 von LV-Neu (drei Excel-Dateien).	Ein Patch ist die Korrektur von Software-fehlern, nachdem die Software operativ genutzt wurde..
18	19.07.05	Patchplanung für Release 4.3 von LV-Neu (drei Excel-Dateien).	siehe Fallstudiendokument 17
19	22.07.05	Beispiel 1 für ein Ergebnisprotokoll eines QS-Review eines Fachkonzeptdokuments. Umfang: 4 Seiten.	
20	22.07.05	Beispiel 2 für ein Fachkonzeptdokument aus dem Projekt LV-Neu. Umfang: 134 Seiten.	Das QS-Review (Dokument 19) bezieht sich auf dieses Fachkonzeptdokument.
21	22.07.05	Beispiel 2 für ein Ergebnisprotokoll eines QS-Review eines Fachkonzeptdokuments. Umfang: 3 Seiten.	
22	22.07.05	Beispiel 3 für ein Ergebnisprotokoll eines QS-Review eines Fachkonzeptdokuments. Umfang: 2 Seiten.	.
23	01.08.05	Problemdatenbank zum Projekt LV-Neu, Stand 04.05.05.	In der Problemdatenbank werden entdeckte Fehler in LV-Neu verfolgt und Änderungs-aufträge verwaltet.
24	30.08.05	Projektplan für die Entwicklung der Releases 4.4 und 4.5 von LV-Neu.	
25	30.08.05	erweiterter Projektplan für die Entwicklung der Releases 4.4 und 4.5 von LV-Neu (inklusive QS-Prozesse).	
26	05.09.05	Testkonzept des Fachbereichs der GDV Umfang: 73 Seiten.	
27	23.09.05	Problemdatenbank zum Projekt LV-Neu, Stand 23.09.05.	In der Problemdatenbank werden entdeckte Fehler in LV-Neu verfolgt und Änderungs-aufträge verwaltet.

Tabelle 11-2: Durch das Unternehmen der Fallstudie bereitgestellte Dokumente

11.2 Anhang C.2: Im Rahmen der Fallstudie erstellte Dokumente

Die Tabelle 11-1 und Tabelle 11-2 listen die Dokumente auf, die der Autor der vorliegenden Arbeit auf der Grundlage von im Unternehmen der Fallstudie erhobenen Informationen erstellt hat. Diese wurden alle Verantwortlichen des Unternehmens präsentiert. In wichtigen Fällen wurden erstellte Dokumente explizit zur Abnahme vorgelegt. Neben einer identifizierenden Nummer[828] ist das Datum aufgeführt, an dem das Dokument fertig gestellt worden ist.

Nr.	Wann fertig gestellt?	Inhalt des Dokuments	Anmerkungen zum Dokument
28	22.07.05	Komponenten der Anwendungssoftware LV-Neu	Abnahme erfolgt.
29	26.07.05	Überarbeiteter Zeitplan für die Releases 4.2 und 4.3 für die Prozesse der Entwicklungsaufgaben und der Qualitätssicherung.	Abnahme erfolgt.
30	03.08.05	Folien zum Vortrag zu den vorläufigen Ergebnissen der Fehlerauswertung	
31	08.08.05	Zuordnung Testkriterien zu Teststufen	Abnahme erfolgt.
32	11.08.05	Vortrag „Konzept für eine neue Fehlerverfolgung“	
33	11.08.05	Zwischenergebnis für die Konfiguration der Fehlerverfolgungssoftware	Abnahme erfolgt.
34	22.08.05	Überarbeitete Folien zum Vortrag zu den vorläufigen Ergebnissen der Fehlerauswertung als Executive Management Summary	Dieser Foliensatz entspricht weitgehend dem Foliensatz unter der Nummer 30.
35	06.09.05	Auflistung Projektmitarbeiter und ihre Rolle im Projekt LV-Neu	Abnahme erfolgt.
36	06.09.05	Zusammenfassung des Testmanagements auf der Kundenseite	Abnahme erfolgt.
37	12.09.05	Zusammenfassung des gesamten Testmanagements auf der Kunden- und Entwicklerseite	Abnahme erfolgt.
38	23.09.05	Messung des absoluten Codeumfangs für Release 8.2	Abnahme erfolgt.
39	19.10.05	Messung des absoluten Codeumfangs für Release 8.3	Abnahme erfolgt.
40	04.11.05	Alle bisher erstellten Zeitpläne für die Releases 8.2 und 8.3	Abnahme erfolgt.
41	17.11.05	Überarbeitete Auflistung Projektmitarbeiter und ihre Rolle im Projekt LV-Neu	Abnahme erfolgt.
42	22.11.05	Dateien für die Vorbereitung der Arbeitssitzung zur vollständigen Fehlerauswertung	
43	23.11.05	Folien zur vollständigen Fehlerauswertung	
44	23.11.05	Protokoll der Arbeitssitzung vom 23.11.05	Abnahme erfolgt.

Tabelle 11-3: Im Rahmen der Fallstudie erstellte Dokumente

[828] Die Nummerierung aus Tabelle 11-2 wird fortgeführt.

Nr.	Wann fertiggestellt?	Inhalt des Dokuments	Anmerkungen zum Dokument
45	29.11.05	Merkmale der untersuchten Anwendungsreleases von LV-Neu	Abnahme erfolgt.
46	30.11.05	Dateien aus der Vorbereitung des dritten Schritts des Verfahrens	
47	01.12.05	Folien zu der Arbeitssitzung „Ursachenanalyse für ausgewählte Fehlermuster“	
48	01.12.05	Protokoll der Arbeitssitzung vom 01.12.05	Abnahme erfolgt.
49	31.01.06	Zusammenstellung der Ergebnisse für das Unternehmen	Abnahme erfolgt.
50	28.02.06	Entwickeltes Programm zur Messung des relativen Codeumfangs eines Anwendungsreleases	Vgl. Anhang C.4
51	15.03.06	Ergebnisse der ersten Messung zum relativen Codeumfangs	Vgl. Anhang C.4
52	17.03.06	Ergebnisse der überarbeiteten Messung zum relativen Codeumfangs (automatisch generierten Code entfernt)	Vgl. Anhang C.4

Tabelle 11-4: Im Rahmen der Fallstudie erstellte Dokumente (Fortsetzung)

11.3 Anhang C.3: Wichtige Ereignisse: Interviews, E-Mails, etc.

Die Tabelle 11-5 bis Tabelle 11-10 listen alle Ereignisse im Rahmen der Fallstudie auf, in der für die Ergebnisse der Fallstudie relevante Informationen erhoben worden sind. Alle Interviews und Arbeitssitzungen wurden protokolliert. Die Gesprächsprotokolle wurden den Teilnehmern jeweils mit der Bitte um eine Abnahme zeitnah zur Verfügung gestellt. Die Teilnehmer werden nicht namentlich genannt, sondern mit einer Rollenbezeichnung. Die Rollen werden im Anhang C.4 näher beschrieben.

Nr.	Datum	Ereignis	Inhalt
1	**01.06.05**	**Interview (Dauer: 60 Minuten)** Teilnehmer: ▪ interner Berater 1 (IT-Power) ▪ SHK 1 (Lehrstuhk) ▪ Autor (Lehrstuhl)	▪ Abstimmung weiteres Vorgehen ▪ Allgemeine Informationen zu IT-Power ▪ Softwareentwicklungsprojekte bei IT-Power
2	**03.06.05**	**Interview (Dauer: 120 Minuten)** Teilnehmer: ▪ interner Berater 1 (IT-Power) ▪ Testmanager 1 (GDV) ▪ SHK 2 (Lehrstuhl) ▪ Autor (Lehrstuhl)	▪ Vorstellung des Verfahrens zur Fehleranalyse durch den Autor (Dauer: 75 Minuten) ▪ Bestimmung der Anwendungsreleases von LV-Neu, die näher untersucht werden sollen ▪ Abstimmung weiteres Vorgehen
3	**10.06.05**	**Interview (Dauer: 120 Minuten)** Teilnehmer: ▪ interner Berater 1 (IT-Power) ▪ Autor (Lehrstuhl)	▪ Allgemeine Informationen zu IT-Power ▪ Informationen zu Softwareentwicklungsprojekte bei IT-Power ▪ Informationen zum Projekt LV-Neu
4	**14.06.05**	**Interview (Dauer: 30 Minuten)** Teilnehmer: ▪ interner Berater 1 (IT-Power) ▪ Autor (Lehrstuhl)	▪ Fehlerverfolgung im Projekt LV-Neu ▪ Informationen zur eingesetzten Problemdatenbank
5	**14.06.05**	**E-Mail** Antwort von Testmanager 1 (GDV)	▪ Fehlerverfolgung im Projekt LV-Neu
6	**05.07.05**	**Interview, Videokonferenz (Dauer: 30 Minuten)** Teilnehmer: ▪ interner Berater 1 (IT-Power), Standort A ▪ Testmanager 1 (GDV), Standort B ▪ externer Berater 1, Standort B ▪ SHK 1 (Lehrstuhl), Standort A ▪ Autor (Lehrstuhl), Standort A	▪ Vorstellung des Fehlerklassifikationsschema

Tabelle 11-5: Wichtige Ereignisse im Rahmen der Fallstudie

Nr.	Datum	Ereignis	Inhalt
7	**05.07.05**	**Interview (Dauer: 75 Minuten)** Teilnehmer: ▪ interner Berater 1 (IT-Power) ▪ SHK 1 (Lehrstuhl) ▪ Autor (Lehrstuhl)	▪ Fehlerklassifikationsschema
8	**08.07.05**	**Interview (Dauer: 90 Minuten)** Teilnehmer: ▪ interner Berater 1 (IT-Power) ▪ Testmanager 1 (GDV) ▪ SHK 1 (Lehrstuhl) ▪ Autor (Lehrstuhl)	▪ Vorstellung einer Stichprobe von klassifizierten Fehlern und Erklärung der Fehlerklassifikation anhand dieser Stichprobe ▪ Komponenten der Software LV-Neu ▪ Vorgehensmodell im Projekt LV-Neu ▪ Attribute in der Problemdatenbank
9	**19.07.05**	**E-Mail** Antwort von Testmanager 1 (GDV)	▪ Ungereimheiten in den Problem-berichten der Problemdatenbank
10	**19.07.05**	**Interview, Telefongespräch (Dauer: 30 Minuten)** Teilnehmer: ▪ Testmanager 1 (GDV) ▪ Autor (Lehrstuhl)	▪ Patch-Management ▪ Fehlerverfolgung im Projekt LV-Neu ▪ Teststufen im Projekt LV-Neu
11	**20.07.05**	**Interview (Dauer: 60 Minuten)** Teilnehmer: ▪ interner Berater 1 (IT-Power) ▪ SHK 1 (Lehrstuhl) ▪ Autor (Lehrstuhl)	▪ DV-Konzept der Anwendungsreleases 4.2 und 4.3 von LV-Neu ▪ Hintergrundinfos zur Releaseentwick-lung 4.2 und 4.3 ▪ Zeitplanung für Release 4.2 und 4.3 ▪ Zwischenergebnis der Fehlererfassung
12	**22.07.05**	**Interview (Dauer: 75 Minuten)** Teilnehmer: ▪ interner Berater 1 (IT-Power) ▪ Testmanager 1 (GDV) ▪ SHK 1 (Lehrstuhl) ▪ Autor (Lehrstuhl)	▪ Zuordnung von Rollen zu Personen, die Problemberichte angelegt haben. ▪ Diskussion einzelner Problemberichte ▪ Zeitplanung für Release 4.2 und 4.3
13	**26.07.05**	**Interview, Telefongespräch (Dauer: 45 Minuten)** Teilnehmer: ▪ Testmanager 1 (GDV) ▪ Autor (Lehrstuhl)	▪ Übergang von LV-Alt auf LV-Neu ▪ Teststufen und Testkriterien

Tabelle 11-6: Wichtige Ereignisse im Rahmen der Fallstudie (Fortsetzung 1)

Nr.	Datum	Ereignis	Inhalt
14	**03.08.05**	**Arbeitssitzung (Dauer: 180 Minuten)** Teilnehmer: ▪ Testmanager 1 (GDV) ▪ Testmanager 2 (IT-Power) ▪ Fehlermanager (IT-Power) ▪ Entwicklungsleiter (IT-Power) ▪ externer Berater ▪ SHK 1 (Lehrstuhl) ▪ Autor (Lehrstuhl)	▪ erste Ergebnisse der Fehlererfassung und –auswertung
15	**08.08.05**	**E-Mail** Antwort von Testmanager 1 (GDV)	▪ Zur Person des Testmanagers 1 (GDV)
16	**08.08.05**	**Interview, Telefongespräch (Dauer: 30 Minuten)** Teilnehmer: ▪ Testmanager 1 (GDV) ▪ Autor (Lehrstuhl)	▪ Zuordnung Testkriterien zu Teststufen ▪ Teststufen im Projekt LV-Neu ▪ Feedback zur Fehlerauswertung vom 03.08.05
17	**09.08.05**	**E-Mail** Antwort von Testmanager 1 (GDV)	▪ Feedback zur Fehlerauswertung vom 03.08.05
18	**11.08.05**	**Interview (Dauer: 270 Minuten)** Teilnehmer: ▪ interner Berater 1 (IT-Power) ▪ SHK 3 (Lehrstuhl) ▪ Autor (Lehrstuhl)	▪ Stärken und Schwächen der praktizierten Fehlerverfolgung im Projekt LV-Neu ▪ Feedback zur Fehlerauswertung vom 03.08.05
19	**22.08.05**	**Interview (Dauer: 45 Minuten)** Teilnehmer: ▪ Bereichsleiter Anwendungsentwicklung (IT-Power) ▪ Autor (Lehrstuhl)	▪ erste Ergebnisse der Fehlererfassung und –auswertung
20	**30.08.05**	**E-Mail** Antwort vom Bereichsleiter Anwendungsentwicklung (IT-Power)	▪ Allgemeine Informationen zu IT-Power ▪ Softwareentwicklungsprojekte bei IT-Power

Tabelle 11-7: Wichtige Ereignisse im Rahmen der Fallstudie (Fortsetzung 2)

Nr.	Datum	Ereignis	Inhalt
21	**02.09.05**	**Interview (Dauer: 90 Minuten)** Teilnehmer: ▪ Tester (GDV) ▪ Autor (Lehrstuhl)	▪ Zusammenarbeit IT-Power und GDV beim Test für das Projekt LV-Neu ▪ Teststufen, die durch die GDV durchgeführt werden.
22	**05.09.05**	**Interview (Dauer: 60 Minuten)** Teilnehmer: ▪ Tester (GDV) ▪ Autor (Lehrstuhl)	▪ Zuordnung von Rollen zu Personen, die Problemberichte angelegt haben. ▪ Ableitung Testkriterien aus Testfällen ▪ Teststufen, die durch die GDV durchgeführt werden.
23	**08.09.05**	**Interview (Dauer: 75 Minuten)** Teilnehmer: ▪ Testmanager 2 (IT-Power) ▪ Autor (Lehrstuhl)	▪ Rahmenbedingungen bei der Entwicklung der Releases 4.2 und 4.3 von LV-Neu ▪ Teststufen im Projekt LV-Neu ▪ Feedback zur Fehlerauswertung vom 03.08.05
24	**08.09.05**	**Interview (Dauer: 45 Minuten)** Teilnehmer: ▪ Leiter Qualitätsmanagement (IT-Power) ▪ Autor (Lehrstuhl)	▪ Rahmenbedingungen bei der Entwicklung der Releases 4.2 und 4.3 von LV-Neu ▪ Informationen zum Projekt LV-Neu ▪ Feedback zur Fehlerauswertung vom 03.08.05
25	**14.09.05**	**E-Mail** Antwort von Testmanager 2 (IT-Power)	▪ Teststufen im Projekt LV-Neu ▪ Unterschiede in den Tests für die Releases 4.2 und 4.3 von LV-Neu
26	**23.09.05**	**Messung des absoluten technischen Codeumfangs** Teilnehmer: ▪ interner Berater (IT-Power) ▪ Autor (Lehrstuhl)	▪ Messung des absoluten technischen Codeumfangs der Releases 4.2 und 4.3 von LV-Neu
27	**02.11.05**	**Interview (Dauer: 45 Minuten)** Teilnehmer: ▪ Produktdatenverantwortlicher (GDV) ▪ Autor (Lehrstuhl)	▪ Informationen zu den Produktdaten in LV-Neu ▪ Qualitätssicherung der Produktdaten und Plausibilitätsbedingungen ▪ Zur Person des Produktdatenverantwortlichen
28	**03.11.05**	**E-Mail** Antwort von Produktdatenverantwortlicher (GDV)	▪ Umfang Produktdaten und Plausibilitätsbedingungen
29	**07.11.05**	**Interview, Telefongespräch (Dauer: 15 Minuten)** Teilnehmer: ▪ Produktdatenverantwortlicher (GDV) ▪ Autor (Lehrstuhl)	▪ Informationen zu den Produktdaten in LV-Neu ▪ Umfang Produktdaten und Plausibilitätsbedingungen

Tabelle 11-8: Wichtige Ereignisse im Rahmen der Fallstudie (Fortsetzung 3)

Nr.	Datum	Ereignis	Inhalt
29	**23.11.05**	**Arbeitssitzung (Dauer: 105 Minuten)** Teilnehmer: ▪ Testmanager 1 (GDV) ▪ Testmanager 2 (IT-Power) ▪ Fehlermanager (IT-Power) ▪ interner Berater (IT-Power) ▪ Student (Lehrstuhl) ▪ Autor (Lehrstuhl)	▪ vollständige Ergebnisse der Fehlererfassung und –auswertung ▪ Auswahl Fehlermuster für Ursachenanalyse ▪ Feedback zu den Ergebnissen der Fehlerauswertung
30	**29.11.05**	**E-Mail** Antwort von Produktdatenverantwortlicher (GDV)	▪ Umfang Produktdaten und Plausibilitätsbedingungen
31	**30.11.05**	**Interview, Telefongespräch (Dauer: 5 Minuten)** Teilnehmer: ▪ Testmanager 2 (IT-Power) ▪ Autor (Lehrstuhl)	▪ Erstellung der Fachkonzepte im Projekt LV-Neu
32	**01.12.05**	**Arbeitssitzung (Dauer: 180 Minuten)** Teilnehmer: ▪ Testmanager 1 (GDV) ▪ Testmanager 2 (IT-Power) ▪ Fehlermanager (IT-Power) ▪ Produktdatenverantwortlicher (GDV) ▪ Student (Lehrstuhl) ▪ Autor (Lehrstuhl)	▪ Ursachenanalyse für ausgewählte Fehlermuster ▪ Vorschlag erster Gegenmaßnahmen
33	**07.12.05**	**E-Mail** Antwort von Testmanager 2 (IT-Power)	▪ Messung des relativen Codeumfangs der Releases 4.2 und 4.3 von LV-Neu
34	**09.12.05**	**Präsentation (Dauer: 75 Minuten)** Teilnehmer: ▪ Leiter Qualitätsmanagement (IT-Power) ▪ 12 Projektleiter verschiedener Softwareentwicklungsprojekte (IT-Power) ▪ Autor (Lehrstuhl)	▪ vollständige Ergebnisse der Fehlererfassung und –auswertung

Tabelle 11-9: Wichtige Ereignisse im Rahmen der Fallstudie (Fortsetzung 4)

Nr.	Datum	Ereignis	Inhalt
35	**23.01.06**	**Arbeitssitzung (Dauer: 180 Minuten)** Teilnehmer: ▪ Testmanager 1 (GDV) ▪ Testmanager 2 (IT-Power) ▪ Produktdatenverantwortlicher (GDV) ▪ interner Berater (IT-Power) ▪ Autor (Lehrstuhl)	▪ Präsentation von empirisch begründeten Verbesserungsvorschlägen für die identifizierten Ursachen
36	**15.03.06**	**Messung des relativen Codeumfangs** Teilnehmer: ▪ interner Berater (IT-Power) ▪ Autor (Lehrstuhl)	▪ Messung des relativen Codeumfangs der Releases 4.2 und 4.3 von LV-Neu
37	**17.03.06**	**überarbeitete Messung des relativen Codeumfangs** Teilnehmer: ▪ interner Berater (IT-Power) ▪ Autor (Lehrstuhl)	▪ Messung des relativen Codeumfangs der Releases 4.2 und 4.3 von LV-Neu
38	**18.03.06**	**Interview, Telefongeespräch (Dauer: 5 Minuten)** Teilnehmer: ▪ interner Berater (IT-Power) ▪ Entwickler (IT-Power) ▪ Autor (Lehrstuhl)	▪ Messung des relativen Codeumfangs der Releases 4.2 und 4.3 von LV-Neu
39	**20.03.06**	**Übergabe des Abschlussberichts**	

Tabelle 11-10: Wichtige Ereignisse im Rahmen der Fallstudie (Fortsetzung 5)

11.4 Anhang C.4 Profile der befragten Personen

Bereichsleiter Anwendungsentwicklung (IT-Power)

- IT-Power ist in 5 Bereiche unterteilt. Einer von diesen ist der Bereich Anwendungsentwicklung.
- Der *Bereichsleiter Anwendungsentwicklung* ist verantwortlich für alle Softwareentwicklungsprojekte, die eine Anwendungssoftware für die GDV zum Ergebnis haben.

Entwickler (IT-Power)

- Der *Entwickler* ist ein Programmierer im Projekt LV-Neu, der einen guten Überblick über den Softwarecode besitzt.

Entwicklungsleiter (IT-Power)

- Der *Entwicklungsleiter* steuert die Entwicklung neuer Releases von LV-Neu organisationsübergreifend.

Externer Berater

- Der *externe Berater* stammt von einem IT-Beratungsunternehmen außerhalb von IT-Power und soll IT-Power bei der Verbesserung der Softwareentwicklungsprozesse unterstützen.

Fehlermanager (IT-Power)

- Der *Fehlermanager* koordiniert die organisationsübergreifende Fehlerverfolgung im Projekt LV-Neu.

Interner Berater (IT-Power)

- Der *interne Berater* ist Mitarbeiter der Stabstelle Qualitätsmanagement, welche vom *Leiter Qualitätsmanagement* verantwortet wird.
- Er ist seit ca. 15 Jahre in der IT-Branche und seit Dezember 2003 bei IT-Power tätig.
- Er berät projektübergreifend die Softwareentwicklungsprojekte bei IT-Power zum Thema Qualitätsmanagement.
- Im Kontext des Projekts LV-Neu leitet er ein Projekt, das die Verbesserung des Softwareentwicklungsprozesses zum Gegenstand hat.

Leiter Qualitätsmanagement (IT-Power)

- Der *Leiter Qualitätsmanagement* verantwortet den Stab für Qualitätsmanagement bei IT-Power, der projektübergreifend für Entwicklungsprozesse verbessern soll.
- Während der Entwicklung der Releases 4.2 und 4.3 von LV-Neu war er der organisationsübergreifende Entwicklungsleiter.

Produktdatenverantwortlicher (GDV)

- Der *Produktdatenverantwortlicher* ist fachlich und technisch für die Komponente „Produktdaten und Plausibilitätsbedindungen“ verantwortlich.
- Er arbeitet in einem Team gemeinsam mit 5 Personen (4 interner und ein externer Mitarbeiter). Die Teammitglieder haben z. T. verschiedene Aufgabenschwerpunkte.
- Der *Produktdatenverantwortlicher* vertritt das Team im Releasemanagement-Team.
- Er ist seit 2,5 Jahren bei der GDV und arbeitet seit über 10 Jahren am Projekt LV-Neu mit (zuvor als externer Mitarbeiter)

SHK 1

- Studentische Hilfskraft des Lehrstuhls für Wirtschaftsinformatik, Systementwicklung der Universität zu Köln.
- Sie schreibt zugleich eine Diplomarbeit im Rahmen der Fallstudie.

SHK 2

- Studentische Hilfskraft des Lehrstuhls für Wirtschaftsinformatik, Systementwicklung der Universität zu Köln.

SHK 3

- Studentische Hilfskraft des Lehrstuhls für Wirtschaftsinformatik, Systementwicklung der Universität zu Köln.

Student

- *Student* der Wirtschaftsinformatik an der Universität zu Köln.
- Er schreibt eine Diplomarbeit im Rahmen der Fallstudie am Lehrstuhl für Wirtschaftsinformatik, Systementwicklung der Universität zu Köln.

Tester (GDV)

- Der *Tester (GDV)* ist ein Mitarbeiter vom *Testmanager 1 (GDV)*.
- Er führt Regressionstest aus, wertet der Ergebnisse aus und erstellt Testfälle auf der Grundlage von Fachkonzepten.

Testmanager 1 (GDV)

- *Testmanager 1* ist seit Anfang 2005 Gruppenleiter für die Qualitätssicherung der Bestandsführung. Ihm sind 8 interne und 2 externe Mitarbeiter unterstellt.
- Seine Aufgaben sind die Steuerung der Durchführung der automatischen Regressionstests im Rahmen der Einführung vom LV-Neu sowie der Freigabe von Geschäftsvorfällen und neuen Produkten. Für das Releasemanagement im Projekt LV-Neu ist er für den gesamten Bereich Test zuständig.
- Zum Projekt LV-Neu ist er zunächst als externer Berater zugestoßen und war für den Aufbau der Testinfrastruktur und des Testmanagements zuständig. Seit 2003 war er als interner Mitarbeiter von GDV für den Bereich Regressionstest und Releasemanagement im Projekt LV-Neu verantwortlich.
- Seit 1992 ist er als Organisationsprogrammierer und später als Teilprojektleiter in verschiedenen Entwicklungs- und Testprojekten tätig.

Testmanager 2 (IT-Power)

- *Testmanager 2* ist für das Testmanagement für LV-Neu auf Seiten von IT-Power zuständig.

11.5 Anhang C.5: Vorgehen zur Messung des Codeumfangs

Absolute und relative Umfang eines Softwarereleases in Anzahl Codeanweisungen

Im Rahmen der Fallstudie wurde sowohl der absolute als auch relative technische Umfang der untersuchten Releases in Anzahl Codeanweisungen in C gemessen.[829]

Die Einheit *technischer Umfang in Codeanweisungen in C* umfasst

- Befehle in C, die mit einem Semikolon abgeschlossen werden,
- Befehle für Kontrollstrukturen wie z. B. `if`, `for` oder `while`,
- Präprozessor-Befehle `#include`, `#define`, und `#undef`.

Die Messung des technischen Umfangs in Codeanweisungen hat gegenüber der Messung der physischen Codezeilen einen erheblichen Vorteil. Erste berücksichtigt, dass eine physische Codezeile mehrerer Codeanweisungen enthalten und ebenso eine Codeanweisung sich auf mehrere Codezeilen verteilen kann. Die Messung in Codeanweisungen ist folglich weniger abhängig vom Programmierstil.

Der absolute Umfang eines Softwarereleases ist die Summe aller Codeanweisungen, während der relative Umfang die Summe aller seit dem letzten Softwarerelease geänderten und neu hinzugefügten Codeanweisungen ist.

Vorgehensweise zur Messung des technischen Umfangs

Für die Messung des absoluten Umfangs in Codeanweisungen wurde die Freeware Source-Monitor in der Version 2.0.4.7 eingesetzt.[830] Um den relativen Umfang eines Softwarereleases zu messen, war es zunächst erforderlich, neue und geänderte Codeanweisungen zu bestimmen. Hierzu kam die ebenfalls kostenlose Software ComponentSoftware Diff, kurz CSDiff, in der Version 5.0 zum Einsatz.[831] Nachdem die neuen und geänderten Codeanweisungen bestimmt wurden, wurde diese ebenfalls mit SourceMonitor gemessen. Die konkrete Zählung der Codeanweisungen erfolgte folglich stets mit der gleichen Software.

---

[829] Vgl. zu Folgendem Kan /Metrics/ 91 f.

[830] Die Software ist kostenlos unter der folgenden URL verfügbar: http://www.campwoodsw.com/ (Stand: 17.10.2007).

[831] Die Software ist kostenlos unter der folgenden URL verfügbar: http://www.componentsoftware.com/products/csdiff/ (Stand: 17.10.2007).

11.6 Anhang C.6: Klassifizierte Implementierungsfehler

Nachfolgend warden die klassifizierten Fehler ausgelistet. Folgende Abkürzungen warden hierbei verwendet:

- Spalte QS-Aktivität: IGT = Integrationstest, ST = Systemtest, AT = Abnahmetest, UT = Update-Test.
- Spalte Komponente: AK = Anwendungskern, PDP = Produktdaten und Plausibilitätsbedingungen, VMR = versicherungsmathematischer Rechenkern.

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
1	04.11.04	04.11.04	8.2	8.2	Entwicklung	IGT	unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
2	04.11.04	11.11.04	8.2	8.2	Entwicklung	IGT	unbekannt	bekannte Anforderung	AK	Beziehung	modifiziert	Eigenentwicklung
3	05.11.04	10.11.04	8.2	8.2	Entwicklung	IGT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
4	05.11.04	10.11.04	8.2	8.2	Entwicklung	IGT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
5	09.11.04	22.11.04	8.2	8.2	Entwicklung	ST	unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
6	10.11.04	19.11.04	8.2	8.2	Entwicklung	ST	A/S	unbekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
7	10.11.04	15.11.04	8.2	8.2	Entwicklung	ST	A/S	bekannte Anforderung	AK	unbekannt	unbekannt	Eigenentwicklung
8	10.11.04	16.11.04	8.2	8.2	Entwicklung	ST	A/S	unbekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
9	10.11.04	13.12.04	8.2	8.3	Entwicklung	ST	unbekannt	unbekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
10	10.11.04	10.11.04	8.2	8.2	Entwicklung	ST	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	entfernt	externe Entwicklung
11	10.11.04	15.12.04	8.2	8.2	Entwicklung	ST	V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
12	10.11.04	21.12.04	8.2	8.2	Entwicklung	ST	V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
13	10.11.04	13.01.05	8.2	8.2	Entwicklung	ST	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
14	10.11.04	03.02.05	8.2	8.3	Entwicklung	ST	V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
15	11.11.04	12.11.04	8.2	8.2	Entwicklung	AT	A/S	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
16	11.11.04	26.11.04	8.2	8.2	Entwicklung	AT	A/S	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
17	11.11.04	18.11.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
18	11.11.04	15.11.04	8.2	8.2	Entwicklung	AT	A/S	bekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
19	12.11.04	11.02.05	8.2	8.2	Entwicklung	AT	A/S	Versionswechsel	PDP	Bedingung	modifiziert	Eigenentwicklung
20	12.11.04	24.11.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	SNT	unklar	unklar	Eigenentwicklung
21	12.11.04	15.11.04	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
22	12.11.04	01.03.05	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
23	15.11.04	16.11.04	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
24	15.11.04	03.01.05	8.2	8.3	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	entfernt	externe Entwicklung
25	15.11.04	17.11.04	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
26	15.11.04	10.03.05	8.2	8.3	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Bedingung	entfernt	externe Entwicklung
27	16.11.04	18.11.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
28	17.11.04	29.04.05	8.2	8.3	Entwicklung	AT	V/A/S	unbekannte Anforderung	AK	unklar	unklar	Eigenentwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
29	17.11.04	03.01.05	8.2	8.3	Entwicklung	AT	V/A/S	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
30	17.11.04	17.11.04	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
31	17.11.04	10.02.05	8.2	8.3	Entwicklung	AT	V/A/S	unbekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
32	18.11.04	23.11.04	8.2	8.2	Entwicklung	AT	unbekannt	Zuverlässigkeit	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
33	18.11.04	18.11.04	8.2	8.2	Entwicklung	AT	unbekannt	Zuverlässigkeit	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
34	18.11.04	18.11.04	8.2	8.2	Entwicklung	AT	unbekannt	Zuverlässigkeit	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
35	18.11.04	17.01.05	8.2	8.3	Entwicklung	AT	unbekannt	bekannte Anforderung	AK	Zuweisung/Initialisierung	eingefügt	Eigenentwicklung
36	18.11.04	10.03.05	8.2	8.3	E#ntwicklung	AT	V/A/S	Benutzbarkeit	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
37	18.11.04	22.11.04	8.2	8.2	Entwicklung	AT	V/A/S	unbekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
38	18.11.04	22.11.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
39	18.11.04	22.11.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
40	18.11.04	22.11.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
41	18.11.04	01.03.05	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
42	19.11.04	25.11.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
43	19.11.04	22.11.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
44	22.11.04	22.11.04	8.2	8.2	Entwicklung	AT	unbekannt	Integrität/Sicherheit	SNT	Algorithmus/Methode	modifiziert	Eigenentwicklung
45	22.11.04	16.12.04	8.2	8.2	Entwicklung	AT	A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
46	22.11.04	23.11.04	8.2	8.2	Entwicklung	AT	V/A/S	Benutzbarkeit	AK	Zuweisung/Initialisierung	modifiziert	Eigenentwicklung
47	23.11.04	01.12.04	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	AK	Zuweisung/Initialisierung	eingefügt	Eigenentwicklung
48	24.11.04	02.12.04	8.2	8.2	Entwicklung	AT	A/S	bekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
49	24.11.04	26.05.05	8.2	8.3	Entwicklung	AT	A/S	bekannte Anforderung	AK	Funktion/Klasse	eingefügt	Eigenentwicklung
50	24.11.04	13.04.05	8.2	8.3	Entwicklung	AT	A/S	bekannte Anforderung	PDP	Bedingung	eingefügt	externe Entwicklung
51	24.11.04	01.12.04	8.2	8.2	Entwicklung	AT	unbekannt	Zuverlässigkeit	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
52	25.11.04	25.11.04	8.2	8.3	Entwicklung	AT	unbekannt	Zuverlässigkeit	AK	Zuweisung/Initialisierung	eingefügt	Eigenentwicklung
53	25.11.04	25.11.04	8.2	8.3	Entwicklung	AT	unbekannt	Zuverlässigkeit	AK	Bedingung	eingefügt	Eigenentwicklung
54	25.11.04	25.11.04	8.2	8.3	Entwicklung	AT	unbekannt	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
55	25.11.04	25.11.04	8.2	8.3	Entwicklung	AT	unbekannt	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
56	25.11.04	25.11.04	8.2	8.3	Entwicklung	AT	unbekannt	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
57	25.11.04	07.12.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
58	25.11.04	01.12.04	8.2	8.2	Entwicklung	AT	unbekannt	Zuverlässigkeit	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
59	25.11.04	02.12.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
60	25.11.04	07.12.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
61	27.11.04	02.12.04	8.2	8.2	Entwicklung	AT	A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
62	29.11.04	01.12.04	8.2	8.2	Entwicklung	AT	unbekannt	unbekannte Anforderung	SNT	Bedingung	entfernt	Eigenentwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
63	29.11.04	14.12.05	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Bedingung	unklar	externe Entwicklung
64	29.11.04	14.12.05	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Bedingung	eingefügt	externe Entwicklung
65	29.11.04	07.12.04	8.2	8.3	Entwicklung	AT	A/S	Zuverlässigkeit	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
66	29.11.04	07.12.04	8.2	8.2	Entwicklung	AT	A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
67	30.11.04	17.01.05	8.2	8.2	Entwicklung	AT	A/S	bekannte Anforderung	AK	unklar	eingefügt	Eigenentwicklung
68	30.11.04	10.12.04	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
69	30.11.04	07.12.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
70	30.11.04	14.01.04	8.2	8.2	Entwicklung	AT	A/S	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
71	30.11.04	10.02.05	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
72	30.11.04	23.05.05	8.2	8.3	Entwicklung	AT	A/S	bekannte Anforderung	unbekannt	unbekannt	unbekannt	Eigenentwicklung
73	30.11.04	07.12.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
74	30.11.04	07.12.04	8.2	8.2	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
75	30.11.04	11.01.05	8.2	8.3	Entwicklung	AT	V/A/S	bekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
76	30.11.04	04.04.05	8.2	8.3	Entwicklung	AT	A/S	bekannte Anforderung	AK	Funktion/Klasse	eingefügt	Eigenentwicklung
77	30.11.04	07.12.04	8.2	8.2	Entwicklung	AT	V/A/S	unbekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
78	30.11.04	06.12.04	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
79	30.11.04	12.01.05	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
80	30.11.04	01.12.04	8.2	8.2	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
81	01.12.04	24.02.05	8.2	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	PDP	Bedingung	entfernt	externe Entwicklung
82	01.12.04	01.02.05	8.2	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
83	01.12.04	14.12.04	8.2	8.2	Entwicklung	UT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
84	01.12.04	01.12.04	8.2	8.2	Entwicklung	UT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
85	01.12.04	21.01.05	8.2	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
86	02.12.04	15.04.05	8.2	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
87	02.12.04	15.04.05	8.2	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	PDP	Bedingung	eingefügt	externe Entwicklung
88	02.12.04	06.12.04	8.2	8.2	Entwicklung	UT	A/S	bekannte Anforderung	SNT	Bedingung	eingefügt	Eigenentwicklung
89	02.12.04	13.04.05	8.2	8.3	Entwicklung	UT	A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	unklar	externe Entwicklung
90	03.12.04	10.01.05	8.2	8.3	Entwicklung	UT	A/S	bekannte Anforderung	AK	interne Schnittstelle	eingefügt	Eigenentwicklung
91	03.12.04	03.12.04	8.2	8.3	Entwicklung	UT	unbekannt	Zuverlässigkeit	AK	Bedingung	eingefügt	Eigenentwicklung
92	07.12.04	20.12.04	8.2	8.3	Produktion		V/A/S	Integrität/Sicherheit	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
93	07.12.04	10.03.05	8.2	8.3	Produktion		V/A/S	unbekannte Anforderung	AK	Funktion/Klasse	eingefügt	Eigenentwicklung
94	07.12.04	16.12.04	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
95	08.12.04	14.12.04	8.2	8.2	Produktion		unbekannt	Zuverlässigkeit	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
96	08.12.04	17.01.05	8.2	8.3	Produktion		unbekannt	unbekannte Anforderung	AK	Bedingung	modifiziert	Eigenentwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
97	09.12.04	09.12.04	8.2	8.2	Produktion		A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
98	09.12.04	13.12.04	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
99	09.12.04	13.12.04	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	SNT	unklar	unklar	Eigenentwicklung
100	09.12.04	14.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
101	09.12.04	09.12.04	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
102	09.12.04	09.03.05	8.2	8.3	Produktion		V/A/S	Benutzbarkeit	PDP	Bedingung	eingefügt	externe Entwicklung
103	10.12.04	20.12.04	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
104	10.12.04	31.01.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	SNT	Algorithmus/Methode	modifiziert	Eigenentwicklung
105	10.12.04	17.01.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
106	13.12.04	14.12.04	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	SNT	Algorithmus/Methode	entfernt	Eigenentwicklung
107	13.12.04	14.12.04	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	SNT	unklar	unklar	Eigenentwicklung
108	13.12.04	16.12.04	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
109	14.12.04	15.12.04	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	unklar	Eigenentwicklung
110	14.12.04	13.01.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
111	14.12.04	16.12.04	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
112	14.12.04	17.05.05	8.2	8.3	Produktion		A/S	unbekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
113	14.12.04	16.12.04	8.2	8.2	Produktion		unbekannt	Zuverlässigkeit	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
114	14.12.04	21.12.04	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
115	15.12.04	04.01.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
116	15.12.04	04.01.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
117	16.12.04	25.01.05	8.2	8.2	Produktion		V/A/S	unbekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
118	16.12.04	11.01.05	8.2	8.2	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
119	16.12.04	07.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
120	21.12.04	19.01.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	SNT	Bedingung	eingefügt	Eigenentwicklung
121	17.12.04	07.12.04	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
122	17.12.04	21.12.04	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
123	22.12.04	28.02.05	8.2	8.2	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
124	22.12.04	28.02.05	8.2	8.2	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
125	17.12.04	13.01.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
126	23.12.04	04.02.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
127	23.12.04	24.02.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	unklar	unklar	externe Entwicklung
128	23.12.04	24.02.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	unklar	unklar	externe Entwicklung
129	22.12.04	12.01.05	8.2	8.2	Produktion		V/A/S	unbekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
130	03.01.05	04.01.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
131	03.01.05	28.02.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	unklar	unklar	externe Entwicklung
132	03.01.05	03.02.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
133	03.01.05	12.01.05	8.2	8.2	Produktion		V/A/S	unbekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
134	05.01.05	10.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	unklar	unklar	externe Entwicklung
135	07.01.05	10.01.05	8.2	8.3	Produktion		A/S	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
136	07.01.05	25.01.05	8.2	8.3	Produktion		A/S	bekannte Anforderung	PDP	unklar	unklar	externe Entwicklung
137	10.01.05	11.01.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
138	11.01.05	14.01.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
139	12.01.05	13.04.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Bedingung	entfernt	externe Entwicklung
140	12.01.05	13.01.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
141	14.01.05	04.02.05	8.2	8.2	Produktion		V/A/S	unbekannte Anforderung	SNT	unklar	unklar	Eigenentwicklung
142	17.01.05	04.05.05	8.2	8.3	Produktion		A/S	unbekannte Anforderung	AK	Bedingung	modifiziert	Eigenentwicklung
143	17.01.05	21.01.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
144	17.01.05	07.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
145	19.01.05	23.03.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
146	20.01.05	09.02.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Bedingung	modifiziert	externe Entwicklung
147	20.01.05	27.01.05	8.2	8.3	Produktion		unbekannt	unbekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
148	20.01.05	26.01.05	8.2	8.3	Produktion		A/S	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
149	21.01.05	01.02.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Zuweisung/Initialisierung	eingefügt	Eigenentwicklung
150	21.01.05	03.02.05	8.2	8.2	Produktion		A/S	bekannte Anforderung	unbekannt	unbekannt	unbekannt	unbekannt
151	21.01.05	24.01.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
152	21.01.05	16.02.05	8.2	8.2	Produktion		A/S	unbekannte Anforderung	SNT	Algorithmus/Methode	eingefügt	Eigenentwicklung
153	24.01.05	07.04.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
154	25.01.05	10.02.05	8.2	8.3	Produktion		unbekannt	Zuverlässigkeit	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
155	25.01.05	14.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
156	25.01.05	21.02.05	8.2	8.3	Produktion		A/S	bekannte Anforderung	AK	Algorithmus/Methode	unklar	Eigenentwicklung
157	26.01.05	07.02.05	8.2	8.2	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
158	26.01.05	14.04.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Bedingung	modifiziert	externe Entwicklung
159	26.01.05	07.04.05	8.2	8.2	Produktion		unbekannt	Versionswechsel	VMR	irrelevant	irrelevant	Wiederverwendung
160	26.01.05	07.04.05	8.2	8.2	Produktion		unbekannt	Versionswechsel	VMR	irrelevant	irrelevant	Wiederverwendung
161	26.01.05	24.02.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Bedingung	entfernt	externe Entwicklung
162	26.01.05	14.02.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
163	27.01.05	27.01.05	8.2	8.2	Produktion		A/S	bekannte Anforderung	AK	interne Schnittstelle	modifiziert	Eigenentwicklung
164	27.01.05	10.02.05	8.2	8.2	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
165	28.01.05	03.02.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
166	28.01.05	03.02.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
167	31.01.05	13.04.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
168	31.01.05	13.04.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
169	31.01.05	02.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	Zuweisung/Initialisierung	modifiziert	Eigenentwicklung
170	31.01.05	21.02.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
171	31.01.05	13.07.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
172	31.01.05	07.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	unbekannt	unklar	unklar	unklar
173	31.01.05	03.06.05	8.2	8.3	Produktion		A/S	unbekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
174	31.01.05	04.02.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
175	01.02.05	21.04.05	8.2	8.3	Produktion		A/S	bekannte Anforderung	AK	unbekannt	unbekannt	unbekannt
176	01.02.05	16.02.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
177	01.02.05	10.02.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
178	01.02.05	09.02.05	8.2	8.3	Produktion		A/S	bekannte Anforderung	AK	Funktion/Klasse	modifiziert	Eigenentwicklung
179	01.02.05	23.02.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
180	02.02.05	07.02.05	8.2	8.3	Produktion		A/S	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
181	02.02.05	18.03.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
182	03.02.05	03.02.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	AK	Zuweisung/Initialisierung	modifiziert	Eigenentwicklung
183	03.02.05	03.02.05	8.2	8.2	Produktion		unbekannt	Zeitverhalten	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
184	03.02.05	03.02.05	8.2	8.2	Produktion		unbekannt	unbekannte Anforderung	SNT	Bedingung	modifiziert	Eigenentwicklung
185	03.02.05	14.03.05	8.2	8.3	Produktion		unbekannt	unbekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
186	03.02.05	16.02.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
187	04.02.05	07.02.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
188	08.02.05	22.02.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
189	11.02.05	23.02.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
190	16.02.05	23.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
191	07.02.05	11.03.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Bedingung	modifiziert	Eigenentwicklung
192	09.02.05	04.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	interne Schnittstelle	modifiziert	Eigenentwicklung
193	17.02.05	24.02.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
194	18.02.05	25.02.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
195	21.02.05	14.04.05	8.2	8.3	Produktion		A/S	bekannte Anforderung	AK	unbekannt	unbekannt	Eigenentwicklung
196	24.02.05	08.03.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
197	24.02.05	08.03.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
198	24.02.05	08.03.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
199	24.02.05	08.03.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
200	24.02.05	01.03.05	8.2	8.2	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
201	24.02.05	30.05.05	8.2	8.3	Produktion		A/S	bekannte Anforderung	AK	unbekannt	unbekannt	unklar
202	24.02.05	02.05.05	8.2	8.3	Produktion		V/A/S	unbekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
203	24.02.05	21.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	Algorithmus/Methode	entfernt	Eigenentwicklung
204	02.03.05	06.04.05	8.2	8.3	Produktion		V/A/S	unbekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
205	03.03.05	25.05.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	SNT	unklar	unklar	unklar
206	03.03.05	03.03.05	8.2	8.3	Produktion		V/A/S	unbekannte Anforderung	PDP	unklar	unklar	externe Entwicklung
207	03.03.05	08.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
208	04.03.05	13.05.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
209	04.03.05	09.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
210	04.03.05	08.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
211	07.03.05	18.04.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	unklar	eingefügt	Eigenentwicklung
212	07.03.05	31.03.05	8.2	8.3	Produktion		A/S	unbekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
213	07.03.05	07.04.05	8.2	8.3	Produktion		A/S	bekannte Anforderung	AK	unbekannt	unbekannt	Eigenentwicklung
214	07.03.05	08.03.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
215	07.03.05	08.03.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
216	08.03.05	16.03.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
217	08.03.05	10.03.05	8.2	8.3	Produktion		A/S	bekannte Anforderung	PDP	Bedingung	modifiziert	externe Entwicklung
218	08.03.05	21.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	Algorithmus/Methode	eingefügt	Eigenentwicklung
219	08.03.05	16.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
220	09.03.05	27.04.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	SNT	unbekannt	unbekannt	unklar
221	09.03.05	24.05.05	8.3	8.3	Entwicklung	IGT	unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
222	09.03.05	23.03.05	8.2	8.3	Produktion		unbekannt	bekannte Anforderung	SNT	Algorithmus/Methode	modifiziert	Eigenentwicklung
223	09.03.05	28.04.05	8.3	8.3	Entwicklung	IGT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
224	11.03.05	07.04.05	8.3	8.4	Entwicklung	IGT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
225	11.03.05	12.04.05	8.3	8.3	Entwicklung	IGT	unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
226	14.03.05	11.04.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	AK	unklar	unklar	unklar
227	15.03.05	28.04.05	8.3	8.3	Entwicklung	IGT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
228	15.03.05	28.04.05	8.3	8.3	Entwicklung	IGT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	entfernt	externe Entwicklung
229	17.03.05	10.05.05	8.3	8.3	Entwicklung	IGT	unbekannt	unbekannte Anforderung	SNT	Funktion/Klasse	eingefügt	Eigenentwicklung
230	22.03.05	29.03.05	8.2	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Bedingung	entfernt	externe Entwicklung
231	23.03.05	11.04.05	8.3	8.3	Entwicklung	IGT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
232	23.03.05	12.04.05	8.3	8.3	Entwicklung	IGT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
233	23.03.05	12.04.05	8.3	8.3	Entwicklung	IGT	V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
234	24.03.05	01.04.05	8.3	8.3	Entwicklung	IGT	V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
235	24.03.05	24.03.05	8.3	8.3	Entwicklung	IGT	unbekannt	bekannte Anforderung	AK	Zuweisung/Initialisierung	modifiziert	Eigenentwicklung
236	24.03.05	05.04.05	8.3	8.3	Entwicklung	IGT	V/A/S	bekannte anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
237	29.03.05	11.04.05	8.3	8.3	Entwicklung	ST	unbekannt	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
238	31.03.05	02.05.05	8.3	8.3	Entwicklung	ST	A/S	bekannte Anforderung	AK	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
239	31.03.05	01.04.05	8.3	8.3	Entwicklung	ST	unbekannt	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
240	31.03.05	06.04.05	8.3	8.3	Entwicklung	ST	unbekannt	bekannte Anforderung	AK	Bedingung	entfernt	Eigenentwicklung
241	31.03.05	01.04.05	8.3	8.3	Entwicklung	ST	unbekannt	bekannte Anforderung	AK	Zuweisung/Initialisierung	eingefügt	Eigenentwicklung
242	31.03.05	01.04.05	8.3	8.3	Entwicklung	ST	unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
243	31.03.05	01.04.05	8.3	8.3	Entwicklung	ST	A/S	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
244	04.04.05	06.04.05	8.3	8.3	Entwicklung	ST	A/S	bekannte Anforderung	AK	interne Schnittstelle	entfernt	Eigenentwicklung
245	04.04.05	01.08.05	8.3	8.3	Entwicklung	ST	A/S	Integrität/Sicherheit	AK	Bedingung	eingefügt	Eigenentwicklung
246	05.04.05	08.04.05	8.3	8.3	Entwicklung	ST	unbekannt	bekannte Anforderung	AK	unklar	modifiziert	Eigenentwicklung
247	05.04.05	09.08.05	8.3	8.4	Entwicklung	ST	unbekannt	unbekannte Anforderung	AK	Algorithmus/Methode	entfernt	Eigenentwicklung
248	05.04.05	06.04.05	8.3	8.3	Entwicklung	ST	unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
249	06.04.05	06.04.05	8.3	8.3	Entwicklung	ST	A/S	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
250	06.04.05	13.04.05	8.3	8.3	Entwicklung	ST	A/S	bekannte Anforderung	AK	Bedingung	modifiziert	Eigenentwicklung
251	07.04.05	30.05.05	8.3	8.3	Entwicklung	ST	unbekannt	unbekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
252	07.04.05	13.05.05	8.3	8.3	Entwicklung	ST	unbekannt	unbekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
253	11.04.05	13.04.05	8.3	8.3	Entwicklung	AT	unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
254	11.04.05	12.04.05	8.3	8.3	Entwicklung	AT	V/A/S	unbekannte Anforderung	PDP	unklar	unklar	externe Entwicklung
255	11.04.05	13.04.05	8.3	8.3	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Bedingung	modifiziert	externe Entwicklung
256	11.04.05	13.04.05	8.3	8.3	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Bedingung	modifiziert	externe Entwicklung
257	11.04.05	14.04.05	8.3	8.3	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	unklar	unklar	externe Entwicklung
258	11.04.05	13.04.05	8.3	8.3	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Bedingung	modifiziert	externe Entwicklung
259	12.04.05	04.05.05	8.2	8.3	Produktion		V/A/S	unbekannte Anforderung	AK	Funktion/Klasse	modifiziert	Eigenentwicklung
260	12.04.05	25.04.05	8.3	8.3	Entwicklung	AT	V/A/S	bekannte Anforderung	PDP	Bedingung	eingefügt	externe Entwicklung
261	12.04.05	15.04.05	8.3	8.3	Entwicklung	AT	A/S	unbekannte Anforderung	AK	Funktion/Klasse	modifiziert	Eigenentwicklung
262	12.04.05	14.04.05	8.3	8.3	Entwicklung	AT	unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
263	12.04.05	20.09.05	8.3	8.4	Entwicklung	AT	V/A/S	bekannte Anforderung	AK	Algorithmus/Methode	unbekannt	Eigenentwicklung
264	14.04.05	18.04.05	8.3	8.3	Entwicklung	AT	unbekannt	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
265	15.04.05	18.04.05	8.3	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	AK	interne Schnittstelle	modifiziert	Eigenentwicklung
266	15.04.05	29.04.05	8.3	8.4	Entwicklung	UT	V/A/S	unbekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
267	15.04.05	10.05.05	8.3	8.3	Entwicklung	UT	A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	unklar	externe Entwicklung
268	15.04.05	19.04.05	8.3	8.3	Entwicklung	UT	A/S	unbekannte Anforderung	AK	Funktion/Klasse	modifiziert	Eigenentwicklung
269	18.04.05	20.07.05	8.3	8.4	Entwicklung	UT	unbekannt	unbekannte Anforderung	AK	interne Schnittstelle	modifiziert	Eigenentwicklung
270	18.04.05	20.07.05	8.3	8.4	Entwicklung	UT	unbekannt	unbekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
271	18.04.05	16.09.05	8.3	8.4	Entwicklung	UT	V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
272	18.04.05	19.04.05	8.3	8.3	Entwicklung	UT	unbekannt	unbekannte Anforderung	AK	Zuweisung/Initialisierung	modifiziert	Eigenentwicklung
273	18.04.05	27.04.05	8.3	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
274	18.04.05	19.04.05	8.3	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
275	18.04.05	19.04.05	8.3	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	AK	Zuweisung/Initialisierung	modifiziert	Eigenentwicklung
276	18.04.05	02.05.05	8.3	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
277	18.04.05	20.06.05	8.3	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
278	19.04.05	29.04.05	8.3	8.3	Entwicklung	UT	A/S	bekannte Anforderung	AK	Zuweisung/Initialisierung	eingefügt	Eigenentwicklung
279	19.04.05	20.04.05	8.3	8.4	Entwicklung	UT	A/S	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
280	19.04.05	21.04.05	8.3	8.4	Entwicklung	UT	A/S	bekannte Anforderung	AK	Zuweisung/Initialisierung	eingefügt	Eigenentwicklung
281	24.04.05	02.05.05	8.3	8.3	Entwicklung	UT	V/A/S	Integrität/Sicherheit	AK	Zuweisung/Initialisierung	eingefügt	Eigenentwicklung
282	20.04.05	03.05.05	8.2	8.3	Produktion		unbekannt	unbekannte Anforderung	AK	Zuweisung/Initialisierung	modifiziert	Eigenentwicklung
283	21.04.05	24.08.05	8.3	8.4	Entwicklung	UT	V/A/S	bekannte Anforderung	AK	Zuweisung/Initialisierung	modifiziert	Eigenentwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
284	21.04.05	31.08.05	8.3	8.4	Entwicklung	UT	V/A/S	bekannte Anforderung	AK	Zuweisung/Initialisierung	modifiziert	Eigenentwicklung
285	22.04.05	28.06.05	8.3	8.4	Entwicklung	UT	V/A/S	bekannte Anforderung	PDP	Bedingung	entfernt	externe Entwicklung
286	25.04.05	28.04.05	8.3	8.3	Entwicklung	UT	A/S	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
287	26.04.05	27.04.05	8.3	8.3	Entwicklung	UT	A/S	Versionswechsel	unbekannt	unbekannt	unbekannt	Eigenentwicklung
288	26.04.05	02.05.05	8.3	8.4	Entwicklung	UT	unbekannt	bekannte Anforderung	SNT	Funktion/Klasse	modifiziert	Eigenentwicklung
289	26.04.05	26.04.05	8.3	8.3	Entwicklung	UT	unbekannt	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
290	27.04.05	12.05.05	8.3	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	AK	interne Schnittstelle	modifiziert	Eigenentwicklung
291	27.04.05	29.04.05	8.3	8.4	Entwicklung	UT	A/S	bekannte Anforderung	AK	Funktion/Klasse	modifiziert	Eigenentwicklung
292	27.04.05	17.05.05	8.3	8.3	Entwicklung	UT	unbekannt	bekannte Anforderung	SNT	Bedingung	modifiziert	Eigenentwicklung
293	27.04.05	17.06.05	8.3	8.4	Entwicklung	UT	unbekannt	bekannte Anforderung	SNT	Funktion/Klasse	eingefügt	Eigenentwicklung
294	28.04.05	08.06.05	8.3	8.4	Entwicklung	UT	unbekannt	bekannte Anforderung	SNT	Algorithmus/Methode	modifiziert	Eigenentwicklung
295	28.04.05	27.07.05	8.3	8.4	Entwicklung	UT	unbekannt	bekannte Anforderung	SNT	Algorithmus/Methode	modifiziert	Eigenentwicklung
296	28.04.05	06.09.05	8.3	8.4	Entwicklung	UT	V/A/S	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
297	28.04.05	13.06.05	8.3	8.4	Entwicklung	UT	V/A/S	bekannte Anforderung	AK	unbekannt	unbekannt	Eigenentwicklung
298	28.04.05	28.04.05	8.3	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	unbekannt	unbekannt	unbekannt	Eigenentwicklung
299	29.04.05	29.04.05	8.3	8.3	Entwicklung	UT	V/A/S	bekannte Anforderung	unbekannt	unbekannt	unbekannt	Eigenentwicklung
300	29.04.05	07.07.05	8.3	8.4	Entwicklung	UT	V/A/S	bekannte Anforderung	PDP	Bedingung	eingefügt	externe Entwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
301	03.05.05	07.06.05	8.3	8.4	Produktion		V/A/S	unbekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
302	04.05.05	13.06.05	8.3	8.4	Produktion		unbekannt	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
303	04.05.05	10.05.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Bedingung	unklar	externe Entwicklung
304	04.05.05	30.05.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
305	04.05.05	01.06.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
306	10.05.05	20.06.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Bedingung	eingefügt	externe Entwicklung
307	12.05.05	23.05.05	8.3	8.3	Produktion		unbekannt	bekannte Anforderung	SNT	unklar	unklar	Eigenentwicklung
308	12.05.05	12.05.05	8.3	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
309	13.05.05	18.05.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
310	13.05.05	17.05.05	8.3	8.4	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
311	13.05.05	03.06.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	SNT	Bedingung	modifiziert	Eigenentwicklung
312	18.05.05	18.05.05	8.3	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	entfernt	Eigenentwicklung
313	18.05.05	18.05.05	8.3	8.3	Produktion		unbekannt	bekannte Anforderung	AK	Algorithmus/Methode	entfernt	Eigenentwicklung
314	18.05.05	29.06.05	8.3	8.3	Produktion		unbekannt	unbekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
315	18.05.05	19.05.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
316	19.05.05	25.08.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
317	20.05.05	09.06.05	8.3	8.4	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
318	23.05.05	09.06.05	8.3	8.4	Produktion		A/S	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
319	24.05.05	02.08.05	8.3	8.4	Produktion		unbekannt	unbekannte Anforderung	AK	Zuweisung/Initialisierung	eingefügt	Eigenentwicklung
320	30.05.05	09.08.05	8.3	8.4	Produktion		unbekannt	unbekannte Anforderung	AK	Funktion/Klasse	eingefügt	Eigenentwicklung
321	30.05.05	09.08.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
322	31.05.05	01.06.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	AK	unklar	unklar	Eigenentwicklung
323	31.05.05	01.06.05	8.3	8.3	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
324	01.06.05	27.06.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	entfernt	externe Entwicklung
325	01.06.05	06.06.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
326	02.06.05	06.06.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
327	02.06.05	23.08.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
328	02.06.05	06.06.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
329	03.06.05	03.08.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	AK	unklar	eingefügt	Eigenentwicklung
330	07.06.05	13.07.05	8.3	8.3	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
331	07.06.05	13.07.05	8.3	8.3	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
332	07.06.05	16.08.05	8.3	8.4	Produktion		unbekannt	bekannte Anforderung	AK	Zuweisung/Initialisierung	modifiziert	Eigenentwicklung
333	08.06.05	27.06.05	8.3	8.3	Produktion		V/A/S	unbekannte Anforderung	AK	Funktion/Klasse	modifiziert	Eigenentwicklung
334	09.06.05	23.06.05	8.3	8.3	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
335	09.06.05	23.06.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
336	28.06.05	28.06.05	8.3	8.3	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
337	28.06.05	30.06.05	8.3	8.4	Produktion		V/A/S	unbekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
338	28.06.05	13.07.05	8.3	8.3	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
339	29.06.05	12.07.05	8.3	8.4	Produktion		unbekannt	unbekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
340	04.07.05	19.07.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
341	04.07.05	06.07.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
342	04.07.05	06.07.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
343	05.07.05	05.07.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
344	05.07.05	07.07.05	8.3	8.4	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
345	06.07.05	01.08.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	PDP	unklar	unklar	externe Entwicklung
346	06.07.05	13.07.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	PDP	Bedingung	modifiziert	externe Entwicklung
347	06.07.05	20.07.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
348	08.07.05	15.07.05	8.3	8.3	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
349	11.07.05	13.07.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	AK	Algorithmus/Methode	modifiziert	Eigenentwicklung
350	11.07.05	11.07.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
351	11.07.05	20.07.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung

Nummer	Fehler aufgetreten	Fehler behoben	Version gefunden	Version behoben	Fehler entdeckt während	QS-Aktivität	QS-Kriterium	Auswirkung	Komponente	Gegenstand der Änderung	Art der Änderung	Ursprung
352	12.07.05	03.08.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
353	18.07.05	29.07.05	8.3	8.4	Produktion		A/S	unbekannte Anforderung	PDP	Bedingung	eingefügt	externe Entwicklung
354	18.07.05	01.09.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	AK	interne Schnittstelle	eingefügt	Eigenentwicklung
355	19.07.05	09.08.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
356	21.07.05	02.08.05	8.3	8.4	Produktion		unbekannt	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
357	25.07.05	05.09.05	8.3	8.4	Produktion		V/A/S	Zuverlässigkeit	AK	unbekannt	unbekannt	Eigenentwicklung
358	27.07.05	20.09.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	PDP	Zuweisung/Initialisierung	modifiziert	externe Entwicklung
359	03.08.05	09.08.05	8.3	8.3	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
360	05.08.05	10.08.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
361	10.08.05	11.08.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
362	10.08.05	20.09.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
363	10.08.05	20.09.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
364	15.08.05	01.09.05	8.3	8.4	Produktion		A/S	bekannte Anforderung	AK	Bedingung	modifiziert	Eigenentwicklung
365	19.08.05	22.08.05	8.3	8.4	Produktion		unbekannt	bekannte Anforderung	VMR	irrelevant	irrelevant	Wiederverwendung
366	24.08.05	06.09.05	8.3	8.4	Produktion		V/A/S	unbekannte Anforderung	AK	Bedingung	eingefügt	Eigenentwicklung
367	25.08.05	31.08.05	8.3	8.4	Produktion		V/A/S	unbekannte Anforderung	PDP	Zuweisung/Initialisierung	eingefügt	externe Entwicklung
368	29.08.05	30.08.05	8.3	8.4	Produktion		V/A/S	bekannte Anforderung	unbekannt	unklar	unklar	unklar

11.7 Anhang C.7: Detaillierte Auswertungen für Release 4.2

Gegenstand der Änderung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Zuweisung/Initialisierung	35%	9%	26%	26%
Entwicklerdokumentation	0%	9%	-9%	9%
Timing/Serialisierung	0%	9%	-9%	9%
Beziehung	0%	9%	-9%	9%
Algorithmus	17%	9%	8%	8%
interne Schnittstelle	1%	9%	-8%	8%
Bedingung	16%	9%	7%	7%
Funktion	2%	9%	-7%	7%
unbekannt	4%	9%	-6%	6%
irrelevant	14%	9%	5%	5%
unklar	11%	9%	2%	2%

Tabelle 11-11: Eindimensionale Auswertungen für das Attribut „Gegenstand der Änderung“

Art der Änderung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
modifiziert	36%	17%	20%	20%
eingefügt	30%	17%	14%	14%
unbekannt	4%	17%	-13%	13%
entfernt	4%	17%	-12%	12%
unklar	12%	17%	-5%	5%
irrelevant	14%	17%	-3%	3%

Tabelle 11-12: Eindimensionale Auswertungen für das Attribut „Art der Änderung“

QS-Kriterium	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Abdeckung / Sequenz	18%	33%	-16%	16%
unbekannt	46%	33%	12%	12%
Variation / Abdeckung / Sequenz	36%	33%	3%	3%

Tabelle 11-13: Eindimensionale Auswertungen für das Attribut „QS-Kriterium“

Phase	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Entwicklung	40%	50%	-10%	10%
Produktion	60%	50%	10%	10%

Tabelle 11-14: Eindimensionale Auswertungen für das Attribut „Phase“

Komponente	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Produktdaten und Plausibilitätsbedingungen	40%	20%	20%	20%
unbekannt	1%	20%	-19%	19%
Anwendungskern	38%	20%	18%	18%
Schnittstellen zu Randsystemen	7%	20%	-13%	13%
versicherungsmathematischer Rechenkern	14%	20%	-6%	6%

Tabelle 11-15: Eindimensionale Auswertungen für das Attribut „Komponente“

QS-Aktivität	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Abnahmetest	73%	25%	48%	48%
Integrationstest	4%	25%	-21%	21%
Systemtest	11%	25%	-14%	14%
Update-Test	12%	25%	-13%	13%

Tabelle 11-16: Eindimensionale Auswertungen für das Attribut „QS-Aktivität“

Auswirkung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
bekannte Softwareanforderung	79%	8%	71%	71%
Installierbarkeit	0%	8%	-8%	8%
Analysierbarkeit von Fehlverhalten	0%	8%	-8%	8%
Standards für Softwareprodukte	0%	8%	-8%	8%
Benutzerdokumentation in der Software	0%	8%	-8%	8%
Wartbarkeit	0%	8%	-8%	8%
Barrierefreiheit	0%	8%	-8%	8%
Zeitverhalten	0%	8%	-7%	7%
Integrität / Sicherheit	1%	8%	-7%	7%
Versionswechsel	1%	8%	-6%	6%
Benutzbarkeit	1%	8%	-6%	6%
unbekannte Softwareanforderung	12%	8%	4%	4%
Zuverlässigkeit	5%	8%	-2%	2%

Tabelle 11-17: Eindimensionale Auswertungen für das Attribut „Auswirkung“

Gegenstand der Änderung	Art der Änderung	Gegenstand der Änderung	Art der Änderung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
irrelevant	irrelevant	14%	14%	14%	2%	12%	12%
Zuweisung/ Initialisierung	modifiziert	35%	36%	23%	13%	10%	10%
unklar	unklar	11%	12%	10%	1%	9%	9%
irrelevant	modifiziert	14%	36%	0%	5%	-5%	5%
Bedingung	eingefügt	16%	30%	10%	5%	5%	5%
Zuweisung/ Initialisierung	irrelevant	35%	14%	0%	5%	-5%	5%
irrelevant	eingefügt	14%	30%	0%	4%	-4%	4%
unklar	modifiziert	11%	36%	0%	4%	-4%	4%
Zuweisung/ Initialisierung	unklar	35%	12%	0%	4%	-4%	4%
unbekannt	unbekannt	4%	4%	4%	0%	3%	3%
Bedingung	modifiziert	16%	36%	4%	6%	-2%	2%
unklar	eingefügt	11%	30%	1%	3%	-2%	2%
Algorithmus	irrelevant	17%	14%	0%	2%	-2%	2%
Bedingung	irrelevant	16%	14%	0%	2%	-2%	2%
Algorithmus	eingefügt	17%	30%	7%	5%	2%	2%
Bedingung	entfernt	16%	4%	3%	1%	2%	2%
Algorithmus	modifiziert	17%	36%	8%	6%	2%	2%
irrelevant	unklar	14%	12%	0%	2%	-2%	2%
…	…	…	….	…	…	…	…

Tabelle 11-18: Zweidimensionale Auswertungen: Gegenstand der Änderung x Art der Änderung

Gegenstand der Änderung	QS-Kriterium	Gegenstand der Änderung	QS-Kriterium	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Zuweisung/Initialisierung	unbekannt	35%	46%	20%	16%	5%	5%
Zuweisung/Initialisierung	Abdeckung / Sequenz	35%	18%	2%	6%	-4%	4%
irrelevant	unbekannt	14%	46%	10%	6%	4%	4%
Bedingung	unbekannt	16%	46%	4%	8%	-3%	3%
unbekannt	Abdeckung / Sequenz	4%	18%	3%	1%	2%	2%
Bedingung	Variation / Abdeckung / Sequenz	16%	36%	8%	6%	2%	2%
unklar	Variation / Abdeckung / Sequenz	11%	36%	6%	4%	2%	2%
unklar	unbekannt	11%	46%	3%	5%	-2%	2%
irrelevant	Abdeckung / Sequenz	14%	18%	0%	2%	-2%	2%
irrelevant	Variation / Abdeckung / Sequenz	14%	36%	3%	5%	-2%	2%
Algorithmus	Abdeckung / Sequenz	17%	18%	5%	3%	2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-19: Zweidimensionale Auswertungen: Gegenstand der Änderung x QS-Kriterium

Gegenstand der Änderung	Phase	Gegenstand der Änderung	Phase	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Zuweisung/Initialisierung	Entwicklung	35%	40%	18%	14%	4%	4%
Zuweisung/Initialisierung	Produktion	35%	60%	17%	21%	-4%	4%
irrelevant	Produktion	14%	60%	10%	8%	2%	2%
irrelevant	Entwicklung	14%	40%	4%	6%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-20: Zweidimensionale Auswertungen: Gegenstand der Änderung x Phase

Gegenstand der Änderung	QS-Aktivität	Gegenstand der Änderung	QS-Aktivität	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Bedingung	Update-Test	18%	12%	5%	2%	3%	3%
irrelevant	Systemtest	10%	11%	4%	1%	3%	3%
Zuweisung/Initialisierung	Systemtest	44%	11%	2%	5%	-3%	3%
Zuweisung/Initialisierung	Abnahmetest	44%	73%	34%	32%	2%	2%
Bedingung	Systemtest	18%	11%	0%	2%	-2%	2%
irrelevant	Abnahmetest	10%	73%	5%	7%	-2%	2%
Algorithmus	Update-Test	13%	12%	0%	2%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-21: Zweidimensionale Auswertungen: Gegenstand der Änderung x QS-Aktivität

Gegenstand der Änderung	Komponente	Gegenstand der Änderung	Komponente	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Zuweisung/Initialisierung	Produktdaten und Plausibilitätsbedingungen	35%	40%	31%	14%	17%	17%
irrelevant	versicherungsmathematischer Rechenkern	14%	14%	14%	2%	12%	12%
Zuweisung/Initialisierung	Anwendungskern	35%	38%	4%	13%	-10%	10%
Algorithmus	Anwendungskern	17%	38%	15%	6%	8%	8%
Algorithmus	Produktdaten und Plausibilitätsbedingungen	17%	40%	0%	7%	-7%	7%
irrelevant	Produktdaten und Plausibilitätsbedingungen	14%	40%	0%	6%	-6%	6%
irrelevant	Anwendungskern	14%	38%	0%	5%	-5%	5%
Zuweisung/Initialisierung	versicherungsmathematischer Rechenkern	35%	14%	0%	5%	-5%	5%
Algorithmus	versicherungsmathematischer Rechenkern	17%	14%	0%	2%	-2%	2%
Zuweisung/Initialisierung	Schnittstellen zu Randsystemen	35%	7%	0%	2%	-2%	2%
Bedingung	versicherungsmathematischer Rechenkern	16%	14%	0%	2%	-2%	2%
Bedingung	Anwendungskern	16%	38%	8%	6%	2%	2%
unklar	Produktdaten und Plausibilitätsbedingungen	11%	40%	3%	4%	-2%	2%
unklar	Schnittstellen zu Randsystemen	11%	7%	2%	1%	2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-22: Zweidimensionale Auswertungen: Gegenstand der Änderung x Komponente

Gegenstand der Änderung	Auswirkung	Gegenstand der Änderung	Auswirkung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Zuweisung/Initialisierung	unbekannte Softwareanforderung	35%	12%	1%	4%	-3%	3%
Bedingung	bekannte Softwareanforderung	16%	79%	10%	13%	-3%	3%
Bedingung	unbekannte Softwareanforderung	16%	12%	4%	2%	2%	2%
Algorithmus	bekannte Softwareanforderung	17%	79%	11%	13%	-2%	2%
Zuweisung/Initialisierung	bekannte Softwareanforderung	35%	79%	29%	27%	2%	2%
irrelevant	bekannte Softwareanforderung	14%	79%	13%	11%	2%	2%
Zuweisung/Initialisierung	Zuverlässigkeit	35%	5%	4%	2%	2%	2%
irrelevant	unbekannte Softwareanforderung	14%	12%	0%	2%	-2%	2%
Algorithmus	unbekannte Softwareanforderung	17%	12%	4%	2%	2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-23: Zweidimensionale Auswertungen: Gegenstand der Änderung x Auswirkung

Art der Änderung	QS-Kriterium	Art der Änderung	QS-Kriterium	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
irrelevant	unbekannt	14%	46%	10%	6%	4%	4%
unbekannt	Abdeckung / Sequenz	4%	18%	3%	1%	2%	2%
modifiziert	unbekannt	36%	46%	19%	17%	2%	2%
eingefügt	unbekannt	30%	46%	12%	14%	-2%	2%
unklar	unbekannt	12%	46%	3%	5%	-2%	2%
unklar	Variation / Abdeckung / Sequenz	12%	36%	6%	4%	2%	2%
irrelevant	Abdeckung / Sequenz	14%	18%	0%	2%	-2%	2%
irrelevant	Variation / Abdeckung / Sequenz	14%	36%	3%	5%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-24: Zweidimensionale Auswertungen: Art der Änderung x QS-Kriterium

Art der Änderung	Phase	Art der Änderung	Phase	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
eingefügt	Entwicklung	30%	40%	16%	12%	4%	4%
eingefügt	Produktion	30%	60%	14%	18%	-4%	4%
irrelevant	Produktion	14%	60%	10%	8%	2%	2%
irrelevant	Entwicklung	14%	40%	4%	6%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-25: Zweidimensionale Auswertungen: Art der Änderung x Phase

Art der Änderung	Komponente	Art der Änderung	Komponente	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
irrelevant	versicherungsmathematischer Rechenkern	14%	14%	14%	2%	12%	12%
modifiziert	Produktdaten und Plausibilitätsbedingungen	36%	40%	23%	15%	8%	8%
eingefügt	Anwendungskern	30%	38%	18%	12%	7%	7%
irrelevant	Produktdaten und Plausibilitätsbedingungen	14%	40%	0%	6%	-6%	6%
irrelevant	Anwendungskern	14%	38%	0%	5%	-5%	5%
modifiziert	versicherungsmathematischer Rechenkern	36%	14%	0%	5%	-5%	5%
eingefügt	versicherungsmathematischer Rechenkern	30%	14%	0%	4%	-4%	4%
modifiziert	Anwendungskern	36%	38%	12%	14%	-2%	2%
unklar	versicherungsmathematischer Rechenkern	12%	14%	0%	2%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-26: Zweidimensionale Auswertungen: Art der Änderung x Komponente

Art der Änderung	QS-Aktivität	Art der Änderung	QS-Aktivität	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
modifiziert	Systemtest	33%	11%	0%	4%	-4%	4%
irrelevant	Systemtest	10%	11%	4%	1%	3%	3%
modifiziert	Abnahmetest	33%	73%	26%	24%	2%	2%
modifiziert	Integrationstest	33%	4%	3%	1%	2%	2%
eingefügt	Update-Test	40%	12%	7%	5%	2%	2%
irrelevant	Abnahmetest	10%	73%	5%	7%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-27: Zweidimensionale Auswertungen: Art der Änderung x QS-Aktivität

Art der Änderung	Auswirkung	Art der Änderung	Auswirkung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
eingefügt	bekannte Softwareanforderung	30%	79%	21%	24%	-3%	3%
irrelevant	bekannte Softwareanforderung	14%	79%	13%	11%	2%	2%
eingefügt	Zuverlässigkeit	30%	5%	4%	2%	2%	2%
eingefügt	unbekannte Softwareanforderung	30%	12%	5%	4%	2%	2%
irrelevant	unbekannte Softwareanforderung	14%	12%	0%	2%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-28: Zweidimensionale Auswertungen: Art der Änderung x Auswirkung

QS-Kriterium	Komponente	QS-Kriterium	Komponente	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Abdeckung / Sequenz	Anwendungskern	18%	38%	12%	7%	5%	5%
unbekannt	Anwendungskern	46%	38%	12%	17%	-5%	5%
unbekannt	versicherungsmathematischer Rechenkern	46%	14%	10%	6%	4%	4%
Variation / Abdeckung / Sequenz	Produktdaten und Plausibilitäts-bedingungen	36%	40%	18%	15%	4%	4%
Abdeckung / Sequenz	Produktdaten und Plausibilitäts-bedingungen	18%	40%	4%	7%	-4%	4%
Abdeckung / Sequenz	versicherungsmathematischer Rechenkern	18%	14%	0%	2%	-2%	2%
Variation / Abdeckung / Sequenz	versicherungsmathematischer Rechenkern	36%	14%	3%	5%	-2%	2%
unbekannt	Schnittstellen zu Randsystemen	46%	7%	5%	3%	2%	2%
Variation / Abdeckung / Sequenz	Schnittstellen zu Randsystemen	36%	7%	1%	2%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-29: Zweidimensionale Auswertungen: QS-Kriterium x Komponente

QS-Kriterium	Auswirkung	QS-Kriterium	Auswirkung	Beobachtet %	Erwartet (%)	Differenz	Relevanz
unbekannt	Zuverlässigkeit	46%	5%	5%	2%	2%	2%
unbekannt	unbekannte Software-anforderung	46%	12%	3%	5%	-2%	2%
Variation / Abdeckung / Sequenz	Zuverlässigkeit	36%	5%	0%	2%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-30: Zweidimensionale Auswertungen: QS-Kriterium x Auswirkung

QS-Kriterium	QS-Aktivität	QS-Kriterium	QS-Aktivität	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
unbekannt	Update-Test	41%	12%	1%	5%	-4%	4%
Variation / Abdeckung / Sequenz	Update-Test	36%	12%	8%	4%	3%	3%
unbekannt	Integrationstest	41%	4%	4%	2%	3%	3%
unbekannt	Abnahmetest	41%	73%	32%	29%	2%	2%
Variation / Abdeckung / Sequenz	Abnahmetest	36%	73%	24%	26%	-2%	2%
Variation / Abdeckung / Sequenz	Integrationstest	36%	4%	0%	2%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-31: Zweidimensionale Auswertungen: QS-Kriterium x QS-Aktivität

Komponente	Phase	Komponente	Phase	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Schnittstellen zu Rand-systemen	Produktion	7%	60%	0%	4%	-4%	4%
Produktdaten und Plausibilitätsbedingungen	Entwicklung	40%	40%	19%	16%	3%	3%
Produktdaten und Plausibilitätsbedingungen	Produktion	40%	60%	21%	24%	-3%	3%
Schnittstellen zu Rand-systemen	Entwicklung	7%	40%	0%	3%	-2%	2%
versicherungsmathematischer Rechenkern	Produktion	14%	60%	10%	8%	2%	2%
versicherungsmathematischer Rechenkern	Entwicklung	14%	40%	4%	6%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-32: Zweidimensionale Auswertungen: Komponente x Phase

Komponente	Auswirkung	Komponente	Auswirkung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Anwendungskern	unbekannte Software-anforderung	38%	12%	9%	5%	5%	5%
Anwendungskern	bekannte Software-anforderung	38%	79%	25%	30%	-5%	5%
Produktdaten und Plausibilitätsbedingungen	unbekannte Software-anforderung	40%	12%	1%	5%	-4%	4%
Produktdaten und Plausibilitätsbedingungen	bekannte Software-anforderung	40%	79%	35%	31%	3%	3%
versicherungsmathematischer Rechenkern	bekannte Software-anforderung	14%	79%	13%	11%	2%	2%
versicherungsmathematischer Rechenkern	unbekannte Software-anforderung	14%	12%	0%	2%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-33: Zweidimensionale Auswertungen: Komponente x Auswirkung

Komponente	QS-Aktivität	Komponente	QS-Aktivität	Beobachtet %	Erwartet (%)	Differenz	Relevanz
versicherungsmathematischer Rechenkern	Systemtest	10%	11%	4%	1%	3%	3%
Produktdaten und Plausibilitätsbedingungen	Systemtest	47%	11%	2%	5%	-3%	3%
Produktdaten und Plausibilitätsbedingungen	Update-Test	47%	12%	8%	6%	2%	2%
versicherungsmathematischer Rechenkern	Abnahmetest	10%	73%	5%	7%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-34: Zweidimensionale Auswertungen: Komponente x QS-Aktivität

Phase	Auswirkung	Phase	Auswirkung	Beobachtet (%)	Erwartet (%)	Differenz	Relevanz
Entwicklung	Zuverlässigkeit	40%	5%	4%	2%	2%	2%
Produktion	Zuverlässigkeit	60%	5%	1%	3%	-2%	2%
...	...	...	....	...	...	...	...

Tabelle 11-35: Zweidimensionale Auswertungen: Phase x Auswirkung

Literaturverzeichnis

Adelson, Soloway /Domain experience/

Beth Adelson, Elliot Soloway: The role of domain experience in software design. In: IEEE Transactions on Software Engineering. Nr. 11, Jg. 11, 1985, S. 1351-1360.

Alavi, Leidner /Knowledge management/

Maryam Alavi, Dorothy E. Leidner: Review: Knowledge management and knowledge management systems. Conceptual foundations and research issues. In: MIS Quarterly. In: 1, Jg. 25, 2001, S. 107-136.

Albert /Traktat/

Hans Albert: Traktat über kritische Vernunft. Tübingen 1991.

Albert /Wissenschaftstheorie/

Hans Albert: Wissenschaftstheorie. In: Erwin Grochla, Waldemar Wittmann (Hrsg.): Handwörterbuch der Betriebswirtschaft. 4 Aufl., Stuttgart 1976, S. 4674-4692

Al-Rawas, Easterbrook /Communication problems/

Amer Al-Rawas, Steve Easterbrook: Communication problems in requirements engineering: a field study. In: o. Hrsg.: Proceedings of the First Westminster Conference on Professional Awareness in Software Engineering, 1-2 February 1996, London, England. London 1996, S. 47-60.

Al-Rawas, Easterbrook /Communications problems/

Amer Al-Rawas, Steve Easterbrook: Communication problems in requirements engineering. A field study. In: Colin Myers, Tracy Hall, Dave Pitt (Hrsg.): Proceedings of the First Westminster Conference on Professional Awareness in Software Engineering (PASE 1996), 1-2 February 1996, London (England). London (England) 1996, S. 47-60.

American Psychological Association /Publication manual/

American Psychological Association (Hrsg.): Publication manual of the American Psychological Association. 5. Aufl., Washington DC 2001.

Aslim /Benutzerbeteiligung/

Sükrü Aslim: Ergebnisse empirischer Untersuchungen zur Benutzerbeteiligung. Dip-

lomarbeit im Fach Wirtschaftsinformatik, Management der Softwareentwicklung an der Universität zu Köln, 2003.

Avcı, Wagner /Anforderungsanalyse/

Oral Avcı, Holger Wagner: Das chronische Problem der Anforderungsanalyse und die Frage: Fehler vermeiden oder früh entdecken? In: Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): Informatik 2005 - Informatik LIVE, Band 2, Lecture Notes in Informatics (LNI), Vol. P-68. Bonn 2005, S. 279-283.

Baitsch u. a. /Büroarbeit/

Christof Baitsch, Christian Katz, Philipp Spinas, Eberhard Ulich: Computerunterstütze Büroarbeit. Ein Leitfaden für Organisation und Gestaltung. Zürich 1989.

Balzert /Software-Entwicklung/

Helmut Balzert: Lehrbuch der Software-Technik. Software-Entwicklung. Heidelberg, Berlin, Oxford 1996.

Balzert /Software-Technik/

Helmut Balzert: Lehrbuch der Software-Technik. Softwareentwicklung. 2. Aufl., Heidelberg, Berlin 2000.

Basili u.a. /Lessons/

Victor R. Basili, Frank E. McGarry, Rose Pajerski, Marvin V. Zelkowitz: Lessons learned from 25 years of process improvement: the rise and fall of the NASA software engineering laboratory. In: ACM (Hrsg.): Proceedings of the 24th International Conference on Software Engineering, May 19 - 25, 2002, Orlando, Florida. New York, NY, USA 2002, S. 69-79.

Basili, Perricone /Software errors/

Victor R. Basili, Barry T. Perricone: Software errors and complexity: an empirical investigation. In: Communications of the ACM. Nr. 1, Jg. 27, 1984, S. 42-52.

Bassin, Biyani, Santhanam /Metrics/

Kathryn Bassin, Shriram Biyani, Padmanabhan Santhanam: Metrics to evaluate vendor-developed software based on test case execution results. In: IBM Systems Journal. Nr. 1, Jg. 41, 2002, S. 13-30.

Bassin, Biyani, Santhanam: /Sydney Olympics/

Kathryn A. Bassin, Shriram Biyani, Padmanabhan Santhanam: Evaluating the software test strategy for the 2000 Sydney Olympics. In: IEEE (Hrsg.): Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE 2001), November 27-30, 2001, Hong Kong. Los Alamitos (California, USA) 2001, S. 264-273.

Bassin, Kratschmer, Santhanam /Software development/

Kathryn A. Bassin, Theresa Kratschmer, Padmanabhan Santhanam: Evaluating software development objectively. In: IEEE Software. Nr. 6, Jg. 15, 1998, S, 66-74.

Bassin, Kratschmer, Santhanam /Software development/

Kathryn Bassin, Theresa Kratschmer, P. Santhanam: Evaluating software development objectively. In: IEEE Software. Nr. 6, Jg. 15, 1998, S. 66-74.

Bassin, Santhanam /Maintenance/

Kathryn Bassin, Padmanabhan Santhanam: Managing the maintenance of ported, outsourced, and legacy software via orthogonal defect classification. In: IEEE (Hrsg.): Proceedings of the 17th IEEE International Conference on Software Maintenance (ICSM 2001), November 6-10, 2001, Florence (Italy). Los Alamitos (California, USA) 2001, S. 726-734.

Bassin, Santhanam /Software triggers/

Kathryn A. Bassin, Padmanabhan Santhanam: Use of software triggers to evaluate software process effectiveness and capture customer usage profiles. In: IEEE (Hrsg).: Proceedings of the Eighth International Symposium on Software Reliability Engineering (ISSRE 1997) - Case Studies, November 02 - 05, 1997, Albuquerque (New Mexico, USA). Los Alamitos (California, USA) 1997, S. 103-114.

Beck /Change/

Kent Beck: Embracing Change with Extreme Programming. In: Computer. Nr. 10, Jg. 32, 1999, S. 70-77.

Becker u. a. /Epistemologische Positionierung/

Jörg Becker, Roland Holten, Ralf Knackstedt, Björn Niehaves: Epistemologische Positionierung in der Wirtschaftsinformatik am Beispiel einer konsensorientierten Informationsmodellierung. In: Ulrich Frank (Hrsg.): Wissenschaftstheorie in Ökonomie und Wirtschaftsinformatik. Wiesbaden 2004, S. 335-366

Bell, Thayer /Software requirements/

T. E. Bell, T. A. Thayer: Software requirements: Are they really a problem? In: IEEE (Hrsg.): Proceedings of the 2nd international conference on software engineering (ICSE), San Francisco, 1976. Los Alamitos (California, USA) 1976, S. 61-68.

Bem /Review article/

Daryl J. Bem: Writing a review article for Psychological Bulletin. In: Psychological Bulletin. Nr. 2, Jg. 118, 1995, S. 172-177.

Ben-Ari /Bug/

Mordechai Ben-Ari: The bug that destroyed a rocket. In: ACM SIGCSE Bulletin. Nr. 2, Jg. 33, 2001, S. 58-59.

Benbasat, Goldstein, Mead /Case research strategy/

Izak Benbasat, David K. Goldstein, Melissa Mead: The case research Strategy in studies of information systems. In: MIS Quarterly. Nr. 3, Jg. 11, 1987, S. 369-386

Benbasat, Zmud /Practice of relevance/

Izak Benbasat, Robert W. Zmud: Empirical research in Information Systems: the practice of relevance. In: MIS Quarterly. Nr. 1, Jg. 23, 1999, S. 3-16.

Berling, Thelin /Case study/

Tomas Berling, Thomas Thelin: An industrial case Study of the verification and validation activities. In: IEEE (Hrsg.): Proceedings of the Ninth International Software Metrics Symposium (METRICS 2003), September 3-5, 2003, Sydney (Australia). Los Alamitos (California, USA) 2003, S. 226-238.

Bhandari /Attribute Focusing/

Inderpal Bhandari: Attribute Focusing: machine-assisted knowledge discovery applied to software production control. In: Knowlegde Acquisition. Nr. 3, Jg. 6, 1994, S. 271-294.

Bhandari u. a. /Case study/

Inderpal Bhandari, Michael Halliday, Eric Tarver, David Brown, Jarir Chaar, Ram Chillarege: A case study of software process improvement during development. In: IEEE Transactions on Software Engineering. Nr. 12, Jg. 19, 1993, S. 1157-1170.

Bhandari u. a. /Improvement/

I. Bhandari, M. J. Halliday, R. Chillarege, K. Jones, J. S. Atkinson, C. Lepori-Costello, P. Y. Jasper, E. D. Tarver, C. C. Lewis, M. Yonezawa: In-process improvement through defect data interpretation. In: IBM Systems Journal. Nr. 1, Jg. 33, 1994, S. 182-214.

Bhandari u. a. /Process feedback/

Inderpal Bhandari, Bonnie Ray, Man-Yuen Wong, David Choi, Akemi Watanabe, Ram Chillarege, Michael Halliday, Alan Dooley, Jarir Chaar: An inference structure for process feedback: technique and implementation. In: Software Quality Journal. Nr. 3, Jg. 3, 1994, S. 167-189.

Bhandari u. a. /NBA data/

Inderpal S. Bhandari, Edward Colet, Jennifer Parker, Zachary Pines, Rajiv Pratap, Krishnakumar Ramanujam: Advanced Scout: Data Mining and Knowledge Discovery in NBA Data. In: Data Mining and Knowledge Discovery. Nr. 1, Jg. 1, 1997, S. 121-125.

Blackburn, Scudder, Van Wassenhove /Speed/

Joseph D. Blackburn, Gary D. Scudder, Luk N. Van Wassenhove: Improving speed and productivity of software development: a global survey of software developers. In: IEEE Transactions on Software Engineering. Nr. 12, Jg. 22, 1996, S. 875-885.

Blaha, Rumbaugh /UML/

Michael Blaha, James Rumbaugh: Object-oriented modeling and design with UML. 2. Aufl., Upper Saddle River (New Jersey) 2004.

Boehm /Economics/

Barry W. Boehm: Software Engineering Economics. Upper Saddle River (New Jersey, USA) 1981.

Boehm /Experiment/

Barry W. Boehm: An experiment in small-scale application software engineering. In: IEEE Transactions on Software Engineering. Nr. 5, Jg. 7, 1981, S. 482-493.

Boehm /Requirements engineering/

Barry W. Boehm: Requirements engineering at age 20: looking back, looking ahead. In: IEEE (Hrsg.): Proceedings of the Second International Conference on Requirements

Engineering (ICRE '96), April 15-18, 1996, Colorado Springs, Colorado. Los Alamitos, California 1996, S. 255.

Boehm /Software/

Barry W. Boehm: Software Engineering. In: IEEE Transactions on Computers. Nr. 12, Jg. 25, 1976, S. 1226-1241.

Boehm, Basili /Software defect/

Barry Boehm, Victor R. Basili: Software defect reduction list. In: Computer. Nr. 1, Jg. 34, 2001, S. 135-137.

Boldyreff u. a. /Traceability/

C. Boldyreff, E. L. Burd, R. M. Hather, M. Munro, E. J. Younger: Greater understanding through maintainer driven traceability. In: IEEE (Hrsg.): Proceedings of the 4th International Workshop on Program Comprehension (WPC 1996), March 29-31, 1996, Berlin. Los Alamitos (California, USA) 1996, S. 100-106.

Booth, Colomb, Williams /Research/

Wayne C. Booth, Gregory G. Colomb, Joseph M. Williams: The craft of research. 2. Aufl., Chicago, London 2003.

Bowen, Heales, Vongphakdi /Reliability factors/

Paul L. Bowen, Jon Heales, Monthira T. Vongphakdi: Reliability factors in business software: volatility, requirements and end-users. In: Information Systems Journal. Nr. 3, Jg. 12, 2002, S. 185-213.

Briand u.a. /Change analysis process/

Lionel C. Briand, Victor R. Basili, Yong-Mi Kim, Donald R. Squier: A change analysis process to characterize software maintenance projects. In: Hausi A. Müller, Mari Georges (Hrsg.): Proceedings of the International Conference on Software Maintenance (ICSM 1994), Victoria, BC, Canada, September 1994. Washington, DC, USA 1994, S. 38-49.

Briand, El Emam, Melo /Inductive method/

Lionel Briand, Khaled El Emam, Walcélio L. Melo: An inductive method for software process improvement: concrete steps and guidelines. In: Khaled El Emam, Nazim H.

Madhavji (Hrsg.): Elements of software process assessment and improvement. Los Alamitos (California) 1999, S. 113-130.

Bridge, Miller /Orthogonal Defect Classification/

Norman Bridge, Corinne Miller: Orthogonal Defect Classification: using defect data to improve software development. In: American Society for Quality (Hrsg.): Proceedings of 7th International Conference on Software Quality (ICSQ'97), October 6-8 1997, Montgomery, Alabama, USA. O. O. 1997, S. 197-213.

Brodie, Stonebraker /Legacy systems/

Michael L. Brodie, Michael Stonebraker: Migrating legacy systems. Gateways, interfaces and the incremental approach. San Francisco (USA, California), 1995.

Brooks /Man-month/

Frederick P. Brooks: The mythical man-month: essays on software engineering. Anniversary Edition. Reading (Massachusetts) u. a. 1995.

Buczilowski /Defect reduction/

Harald Buczilowski: Defect reduction in VSE. Methodology and results. In: IEEE (Hrsg.): Proceedings of the Sixth International Symposium on Software Reliability Engineering (ISSRE 1995), October 24-27, 1995, Toulouse (France). Los Alamitos (California, USA) 1995, S. 274-277.

Butcher, Munro, Kratschmer /Software testing/

Mark Butcher, Hilora Munro, Theresa Kratschmer: Improving software testing via ODC: Three case studies. In: IBM Systems Journal. Nr. 1, Jg. 41, 2002, S. 31-44.

Büttemeyer /Wissenschaftstheorie/

Wilhelm Büttemeyer: Wissenschaftstheorie für Informatiker. Heidelberg, Berlin, Oxford 1995.

Card /Mistakes/

David N. Card: Learning from Our Mistakes with Defect Causal Analysis. In: IEEE Software. Nr. 1, Jg. 15, 1998, S. 56-63.

Chaar u. a. /In-process evaluation/

Jarir K. Chaar, Michael J. Halliday, Inderpal S. Bhandari, Ram Chillarege: In-process

evaluation for software Inspection and test. IEEE Transactions on Software Engineering. Nr. 11, Jg. 19, 1993, S. 1055-1070.

Chatzoglou /Factors/

Prodomos D. Chatzoglou: Factors affecting completion of the requirements capture stage of projects with different characteristics. In: Information and Software Technology. Nr. 9, Jg. 39, 1997, S. 627-640.

Chatzoglou /Methodologies/

P. D. Chatzoglou: Use of methodologies: an empirical analysis of their impact on the economics of the development process. In: European Journal of Information Systems. Nr. 4, Jg. 6, 1997, S. 256-270.

Checkland /Systems Thinking/

Peter Checkland: Systems Thinking, Systems Practice. Includes a 30-Year Retrospective. Chichester u. a. 1999.

Chernak /Statistical approach/

Yuri Chernak: A statistical approach to the inspection checklist formal synthesis and improvement. IEEE Transactions on Software Engineering. Nr. 12, Jg. 22, 1996, S. 866-874.

Chillarege /Defect/

Ram Chillarege: Orthogonal Defect Classification. In: Michael R. Lyu (Hrsg.): Handbook of software reliability engineering. Los Alamitos, California u. a. 1996, S. 359-400.

Chillarege u. a. /In-process measurements/

Ram Chillarege, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnie K. Ray, Man-Yuen Wong: Orthogonal defect classification - a concept for in-process measurements. In: IEEE Transactions on Software Engineering. Nr. 11, Jg. 18, 1992, S. 943-956.

Chillarege, Bassin /Software triggers/

Ram Chillarege and Kathryn A. Bassin: Software triggers as a function of time. ODC on field faults. In: IFIP (Hrsg.): Proceedings of the Fifth IFIP Working Conference on Dependable Computing for Critical Applications (DCCA-5), September 27-29, 1995,

Urbana-Champaign (Illinois, USA). O. O. 1995, S. 327-342.
(Auch online unter http://www.chillarege.com/odc/articles/trig/trig.html verfügbar. Stand 31.09.2006)

Chillarege, Biyani /Risk/
Ram Chillarege, Shriram Biyani: Identifying risk using ODC based growth models. In: IEEE (Hrsg.): Proceedings of the 5th International Symposium on Software Reliability Engineering (ISSRE 1994), November 6-9, 1994, Monterey (California, USA). Los Alamitos (California, USA) 1994, S. 282-288.

Chillarege, Kao, Condit /Defect type/
Ram Chillarege, Wei-Lun Kao, Richard G. Condit: Defect type and its impact on the growth curve. In: IEEE (Hrsg.): Proceedings of the 13th International Conference on Software Engineering (ICSE 1991), May 13-16, 1991, Austin (Texas, USA). Los Alamitos (California, USA) 1991, S. 246-255.

Chillarege, Prasad /ODC Triggers/
Ram Chillarege and Kothanda Ram Prasad: Test and development process retrospective. A case study using ODC Triggers. In: IEEE (Hrsg.): Proceedings of the International Conference on Dependable Systems and Networks (DSN 2002), June 23-26, 2002, Washington, D.C. (USA). Los Alamitos (California, USA) 2002, S. 669-678.

Christel, Kang /Requirements elicitation/
Michael G. Christel, Kyo C. Kang: Issues in requirements elicitation. Software Engineering Institute, Technical Report CMU/SEI-92-TR-12. Pittsburgh, 1992.
(Auch online unter http://www.sei.cmu.edu/publications/documents/92.reports/92.tr.012.html verfügbar. Stand 24.09.2006)

Christmansson, Santhanam /Error injection/
J. Christmansson, P. Santhanam: Error injection aimed at fault removal in fault tolerance mechanisms. Criteria for error selection using field data on software faults. In: IEEE (Hrsg.): Proceedings of the Seventh International Symposium on Software Reliability Engineering (ISSRE 1996), October 30 – November 2, 1996, White Plains (New York, USA). Los Alamitos (California, USA) 1996, S. 175-184.

Coldewey /Entwicklung/

Jens Coldewey: Agile Entwicklung Web-basierter Systeme: Einführung und Überblick. In: Wirtschaftsinformatik. Nr. 3, Jg. 44, 2002, S. 237-248.

Coldewey /Wandel/

Jens Coldewey: Beständig ist nur der Wandel: agile Entwicklung und Änderbarkeit. In: Objektspektrum. Nr. 6, 2001, S. 78-83.

Collofello, Balcom /Software error classification/

James S. Collofello, L. B. Balcom: A proposed causative software error classification scheme. In: American Fede-ration of Information Processing Societies (Hrsg.): Proceedings of the National Computer Conference (Vo-lume 53), Las Vegas, Nevada, USA, July 9-12, 1984. Reston (Virginia, USA) 1984, S. 537-545.

Crowston, Kammerer /Coordination/

Kevin Crowston, Ericka Eve Kammerer: Coordination and collective mind in software requirements development. In: IBM Systems Journal. Nr. 2, Jg. 37, 1998, S. 227-246.

Curtis, Krasner, Iscoe /field study/

Bill Curtis, Herb Krasner, Neil Iscoe: A field study of the software design process for large systems. In: Communications of the ACM. Nr. 11, Jg. 31, 1988, S. 1268-1287.

Cusumano, Selby /Microsoft-Methode/

Michael A. Cusumano, Richard W. Selby: Die Microsoft-Methode. Sieben Prinzipien, wie man ein Unternehmen an die Weltspitze bringt. München 1997.

Dalal u. a. /Defect patterns/

Siddhartha Dalal, Michael Hamada, Paul Matthews, Gardner Patton: Using defect patterns to uncover opportunities for improvement. In: Software Quality Engineering (Hrsg.): Proceedings of the International Conference on Applications of Software Measurement (ASM 1999), February 15-19, 1999, San Jose (California, USA). O.O. 1999, o. S.

(Auch online unter http://www.argreenhouse.com/papers/paul4/edaP4.pdf verfügbar. Stand 31.09.2006)

Daly /Software Development/

Edmund B. Daly: Management of Software Development. In: IEEE Transactions on Software Engineering. Nr. 3, Jg. 3, 1977, S. 229-242.

Damian u. a. /Case study/

Daniela Damian, Didar Zowghi, Lakshminarayanan Vaidyanathasamy, Yogendra Pal: An industrial case study of immediate benefits of requirements engineering process improvement at the Australian Center for Unisys Software. In: Empirical Software Engineering. Nr. 1-2, Jg. 9, 2004, S. 45-75.

Damian u. a. /Downstream development/

Daniela Damian, James Chisan, Lakshminarayanan Vaidyanathasamy, Yogendra Pal: An Industrial Case Study of the Impact of Requirements Engineering on Downstream Development. In: IEEE (Hrsg.): Proceedings of the 2nd International Symposium on Empirical Software Engineering (ISESE 2003), September/October 2003, Roman Castles (Rome, Italy). Los Alamitos (California, USA) 2003, S. 40-49.

Damian u. a. /Process improvement/

Daniela Damian, Didar Zowghi, Lakshminarayanan Vaidyanathasamy, Yogendra Pal: An industrial experience in process improvement: an early assessment at the Australian Center for Unisys Software. In: IEEE (Hrsg.): Proceedings of the 1st International Symposium on Empirical Software Engineering (ISESE 2002), October 3-4 2002, Nara (Japan), October 2002. Los Alamitos (California, USA) 2002, S. 111-126.

Damian u. a. /Requirements engineering/

Daniela Damian, James Chisan, Lakshminarayanan Vaidyanathasamy, Yogendra Pal: Requirements engineering and downstream software development: findings from a case study. In: Empirical Software Engineering. Nr. 3, Jg. 10, 2005, S. 255-283.

Damian, Zowghi / Requirements engineering/

Daniela E. Damian, Didar Zowghi: Requirements engineering challenges in multi-site software development organisations. In: Requirements Engineering. Nr. 3, Jg. 8, 2003, S. 149-160.

Damian, Zowghi /Impact/

Daniela E. Damian, Didar Zowghi: The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization. In: : IEEE (Hrsg.): Pro-

ceedings of 10th Anniversary IEEE Joint International Conference on Requirements Engineering (ICRE 2002), September 9-13 2002, Essen (Germany). Los Alamitos (California, USA) 2002, S. 319-330.

Damm, Lundberg /Test process improvement/

Lars-Ola Damm, Lars Lundberg: Identification of test process improvements by combining fault trigger classification and faults-slip-through measurement. In: IEEE (Hrsg.): Proceedings of the 4th Symposium on Empirical Software Engineering (ISESE 2005), November 17-18, 2005, Noosa Heads (Australia). Los Alamitos (California, USA) 2005, S. 152-161.

Damm, Lundberg, Wohlin /Improvement potential/

Lars-Ola Damm, Lars Lundberg, Claes Wohlin: Determining the improvement potential of a software development organization through fault analysis: a method and a case study. In: T. Dingsoyr (Hrsg.): Software Process Improvement. Proceedings of the 11th European Conference on Software Process Improvement (EuroSPI 2004), November 10-12, 2004, Trondheim (Norway) 2004. Springer Lecture Notes in Computer Science 3281. Berlin, Heidelberg 2004, S. 138-149.

Daneva /ERP/

Maya Daneva: ERP requirements engineering practice: lessons learned. In: IEEE Software. Nr. 2, Jg. 21, 2004, S. 26-33.

Davis /Software requirements/

Alan M. Davis: Software requirements: objects, functions and states. Englewood Cliffs, New Jersey 1993.

Dawson, Swatman /Requirements engineering/

Linda Dawson, Paul Swatman: The use of object-oriented models in requirements engineering: a field study. In: ACM (Hrsg.): Proceeding of the 20th international conference on information systems (ICIS ‘99), December 12 - 15, 1999, Charlotte (North Carolina, USA). Atlanta (Georgia, USA) 1999, S. 260-273.

Dion /Process improvement/

Raymond Dion: Process improvement and the corporate balance sheet. In: IEEE Software. Nr. 4, Jg. 10, 1993, S. 28-35.

Dromey /Software product quality/

R. Geoff Dromey: A model for software product quality. In: IEEE Transaction on Software Engineering. Nr. 2, Jg. 21, 1995, S. 146-162.

Dubé, Paré /Case research/

Line Dubé, Guy Paré: Rigor in information systems positivist case research: current practices, trends and recommendations. In: MIS Quarterly. Nr. 4, Jg. 27, 2003, S. 597-635.

Ebert, De Man /Requirements uncertainty/

Christof Ebert, Jozef De Man: Requirements uncertainty. Influencing factors and concrete improvements. In: ACM (Hrsg.): Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), May 15 - 21, 2005, St. Louis (Missouri, USA). New York (New York, USA) 2005, S. 553-560.

Eisenberg /Literature review/

Nancy Eisenberg: Writing a literature review. In: Robert J. Sternberg (Hrsg.): Guide to publishing in psychological journals. Cambridge u. a. 2000, S. 17-34.

Eisenhardt /Case study research/

K. M. Eisenhardt: Building theories from Case study research. In: Academy of Management Review. Nr. 4, Jg. 4, 1989, S. 532-550.

El Emam, Madhavji /Requirements Engineering/

Khaled El Emam, Nazim H. Madhavji: A field study of requirements engineering practices in information systems development. In: IEEE (Hrsg.): Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE '95), March 27 - 29, 1995, York (England). Los Alamitos (California, USA) 1995, S. 68-80.

El Emam, Madhavji /Success/

Khaled El Emam, Nazim H. Madhavji: Measuring the success of requirements engineering processes. In: IEEE (Hrsg.): Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE '95), March 27 - 29, 1995, York, England. Los Alamitos, California 1995, S. 204-211.

El Emam, Wieczorek /Repeatability/

Khaled El Emam, Isabella Wieczorek: The repeatability of code defect classifications.

In: IEEE (Hrsg.): Proceedings of the Ninth International Symposium on Software Reliability Engineering (ISSRE 1998), November 4-7, 1998, Paderborn (Germany). Los Alamitos (California, USA), S. 322-333.

Endres /Analysis/

Albert Endres: An analysis of errors and their causes. In: ACM (Hrsg.): Proceedings of the International Conference on Reliable Software, Los Angeles, California, April 21-23, 1975. New York 1975, S. 327-336.

Fahrmeir u.a. /Statistik/

Ludwig Fahrmeir, Rita Künstler, Iris Pigeot, Gerhard Tutz: Statistik. Der Weg zur Datenanalyse. 5. Aufl., Berlin, Heidelberg, New York 2004.

Fenton, Neil /Critique/

Norman E. Fenton, Martin Neil: A Critique of Software Defect Prediction Models. In: IEEE Transactions on Software Engineering. Nr. 5, Jg. 25, 1999, S. 675-689.

Fenton, Pfleeger /Software metrics/

Norman E. Fenton, Shari Lawrence Pfleeger: Software metrics. A rigorous and practical approach. 2. Aufl., Boston u. a. 1997.

Fenton, Pfleeger, Glass /Science and Substance/

Norman Fenton, Shari Lawrence Pfleeger, Robert L. Glass: Science and Substance. A Challenge to Software Engineers. In: IEEE Software. Nr. 4, Jg. 11, 1994, S. 86-95.

Fredericks, Basili /Defect tracking/

Michael Fredericks, Victor Basili: Using defect tracking and analysis to improve software quality. A DACS state-of-the-art report. Air Force Research Laboratory, Report Number DACS-SOAR-98-2. Rome (New York, USA) 1998.
(Auch online unter http://www.thedacs.com/techs/defect/ verfügbar. Stand 06.10.2006)

Freimut /Defect classification schemes/

Bernd Freimut: Developing and Using Defect Classification Schemes. IESE-Report No. 072.01/E, Version 1.0. Kaiserslautern 2001.
(Auch online unter http://www.iese.fhg.de/pdf_files/iese-072_01.pdf verfügbar. Stand 08.07.2006)

Freimut u. a. /Empirical Studies/

Bernd Freimut, Teade Punter, Stefan Biffl, Markus Ciolkowski: State-of-the-art in empirical studies. Virtuelles Software Engineering Kompetenzzentrum, ViSEK/007/E, Version 1.0. O.O. 2002.
(Auch online unter http://wwwagse.informatik.uni-kl.de/pubs/repository/freimut2002/ViSEK_Report_007E_v1_0.pdf verfügbar. Stand 18.08.2004)

Freimut, Denger /Defect classification scheme/

Bernd Freimut, Christian Denger: A defect classification scheme for the inspection of QUASAR requirements documents. IESE-Report No. 076.03/E, Version 1.0. Kaiserslautern 2003.
(Auch online unter http://publica.fraunhofer.de/eprints/N-18986.pdf verfügbar. Stand 03.07.2007)

Frese /Aufgabenanalyse/

Erich Frese: Aufgabenanalyse und –synthese. In: Erwin Grochla (Hrsg.): Handwörterbuch der Organisation. 2. Aufl., Stuttgart 1980, S. 207

Gadenne /Bewährung/

Volker Gadenne: Bewährung. In: Herbert Keuth (Hrsg.): Karl Popper, Logik der Forschung. 2. Aufl., Berlin 2004, S. 125-144.

Gadenne /Rationalismus/

Volker Gadenne: Der kritische Rationalismus und die Rolle von Theorien in der Wirtschaftsinformatik. In: Reinhard Schütte, Jukka Siedentopf, Stephan Zelewski (Hrsg.): Wirtschaftsinformatik und Wissenschaftstheorie. Grundpositionen und Theoriekerne: Arbeitsberichte des Instituts für Produktion und Industrielles Informationsmanagement. Nr. 4, Universität Essen. Essen 1998, S. 5-19.
(Auch online http://www.pim.uni-essen.de/wissportal/bibliographie/bericht4.pdf verfügbar. Stand 21.07.2006)

Gerhardt /Strategie/

Tilman Gerhardt: Strategie und Struktur in der deutschen Softwareindustrie: eine industrieökonomische Untersuchung der Unternehmensentwicklung in der Softwarebranche. München 1992.

Gotel, Finkelstein /Requirements traceability/

Orlena C. Z. Gotel, Anthony C. W. Finkelstein: An analysis of the requirements traceability Problem. In: IEEE (Hrsg.): Proceedings of the First International Conference On Requirements Engineering (ICRE 1994), April 18-22, 1994, Colorado Springs, Colorado. Los Alamitos (USA) 1994, S. 94-101.

Grady /Software failure analysis/

Robert B. Grady: Software failure analysis for high-return process improvement decisions. In: Robin B. Hunter, Richard H. Thayer (Hrsg.): Software process improvement. Los Alamitos (California, USA) 2001, S. 155-162d.
Es handelt sich bei diesem Beitrag um ein Nachdruck des Artikels: Robert B. Grady: Software failure analysis for high-return process improvement decisions. In: Hewlett-Packard Journal. Nr. 4, Jg. 47, 1996, S. 15-24.
(Auch online unter http://www.hpl.hp.com/hpjournal/96aug/aug96a2.htm verfügbar. Stand 13.03.2006)

Grady /Software metrics/

Robert B. Grady: Practical software metrics for project management and process improvement. Englewood Cliffs, New Jersey 1992.

Grady /Software quality/

Robert B. Grady: Practical results from measuring software qualiy. In: Communications of the ACM. Nr. 11, Jg. 36, 1993, S. 62-68.

Grochla /Organisationstheorie/

Erwin Grochla: Einführung in die Organisationstheorie. Stuttgart, 1978.

Guba, Lincoln /Paradigms/

Egon G. Guba, Yvonna S. Lincoln: Competing paradigms in qualitative research. In: Norman K. Denzin, Yvonna S. Lincoln (Hrsg.): Handbook of qualitative research. Thousand Oaks 1994, S. 105-117.

Günther, Rombach, Ruhe /Qualitätsverbesserung/

Holger Günther, H.D. Rombach, Günther Ruhe: Kontinuierliche Qualitätsverbesserung in der Software-Entwicklung. Erfahrungen bei der Allianz Lebensversicherungs-AG. In: Wirtschaftsinformatik. Nr. 2, Jg. 38, 1996, S. 160-171.

Haag, Raja, Schkade /Quality Function Deployment/

Stephen Haag, M.K. Raja, and L.L. Schkade: Quality Function Deployment usage in software development. In: Communications of the ACM. Nr. 1, Jg. 39, 1996, S. 41-49.

Hall, Beecham, Rainer /Requirements/

Tracy Hall, Sarah Beecham, Austen Rainer: Requirements problems in twelve software companies: an empirical analysis. In: IEE Proceedings - Software. Nr. 5, Jg. 149, 2002, S. 153-160.

Halliday u. a. /Experiences/

Michael Halliday, Inderpal Bhandari, Jarir Chaar, Ram Chillarege: Experiences in transferring a software process improvement methodology to production laboratories. In: Journal of Systems and Software. Nr. 1, Jg. 26, 1994, S. 61-68.

Harter, Krishnan, Slaughter /Process maturity/

Donald E. Harter, Mayuram S. Krishnan, Sandra A. Slaughter: Effects of process maturity on quality, cycle time, and effort in software product development. In: Management Science. Nr. 4, Jg. 46, 2000, S. 451-466.

Harter, Slaughter /Quality improvement/

Donald E. Harter, Sandra A. Slaughter: Quality Improvement and infrastructure activity costs in software development: a longitudinal analysis. In: Management Science. Nr. 6, Jg. 49, 2003, S. 784-800.

Haug /Wissenschaftstheoretische Problembereiche/

Sonja Haug: Wissenschaftstheoretische Problembereiche empirischer Wirtschafts- und Sozialforschung. Induktive Forschungslogik, naiver Realismus, Instrumentalismus, Relativismus. In: Ulrich Frank (Hrsg.): Wissenschaftstheorie in Ökonomie und Wirtschaftsinformatik. Wiesbaden 2004, S. 85-107

Hayes /Requirement fault/

Jane Huffman Hayes: Building a requirement fault taxonomy: experiences from a NASA verification and validation research project. In: IEEE (Hrsg.): Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE 2003), 17-20 November 2003, Denver, Colorado. 2003. Los Alamitos (USA) 2003, S. 49-59.

Heinbokel /Benutzerbeteiligung/

Torsten Heinbokel: Benutzerbeteiligung: Schlüssel zum Erfolg oder Hemmschuh der Entwicklung? In: Felix C. Brodbeck, Michael Frese (Hrsg.): Produktivität und Qualität in Softwareprojekten. Psychologische Analyse und Optimierung von Arbeitsprozessen in der Softwareentwicklung. München, Oldenbourg 1994, S. 105-124.

Heinrich /Forschungsmethodik/

Lutz J. Heinrich: Forschungsmethodik einer Integrationsdisziplin: Ein Beitrag zur Geschichte der Wirtschaftsinformatik. In: NTM International Journal of History & Ethics of Natural Sciences, Technology & Medicine. Nr. 2, Jg. 13, 2005, S. 104-117.

Heinrich /Wirtschaftsinformatik/

Lutz J. Heinrich: Wirtschaftsinformatik. Einführung und Grundlegung. 2. Aufl., München, Wien, Oldenburg 2001.

Heinzl /Aktivitätsniveau/

Armin Heinzl.: Zum Aktivitätsniveau empirischer Forschung in der Wirtschaftsinformatik - Erklärungsansatz und Handlungsoptionen. Universität Bayreuth, Arbeitspapier des Lehrstuhls für Wirtschaftsinformatik, Nr. 7/2001. Bayreuth 2001.
(Auch online unter
http://wifo1.bwl.uni-mannheim.de/fileadmin/files/publications/Arbeitspapier2001_7.pdf verfügbar. Stand 05.07.2006)

Henningsson, Wohlin /Fault classification agreement/

Kennet Henningsson, Claes Wohlin: Assuring fault classification agreement: an empirical evaluation. In: IEEE (Hrsg.): Proceedings of the 3rd International Symposium on Empirical Software Engineering (ISESE 2004), August 19-20, 2004, Redondo Beach (Colifornia, USA). Los Alamitos (California, USA) 2004, S. 95-104.

Herzwurm /Softwareproduktentwicklung/

Georg Herzwurm: Systematische Herleitung eines Instrumentariums zur kundenorientierten Softwareproduktentwicklung. Habil. Köln 1998.

Herzwurm, Schockert, Mellis /QFD/

Georg Herzwurm, Sixten Schockert, Werner Mellis: Joint Requirements Engineering. QFD for Rapid Customer-Focused Software and Internet Development. Braunschweig, Wiesbaden 2000.

Hickey, Davis / Elicitation technique /

Ann M. Hickey, Alan M. Davis: Elicitation technique selection: how do experts do it? In: IEEE (Hrsg.): Proceedings of the 11th IEEE International Conference on Requirements Engineering (RE '03), September 8- 12, 2003, Monterey Bay, CA, USA. Los Alamitos (USA) 2003, S. 169-178.

Hickey, Davis /Requirements elicitation/

Ann M. Hickey, Alan M. Davis: A unified model of requirements elicitation. In: Journal of Management Information Systems. Nr. 4, Jg. 20, 2004, S. 65-84.

Hierholzer /Benchmarking/

Andreas Hierholzer: Benchmarking der Kundenorientierung von Softwareprozessen. Konzeption eines Vorgehensmodells zur kontinuierlichen Verbesserung der Kundenorientierung von Softwareherstellern. Aachen 1996.

Hoffmann /Aufgabe/

Friedrich Hoffmann: Aufgabe. In: Erwin Grochla (Hrsg.): Handwörterbuch der Organisation. 2. Aufl., Stuttgart 1980, S. 200-208.

Hofmann, Lehner /Requirements engineering/

Hubert F. Hofmann, Franz Lehner: Requirements Engineering as a success factor in software projects. In: IEEE Software. Nr. 4, Jg. 18, 2001, S. 58-66.

Houdek, Pohl /Requirements Engineering/

Frank Houdek, Klaus Pohl: Analyzing Requirements Engineering Processes: a case study. In: IEEE (Hrsg.): Proceedings of the 11th IEEE International Workshop on Database and Expert Systems Applications (DEXA ’00), Los Angeles, 2000. Los Alamitos 2000, S. 983-987.

Hoyer, Hoyer /Quality/

Robert W. Hoyer, Brooke B.Y. Hoyer: What Is Quality? In: Quality Progress. Nr. 7, Jg. 34, 2001, S. 53-62.

Hunton, Beeler /User participation/

James E. Hunton, Jesse D. Beeler: Effects of user participation in systems development: a longitudinal field experiment. In: MIS Quarterly. Nr. 4, Jg. 21, 1997, S. 359-388.

IBM Research /ODC Extensions/
IBM Research (Hrsg.): ODC Extensions V5.11.
http://www.research.ibm.com/softeng/SDA/EXTODC.HTM, Abruf am 07.06.2007.

IBM Research /ODC/
IBM Research (Hrsg.): Details of ODC v 5.11.
http://www.research.ibm.com/softeng/ODC/DETODC.HTM, Abruf am 31.08.2004.

IEEE /Glossary/
IEEE (Hrsg.): IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 620.12-1990. New York 1990.

IEEE /Guide/
IEEE (Hrsg.): IEEE guide to classification for software anomalies. IEEE Std. 1044.1-1995. New York 1996.

IEEE /Software Anomalies/
IEEE (Hrsg.): IEEE standard classification for software anomalies. IEEE Std. 1044-1993. New York 1993.

IEEE /Software Maintenance/
IEEE (Hrsg.): IEEE standard for software maintenance. IEEE Std. 1219-1998. New York 1998.

IEEE /Software requirements/
IEEE (Hrsg.): IEEE Recommended practice for Software requirements specifications. IEEE Std. 830-1998. New York 1998.

IEEE /SWEBOK/
IEEE (Hrsg.): Guide to the Software Engineering Body of Knowledge (SWEBOK). 2004 Version. Los Alamitos u. a. 2004.
(Auch online unter http://www.swebok.org/ironman/pdf/SWEBOK_Guide_2004.pdf verfügbar. Stand 23.05.2005)

International Software Testing Qualification Board /Standard glossary/
International Software Testing Qualification Board (Hrsg.): Standard glossary of terms used in Software Testing. Version 1.2 (June, 4th 2006).
http://www.istqb.org/downloads/glossary-current.pdf, Abruf am 10.06.2007.

Ishikawa /Quality Control/

Kaoru Ishikawa: Guide to quality control. 11. Aufl., Tokyo 1993.

Ives, Olson /User involvement/

Blake Ives, Margrethe H. Olson: User Involvement and MIS success: a review of research. In: Management Science. Nr. 5, Jg. 30, 1984, S. 586-603.

Jacobs u. a. /Defect causes/

Jef Jacobs, Jan van Moll, Paul Krause, Rob Kusters, Jos Trienekens, Aarnout Brombacher: Exploring defect causes in products developed by virtual teams. In: Information and Software Technology. Nr. 6, Jg. 47, 2005, S. 399-410.

Jones /Software Costs/

T. Capers Jones: Estimating Software Costs. New York u. a. 1998.

Juhos /Positivismus/

Belá Juhos: Formen des Positivismus. In: Journal for General Philosophy of Science. Nr. 1, Jg. 2, 1971, S. 27-62.

Juristo, Moreno, Silva /Requirements Engineering/

Natalia Juristo, Ana M. Moreno, Andrés Silva: Is the European industry moving toward solving requirements engineering problems? In: IEEE Software. Nr. 6, Jg. 19, 2002, S. 70-77.

Kamsties, Hörmann, Schlich /Requirements Engineering/

Erik Kamsties, Klaus Hörmann, Maud Schlich: Requirements Engineering in Small and Medium Enterprises. In: Requirements Engineering. Nr. 2, 1998, S. 84-90.

Kan /Metrics/

Stephen H. Kan: Metrics and Models in Software Engineering Quality Engineering. 2. Aufl., Boston u. a. 2004.

Kaner, Falk, Nguyen /Computer software/

Cem Kaner, Jack Falk, Hung Quoc Nguyen: Testing computer software. 2. Aufl., New York 1999.

Kauppinen u. a. /Cultural change/

Marjo Kauppinen, Sari Kujala, Tapani Aaltio, Laura Lehtola: Introducing requirements

engineering: how to make a cultural change happen in practice. In: IEEE (Hrsg.): Proceedings of 10th Anniversary IEEE Joint International Conference on Requirements Engineering (ICRE 2002), September 9-13 2002, Essen (Germany).). Los Alamitos (California, USA) 2002, S. 43-51.

Kauppinen, Aaltio, Kujala /Requirements Engineering/

Marjo Kauppinen, Tapani Aaltio, Sari Kujala: Lessons learned from applying the Requirements Engineering Good Practice Guide for process improvement. In: Jyrki Kontio, Reidar Conradi (Hrsg.): Software Quality - ECSQ 2002. Quality Connection - 7th European Conference on Software Quality, Helsinki, Finland, June 9-13, 2002. Proceedings. Berlin u.a. 2002, S. 73-81.

Kauppinen, Kujala /Improvement/

Marjo Kauppinen, Sari Kujala: Starting Improvement of Requirements Engineering Processes: an experience report. In: Frank Bomarius, Seija Komi-Sirviö (Hrsg.): Product focused software process improvement. Third International Conference, PROFES 2001, Kaiserslautern, Germany, September 2001. Proceedings. Berlin u. a. 2001, S. 196-209.

Kerzner /Project management/

Harold Kerzner: Project management. A systems spproach to planning, scheduling, and controlling. 8. Aufl., Hoboken (New Jersey, USA) 2003.

Kitchenham u. a. /Guidelines/

Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam, Jarret Rosenberg: Preliminary guidelines for empirical research in software engineering. In: IEEE Transactions on Software Engineering. Nr. 8, Jg. 28, 2002, S. 721-734.

Kitchenham u. a. /Guidelines/

Barbara A. Kitchenham: Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam, Jarrett Rosenberg: Preliminary guidelines for empirical research in software engineering. In: IEEE Transactions on Software Engineering. Nr. 8, Jg. 28, 2002, S. 721-734.

Koru, Tian /Defect handling/

A. Günes Koru, Jeff Tian: Defect handling in medium and large open source projects. In: IEEE Software. Nr. 4, Jg. 21, 2004, S. 54-61.

Kosiol /Aufgabenanalyse/

Erich Kosiol: Aufgabenanalyse und Aufgabensynthese. In: Erwin Grochla (Hrsg.): Elemente der organisatorischen Gestaltung. Hamburg 1978, S. 66-84.

Kotonya, Sommerville /Requirements engineering/

Gerald Kotonya, Ian Sommerville: Requirements Engineering. Processes and techniques. Chichester u.a. 1998.

Krauss /Research paradigms/

Steven Eric Krauss: Research paradigms and meaning making: a primer. In: The Qualitative Report. Nr. 4, Jg. 10, 2005, S. 758-770.
(Auch online unter http://www.nova.edu/ssss/QR/QR10-4/krauss.pdf verfügbar. Stand 22.04.2006)

Krishnan u. a. /Productivity/

M. S. Krishnan, C. H. Kriebel, Sunder Kekre, Tridas Mukhopadhyay: An Empirical Analysis of Productivity and Quality in Software Products. In: Management Science. Nr. 6, Jg. 46, 2000, S. 745-759.

Krishnan, Kellner /Process consistency/

M. S. Krishnan, Marc I. Kellner: Measuring process consistency: implications for reducing software defects. In: IEEE Transactions on Software Engineering. Nr. 6, Jg. 25, 1999, S. 800-815.

Kubicek /Organisationsforschung/

Herbert Kubicek: Empirische Organisationsforschung. Konzeption und Methodik. Stuttgart, 1975.

Lamnek /Gruppendiskussion/

Siegfried Lamnek: Gruppendiskussion. Theorie und Praxis. Weinheim 1998.

Lanubile, Shull, Basili /Requirements documents/

Filippo Lanubile, Forrest Shull, Victor R. Basili: Experimenting with error abstraction in requirements documents. In: IEEE (Hrsg.): Proceedings of the Fifth International Software Metrics Symposium (METRICS 1998), March 20-21, 1998, Bethesda, (Maryland, USA). Los Alamitos (California, USA) 1998, S. 114-121.

Latino, Latino /Root cause analysis/

Robert J. Latino, Kenneth C. Latino: Root cause analysis: improving performance for bottom line results. 2 Aufl., Boca Raton (Florida) 2002.

Lauesen, Vinter /Requirement defects/

Soren Lauesen, Otto Vinter: Preventing requirement defects: An experiment in process improvement. In: Requirements Engineering. Nr. 1, Jg. 6, 2001, S. 37-50.

Lee /Case studies/

A. S. Lee: A scientific methodology for MIS case studies, In: MIS Quarterly. Nr. 1, Jg. 13, 1989, S. 33-52.

Lehner /Theoriebildung/

Franz Lehner: Theoriebildung in der Wirtschaftsinformatik. In: Jörg Becker, Wolfgang König, Reinhard Schütte, Oliver Wendt, Stephan Zelewski (Hrsg.): Wirtschaftsinformatik und Wissenschaftstheorie. Bestandsaufnahme und Perspektive. Wiesbaden 1999, S. 5-24.

Leszak, Perry, Stoll /case study/

Marek Leszak, Dewayne E. Perry, Dieter Stoll: A case study in root cause defect analysis. In: ACM (Hrsg): Proceedings of the 22nd International Conference on Software engineering (ICSE 2000), Limerick, Ireland, June 2000. New York 2000, S. 428-437.

Levendel /Reliability analysis/

Y. Levendel: Reliability analysis of large software systems: defect data modeling. In: IEEE Transactions on Software Engineering. Nr. 2, Jg. 16, 1990, S. 141-152.

Levenson, Turner /Therac-25/

Nancy G. Leveson, Clark S. Turner: An investigation of the Therac-25 accidents. In: IEEE Computer. Nr. 7, Jg. 26, 1993, S. 18-41.

Liggesmeyer /Software-Qualität/

Peter Liggesmeyer: Software-Qualität. Testen, analysieren und verifzieren von Software. Heidelberg, Berlin 2002.

Lincoln, Guba /Inquiry/

Yvonna S. Lincoln, Egon G. Guba: Naturalistic Inquiry. Beverly Hills, 1985.

Lubars, Potts, Richter /Requirements modeling/

Mitch Lubars, Colin Potts, Charlie Richter: A review of the state of the practice in requirements modeling. In: IEEE (Hrsg.): Proceedings of IEEE International Symposium on Requirements Engineering (ISRE 1993), January 4-6, 1993, San Diego (California, USA). Los Alamitos (California, USA) 1993, S. 2-14.

Lutz /Software requirements errors/

Robyn R. Lutz: Analyzing software requirements errors in safety-critical, embedded systems. In: IEEE (Hrsg,): Proceedings of IEEE International Symposium on Requirements Engineering (ISRE 1993), San Diego, California, January 4-6, 1993. Los Alamitos (USA) 1993, S. 126-133.

Lutz, Mikulski /Anomalies/

Robyn R. Lutz, Inés Carmen Mikulski: Empirical analysis of safety-critical anomalies during operations. In: IEEE Transactions on Software Engineering. Nr. 3, Jg. 30, 2004, S. 172-180.

Lutz, Mikulski /Requirements Discovery/

Robyn R. Lutz, Inés Carmen Mikulski: Requirements discovery during the testing of safety-critical software. In: IEEE (Hrsg.): Proceedings of the 25th international Conference on Software Engineering (ICSE 2003), May 3-10, 2003, Portland (Oregon, USA). Los Alamitos (California, USA) 2003, S. 578-583.

Lutz, Mikulski /Requirements/

Robyn R. Lutz, Inés Carmen Mikulski.: Ongoing Requirements Discovery in High-Integrity Systems. In: IEEE Software. Nr. 2, Jg. 21, 2004, S. 19-25.

Lutz, Mikulski /Testing/

Robyn R. Lutz, Inés Carmen Mikulski: Resolving requirements discovery in testing and operations. In: IEEE (Hrsg.): Proceedings of the 11th IEEE International Requirements Engineering Conference (RE 2003), September 8-12, 2003, Monterey Bay (California, USA). Los Alamitos (California, USA) 2003, S. 33-41.

Malone, Crowston /Coordination/

Thomas W. Malone, Kevin Crowston: The interdisciplinary study of coordination. In: ACM Computing Surveys. Nr. 1, Jg. 26, 1994, S. 87-119.

Mays u. a. /Defect prevention/

R. G. Mays, C. L. Jones, G. J. Holloway, D. P. Studinski: Experiences with defect prevention. In: IBM Systems Journal. Nr. 1, Jg. 29, 1990, S. 4-32.

McGarry u. a. /Software Process Improvement/

F. McGarry, R. Pajerski, G. Page, S. Waligora, V. Basili, M. Zelkowitz: Software Process Improvement in the NASA Software Engineering Laboratory. Software Engineering Institute, Technical Report CMU/SEI-TR-22. Pittsburgh, 1994. (Auch online unter http://www.sei.cmu.edu/publications/documents/94.reports/94.tr.022.html verfügbar. Stand 18.08.2004)

Mead /Requirements management/

Nancy R. Mead: Requirements Management and Requirements Engineering: You can't have one without the other. In: Cutter IT Journal. Nr. 5, Jg. 3, 2000, S. 4-8

Mellis /Projektmanagement/

Werner Mellis: Projektmanagement der SW-Entwicklung. Eine umfassende und fundierte Einführung. Wiesbaden 2004.

Mellis /Softwareentwicklungsprozess/

Werner Mellis: Der Softwareentwicklungsprozess der Zukunft: 'optimizing', 'agile' oder 'extreme'? In: Hans-Georg Kemper, Wilhelm Mülder (Hrsg.): Informationsmanagement: Neue Herausforderungen in Zeiten des E-Business. Festschrift für Prof. Dr. Dietrich Seibt anlässlich seines 65. Geburtstages. Lohmar 2003, S. 383-408.

Mellis u. a. /Software development/

Werner Mellis, Georg Herzwurm, Uwe Müller, Harald Schlang, Sixten Schockert, Ralph Trittmann: Rapid software development. Aachen 2001.

Mellis, Herzwurm, Stelzer /TQM/

Werner Mellis, Georg Herzwurm, Dirk Stelzer: TQM der Softwareentwicklung: mit Prozessverbesserung, Kundenorientierung und Change Management zu erfolgreicher Software. 2. Aufl., Wiesbaden 1998.

Mellis, Stelzer /Rätsel/

Werner Mellis, Dirk Stelzer: Das Rätsel des prozessorientierten Softwarequalitäts-managements. In: Wirtschaftsinformatik. Nr. 1, Jg. 41, 1999, S. 31-39.

Mertens, Wieczorrek /Strategien/

Peter Mertens, Hans Wilhelm Wieczorrek: Data x Strategien. Data Warehouse, Data Mining und operationale Systeme für die Praxis. Berlin u. a. 2000.

Mingers /Pluralist Methodology/

John Mingers: Combining IS research methods: Towards a pluralist methodology. In: Information Systems Research. Nr. 3, Jg. 12, 2001, S. 240-259.

Mizuno /Software Quality/

Yukio Mizuno: Software Quality Improvement. In: Computer. Nr. 3, 1983, S. 66-72.

Möller /Qualitätsmetriken/

Karl Heinrich Möller: Ausgangsdaten für Qualitätsmetriken - eine Fundgrube für Analysen. In: Christof Ebert, Reiner Dumke (Hrsg.): Software-Metriken in der Praxis: Einführung und Anwendung von Software-Metriken in der industriellen Praxis. Berlin, Heidelberg, New York 1996, S. 105-116.

Morris, Masera, Wilikens /Requirements Engineering/

Philip Morris, Marcelo Masera, Marc Wilikens: Requirements Engineering and Industrial Uptake. In: IEEE (Hrsg.): Proceedings of the Third IEEE International Conference on Requirements Engineering (ICRE '98), April 06 - 10, 1998, Colorado Springs, Colorado. Los Alamitos (California, USA) 1998, S. 130-137.

Müller /Prüf- und Testprozesse/

Uwe Müller: Prüf- und Testprozesse in der Softwareentwicklung. Bestandsaufnahme von Gestaltungsbereichen und deren Einflussfaktoren. Aachen 1999.

Müller, Wiegmann, Avci /Prüf- und Testprozesse/

Uwe Müller, Thomas Wiegmann, Oral Avci: „State of the Practice" der Prüf- und Testprozesse in der Softwareentwicklung. Ergebnisse einer empirischen Untersuchung bei deutschen Softwareunternehmen. Studien zur Systementwicklung des Lehrstuhls für Wirtschaftsinformatik an der Universität zu Köln, Band 16. Köln 1998.

Müller-Böling /Organisationsforschung/

Detlef Müller-Böling: Methodik der empirischen Organisationsforschung. In: Erich Frese (Hrsg.): Handwörterbuch der Organisation. 3. Aufl., Stuttgart 1992, S. 1491-1505.

Munson, Nikora /Definition/

John C. Munson, Allen P. Nikora: Toward A Quantifiable Definition of Software Faults. In: IEEE (Hrsg.): Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE'02), Annapolis, MD, USA, 12-15 November 2002. Los Alamitos, California 2002, S. 388-396.

Nakajo, Kume /Case history analysis/

Takeshi Nakajo, Hitoshi Kume: A case history analysis of software error cause-effect relationships. In: IEEE Transactions on Software Engineering. Nr. 8, Jg. 17, 1991, S. 830-838.

Neill, Laplante /Requirements Engineering/

Colin J. Neill, Phillip A. Laplante: Requirements engineering: the state of the practice. In: IEEE Software. Nr. 6, Jg. 20, 2003, S. 40-45.

Neufelder /Impact/

Ann Marie Neufelder: How to Measure the Impact of Specific Development Practices on Fielded Defect Density. In: IEEE (Hrsg.): Proceedings of 11th International Symposium on Software Reliability Engineering (ISSRE'00), San Jose, CA, October 08 - 11, 2000. Washington, DC, USA 2000, S. 148-160.

Nikula, Sajaniemi, Kälviäinen /Management View/

U. Nikula, J. Sajaniemi, H. Kälviäinen: Management View on Current Requirements Engineering Practices in Small and Medium Enterprises. In: o. Hrsg.: The Fifth Australian Workshop on Requirements Engineering, Queensland University of Technology, Brisbane, Australia, Faculty of Information Technology, University of Sydney. O. O. 2000, S. 81-89.

Nikula, Sajaniemi, Kälviäinen /Requirements Engineering/

U. Nikula, J. Sajaniemi, H. Kälviäinen: A State-of-the-Practice Survey on Requirements Engineering in Small- and Medium-Sized Enterprises. Telecom Business Research Center, Research Report 1, ISBN: 951-764-431-0. Lappeenranta, Finnland 2000.

(Auch online unter http://www2.lut.fi/~unikula/Publications/TBRCRR1.pdf verfügbar. Stand 18.08.2004)

Nonaka /Knowledge/

Ikujiro Nonaka: A dynamic theory of organizational knowledge creation. In: Organization Science. Nr. 1, Jg. 5, 1994, S. 14-37.

Noth, Kretschmar /Aufwandsschätzung/

Thomas Noth, Matthias Kretzschmar: Aufwandsschätzung von DV-Projekten. Darstellung und Praxisvergleich der wichtigsten Verfahren. 2. Aufl., Berlin u. a. 1986.

Orlikowski, Baroudi /Research approaches/

Wanda J. Orlikowski, Jack J. Baroudi: Studying information technology in organizations: research approaches and assumptions. In: Information Systems Research. Nr. 1, Jg. 2, 1991, S. 1-28.

Ostrand, Weyuker /Software error data/

Thomas J. Ostrand, Elaine J. Weyuker: Collecting and Categorizing Software Error Data in an Industrial Environment. In: Journal of Systems and Software. Nr. 4, Jg. 4, 1984, S. 289-300.

Paech, Kamsties /Improvement/

Barbara Paech, Erik Kamsties: Goal-oriented Improvement of Requirements Processes. In: E. Georgiadou, G. King, P. Pouyiotas, M. Ross, G. Staples (Hrsg.): Proceedings of the Fifth International Conference on Software Process Improvement Research, Education & Training (INSPIRE 2000), London, UK, September 2000. London 2000, S. 169-182.

Paré /Case study research/

Guy Paré: Investigating information systems with positivist case study research. In: Communications of the Association for Information Systems. Nr. 18, Jg. 13, 2004, S. 233-264.

Partsch /Requirements-Engineering/

Helmut Partsch: Requirements-Engineering systematisch. Modellbildung für softwaregestützte Systeme. Berlin u.a. 1998.

Paulk u. a. /Capability Maturity Model/

Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, Charles V. Weber: Capability Maturity Model for Software, Version 1.1. Technical Report CMU/SEI-93-TR-024. Pittsburgh, Pennsylvania, 1993.

Paulk, Goldenson, White /Survey/

Mark C. Paulk, Dennis Goldenson, David M. White: The 1999 survey of high maturity organizations. Special Report CMU/SEI-2000-SR-002. Pittsburgh (USA), 2000. (Auch online unter http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00sr002.pdf verfügbar. Stand 17.06.2007)

Perry, Porter, Votta /Empirical studies/

Dewayne Perry, Adam Porter, Lawrence Votta: Empirical studies of software engineering. A roadmap. In: ACM (Hrsg.): Proceedings of the conference on the future of software engineering, Limerick, Ireland, June 2000. New York 2000, S. 345-355.

Peschke /Systementwicklung/

Helmut Peschke: Betroffenenorientierte Systementwicklung. Prozess und Methoden der Entwicklung menschengerechter Informationssysteme. Frankfurt am Main, Bern, New York 1986.

Pohl /Requirements engineering/

Klaus Pohl: Process-centered requirements engineering. New York u. a. 1996.

Potts, Takahashi, Anton /requirements/

Colin Potts, Kenji Takahashi, Annie I. Anton: Inquiry-based requirements analysis. In: IEEE Software. Nr. 2, Jg. 11, 1994, S. 21-32.

Poulin, Brown /Quality improvement/

Jeffrey S. Poulin, David D. Brown: Measurement-driven quality improvement in the MVS/ESA operating system. In: IEEE (Hrsg.): Proceedings of the 2nd International Software Metrics Symposium (METRICS 1994), October 24-26, 1994, London. Los Alamitos (California, USA) 1994, S. 17-25.

Raasch /Strukturierte Methoden/

Jörg Raasch: Systementwicklung mit Strukturierten Methoden. Ein Leitfaden für Praxis und Studium. 3. Aufl., München 1993.

Reason /Human error/

James Reason: Human error. Cambridge, UK 1990.

Robson /Research/

Colin Robson: Real World Research. 2. Aufl., Oxford, Malden (Massachusetts, USA) 2002.

Rolf /Wirtschaftsinformatik/

Arno Rolf: Herausforderungen für die Wirtschaftsinformatik. In: Informatik-Spektrum. Nr. 5, Jg. 21, 1998, S. 259-264.

Roman /Requirements engineering/

Gruia-Catalin Roman: A taxonomy of current issues in requirements engineering. In: Computer. Nr. 4, Jg. 18, 1985, S. 14-23.

Rooney, Heuvel /Root cause analysis/

James J. Rooney, Lee N. Vanden Heuvel: Root Cause Analysis for beginners. In: Quality Progress. Nr. 7, Jg. 37, 2004, S. 45-53.

Ross, Schoman /Structured analysis/

D. T. Ross, K. E. Schoman: Structured analysis for requirements definition. In: IEEE Transactions on Software Engineering. Nr. 1, Jg. 3, 1977, S. 6-15.

Rubey /Software validation/

Raymond J. Rubey: Quantitative aspects of software validation. In: ACM (Hrsg.): Proceedings of the international conference on Reliable software, Los Angeles, April 21 - 23, 1975. New York 1975, S. 246-251.

Sakthivel /Survey/

Sachidanandam Sakthivel: A survey of requirements verification techniques. In: Journal of Information Technology. Nr. 2, Jg. 6, 1991, S. 68-79.

Sawyer, Sommerville, Viller /Requirements Process Improvement/

Pete Sawyer, Ian Sommerville, Stephen Viller: Requirements Process Improvement Throught the phased introduction of good practice. In: Software Process Improvement and Practice. Nr. 1, Jg. 3, 1997, S. 19-34.

Schneider, Martin, Tsai /Study/

G. Michael Schneider, Johnny Martin, W. T. Tsai: An experimental study of fault detection in user requirements documents. In: ACM Transactions on Software Engineering and Methodology. Nr. 2, Jg. 1, 1992, S. 188-204.

Schurz /Induktion/

Gerhard Schurz: Das Problem der Induktion. In: Herbert Keuth (Hrsg.): Karl Popper, Logik der Forschung. 2. Aufl., Berlin 2004, S. 25-40.

Schütte /Basispositionen/

Reinhard Schütte: Basispositionen in der Wirtschaftsinformatik – ein gemäßigt-konstruktivistisches Programm. In: Jörg Becker, Wolfgang König, Reinhard Schütte, Oliver Wendt, Stephan Zelewski (Hrsg.): Wirtschaftsinformatik und Wissenschaftstheorie. Bestandsaufnahme und Perspektive. Wiesbaden 1999, S. 212-241.

Schwarze /Projektmanagement/

Jochen Schwarze: Projektmanagement mit Netzplantechnik. 8. Aufl., Herne, Berlin 2001.

Seaman /Methods/

Carolyn B. Seaman: Qualitative methods in empirical studies of software engineering, In: IEEE Transactions on Software Engineering. Nr. 4, Jg. 25, 1999, S. 557-572.

Seibt /Organisation/

Dietrich Seibt: Organisation von Softwaresystemen. Wiesbaden 1972.

Seibt /Systemlebenszyklus/

Dietrich Seibt: Management des Systemlebenszyklus. In: Peter Mertens (Hrsg.): Lexikon der Wirtschaftsinformatik. 4. Aufl., Berlin 2001, S.456 - 458.

Shanks, Parr /Case study research/

Graemes Shanks, Anne Parr: Positivist single case study research in information systems: a critical analysis. In: C. Ciborra, R. Mercurio, M. de Marco, M. Martinez , A. Carignani (Hrsg.): Proceedings of the Eleventh European Conference on Information Systems (ECIS 2003), Naples, Italy, June 6-21 2003. O. O. 2003, o. S.

Sheldon u. a. /Reliability measurement/

Frederick T. Sheldon, Krishna M. Kavi, Robert C. Tausworthe, James T. Yu, Ralph

Brettschneider, William W. Everett: Reliability measurement: from theory to practice. In: IEEE Software. Nr. 4, Jg. 9, 1992, S. 13-20.

Shooman, Bolsky /Programming errors/

M. L. Shooman, M. I. Bolsky: Types, distribution, and test and correction times for programming errors. In: Martin L. Shooman, Raymond T. Yeh (Hrsg.): Proceedings of the international conference on reliable software, 21-23 April 1975, Los Angeles, California. New York, USA 1975, S. 347-357.

Shull u. a. /Defects/

Forrest Shull, Vic Basili, Barry Boehm, A. Winsor Brown, Patricia Costa, Mikael Lindvall, Dan Port, Ioana Rus, Roseanne Tesoriero, Marvin Zelkowitz: What we have learned about fighting defects. In: IEEE (Hrsg.): Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS 2002), Ottawa, Canada, June 04 - 07, 2002. Los Alamitos, California 2002, S. 249-258.

Silberman /Robot Orthogonal Defect Classification/

Jack Silberman: Robot Orthogonal Defect Classification. Towards an in-process measurement system for mobile robot development. Dissertation, technical report CMU-RI-TR-99-05, Robotics Institute, Carnegie Mellon University. Pittsburgh (USA) 1998. (Auch online unter http://www.ri.cmu.edu/pubs/pub_2680.html verfügbar. Stand 07.06.2007)

Slaughter, Harter, Krishnan /Software quality/

Sandra A. Slaughter, Donald E. Harter, Mayuram S. Krishnan : Evaluating the Cost of Software Quality. In: Communications of the ACM. Nr. 8, Jg. 41, 1998, S. 67-73.

Smith, Keil /Reluctance/

H. Jeff Smith, Mark Keil: The reluctance to report bad news on troubled software projects: a theoretical model. In: Information Systems Journal. Nr. 1, Jg. 13, 2003, S. 69-95.

Sommerville /Software Engineering/

Ian Sommerville: Software Engineering. 4. Aufl., Harlow (England) 1992

Sommerville, Sawyer /Requirements Engineering/

Sawyer Sommerville, Pete Sawyer: Requirements Engineering: a good practice guide. Chichester u. a. 1997.

Spillner, Linz /Softwaretest/

Andreas Spillner, Tilo Linz: Basiswissen Softwaretest. Aus- und Weiterbildung zum Certified Tester. Foundation Level nach ASQF- und ISEB-Standard. Heidelberg, 2003.

Staehle /Empirische Analyse/

Wolfgang H. Staehle: Empirische Analyse von Handlungssituationen. In: Richard Köhler (Hrsg.): Empirische und handlungstheoretische Forschungskonzeptionen in der Betriebswirtschaftslehre. Stuttgart 1977, S. 103-115.

Staehle /Management/

Wolfgang H. Staehle: Management. Eine verhaltenswissenschaftliche Perspektive. 8. Aufl., München 1999.

Stahlknecht, Hasenkamp /Wirtschaftsinformatik/

Peter Stahlknecht, Ulrich Hasenkamp: Einführung in die Wirtschaftsinformatik. 9. Aufl., Berlin, Heidelberg, New York 1999.

Stark, Skillicorn, Ameele /Examination/

George Stark, Al Skillicorn, Ryan Ameele: An examination of the effects of requirements changes on software releases. In: CrossTalk, The Journal of Defense Software Engineering. Nr. 12, Jg. 11, 1998, S. 11-16.

Steinke /Qualitative Research/

Ines Steinke: Quality Criteria in Qualitative Research. In: Uwe Flick; Ernst von Kardorff; Ines Steinke (Hrsg.): A Companion to Qualitative Research. London, Thousand Oaks, New Delhi 2004, S. 184-190.

Stelzer /Software-Qualitätsmanagements/

Dirk Stelzer: Möglichkeiten und Grenzen des prozessorientierten Software-Qualitätsmanagements. Habil. Köln, 1998.

Straub, Boudreau, Gefen /Validation guidelines/

Detmar Straub, Marie-Claude Bodreau, David Gefen: Validation guidelines for IS

positivist research. In: Communications of the Association for Information Systems. Nr. 24, Jg. 13, 2004, S. 380-427.

Strauss, Corbin /Grounded theory/

Anselm Strauss, Juliet Corbin: Basics of qualitative research. Techniques and procedures for developing grounded theory. 2. Aufl., Thousand Oaks, London, New Delhi 1998.

Sullivan, Chillarege /Comparison/

Mark Sullivan, Ram Chillarege: A comparison of software defects in database management systems and operating Systems. In: IEEE (Hrsg.): Proceedings of the 22nd IEEE International Symposium on Fault Tolerant Computing (FTCS-22), July 8-10, 1992, Boston (Massachusetts, USA.). Los Alamitos (California, USA) 1992, S. 475-484.

Sutcliffe, Economou, Markis /Requirements errors/

Alistair G. Sutcliffe, A. Economou, P. Markis: Tracing requirements errors to problems in the requirements engineering process. In: Requirements Engineering. Nr. 3, Jg. 4, 1999, S. 134-151.

Sutton, Staw /Theory/

Robert I. Sutton, Barry M. Staw: What Theory is not. In: Administrative Science Quarterly. Nr. 3, Jg. 40, 1995, S. 371-384.

Takahashi u. a. /Methodologies/

Kenji Takahashi, Atsuko Oka, Shuichiro Yamamoto, Sadahiro Isoda: A comparative study of structured and text-oriented analysis and design methodologies. In: Journal of Systems and Software. Nr. 1, Jg. 28, 1995, S. 69-75.

Tassey /Software Testing/

Gregory Tassey (Hrsg.): The Economic Impacts of Inadequate Infrastructure for Software Testing. Planning Report 02-3, National Institute of Standards and Technology (NIST). Gaithersburg (Maryland, USA), 2002.
(Auch online unter http://www.nist.gov/director/prog-ofc/report02-3.pdf verfügbar. Stand 18.08.2004)

Tavolato, Vincena /Prototyping/

Paul Tavolato, Karl Vincena: A Prototyping methodology and its tool. In: R. Budde, K.

Kuhlenkamp, H. Matthiassen, H. Zullinghoven (Hrsg.): Approaches to Prototyping. Berlin 1984, S. 434-446.

The Standish Group /CHAOS/

The Standish Group International, Inc. (Hrsg.): The CHAOS Report (1994). http://www.standishgroup.com/sample_research/PDFpages/chaos1994.pdf, Abruf am 31.08.2004.

Thompson /Organizations/

James D. Thompson: Organizations in action. Social science of administrative theory. New York u. a. 1967.

Tichy /Empirical work/

Walter F. Tichy: Hints for reviewing empirical work in software engineering. In: Empirical Software Engineering. Nr. 4, Jg. 5, 2000, S. 309-312.

Tiwana /Knowledge integration/

Amrit Tiwana: An empirical study of the effect of knowledge integration on software development performance. In: Information and Software Technology. Nr. 13, Jg. 46, 2004, S. 899-906.

Trittmann /Agile Softwareprojekte/

Ralph Trittmann: Wann sind agile Softwareprojekte erfolgreich? Ergebnisse einer empirischen Studie. In: Bernd Oestereich (Hrsg.): Agiles Projektmanagement. Glashütten 2006, S. 69-81.

U.S.-Canada Power System Outage Task Force /Causes/

U.S.-Canada Power System Outage Task Force: Interim Report: Causes of the August 14th blackout in the United States and Canada. http://www.ferc.gov/cust-protect/moi/blackout-report.pdf, Abruf am 2006-05-11.

Vinter u.a. /Prevention/

Otto Vinter, Per-Michael Poulsen, Knud Nissen, J. M. Thomson: The Prevention of errors through experience-driven test effort (PET). Final report, European Systems & Software Initiative Project No. 10438. O. O., 1996.
(Auch online unter http://www.esi.es/VASIE/Reports/All/10438/Download.html verfügbar. Stand 25.05.2004)

Wallmüller /Software-Qualitätssicherung/

Ernest Wallmüller: Software-Qualitätssicherung in der Praxis. München, Wien 1990.

Walz, Elam, Curtis /Softtware design team/

Diane B. Walz, Joyce J. Elam, Bill Curtis: Inside a software design team: knowledge acquisition, sharing, and integration. In: Communications of the ACM. Nr. 10, Jg. 36, 1993, S. 63-77.

Webster, Watson /Literature review/

Jane Webster, Richard T. Watson: Analyzing the past to prepare for the future: writing a literature review. In: MIS Quarterly. Nr. 2, Jg. 26, 2002, S. xiii-xxiii.

Weick, Roberts /Collective mind/

Karl E. Weick, Karlene. H. Roberts: Collective mind in organizations. Heedful interrelating on flight decks. In: Administrative Science Quarterly. Nr. 3, Jg. 38, 1993, S. 357-381.

Weltz, Ortmann /Softwareprojekt/

Friedrich Weltz, Rolf G. Ortmann: Das Softwareprojekt. Projektmanagement in der Praxis. Frankfurt 1992.

Westland /Errors/

J. Christopher Westland: The cost of errors in software development: evidence from industry. In: Journal of Systems and Software. Nr. 1, Jg. 62, 2002, S. 1-9.

Wiegers /Requirements/

Karl E. Wiegers: Software Requirements. Redmond 1999.

Williams, Hall, Kennedy /Framework/

D. W. Williams, T. Hall, M. Kennedy: A Framework for Improving the Requirements Engineering Process Management. In: Software Quality Journal. Nr. 2, Jg. 8, 1999, S. 133-147.

WKWI /Wirtschaftsinformatik/

Wissenschaftliche Kommission Wirtschaftsinformatik im Verband der Hochschullehrer für Betriebswirtschaft e.V. (WKWI): Profil der Wirtschaftsinformatik. In: Wirtschaftsinformatik. Nr. 1, Jg. 36, 1994, S. 80-81.

Wöhe, Döring /Betriebswirtschaftslehre/

Günter Wöhe, Ulrich Döring: Einführung in die Allgemeine Betriebswirtschaftslehre. 18. Aufl., Münschen, Vahlen 1993

Yin /Case study crisis/

Robert K. Yin: The case study crisis. Some Answers. In: Administrative Science Quarterly. Nr. 1, Jg. 26, 1981, S. 58-65.

Yin /Case study research/

Robert K. Yin: Case study research. Design and methods. 3. Aufl., Thousand Oaks, London, New Delhi 2003.

Zelkowitz, Wallace /Models/

Marvin V. Zelkowitz, Dolores R. Wallace: Experimental models for validating technology. In: Computer. Nr. 5, Jg. 31, 1998, S. 23-31.

Zheng u. a. /Static analysis/

Jiang Zheng, Laurie Williams, Nachiappan Nagappan, Will Snipes, John P. Hudepohl, Mladen A. Vouk: On the value of static analysis for fault detection in software. In: IEEE Transactions on Software Engineering. Nr. 4, Jg. 32, 2006, S. 240-253.

Zuse /Software measurement /

Horst Zuse: A framework of software measurement. Berlin, New York 1998.