# Verknüpfung von Anforderungen und Systemausführungen mithilfe einer Multilevel Markup Language

Inaugural-Dissertation
zur
Erlangung des Doktorgrades

Doktor der Naturwissenschaft
— doctor rerum naturalium (Dr. rer. nat.) —

der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität zu Köln

vorgelegt von
Florian Pudlitz, M.Sc.
geb. in Lübben

Vorsitzender:
Prof. Dr.-Ing. Gregor Gassner

Prüfer der Dissertation:
Prof. Dr. rer. nat. Andreas Vogelsang
Prof. Dr. rer. nat. Martin Glinz

Tag der wissenschaftlichen Aussprache:
22.07.2021

Köln 2021

# VERÖFFENTLICHUNGEN

Die in dieser Dissertation vorgestellte Arbeit entstand vorrangig an der Technischen Universität Berlin und wurde an der Universität zu Köln beendet. Einige Ideen und Ergebnisse sind bereits in den aufgelisteten Veröffentlichungen erschienen:

### KONFERENZBEITRÄGE:

**F. Pudlitz**, A. Vogelsang, F. Brokhausen: *A Lightweight Multilevel Markup Language for Connecting Software Requirements and Simulations*, Requirements Engineering: Foundation for Software Quality. 25th International Working Conference (REFSQ19), März 2019, Essen, Deutschland, ISBN 978-3-030-15538-4, http://dx.doi.org/10.14279/depositonce-8300

**F. Pudlitz**, F. Brokhausen, A. Vogelsang: *Extraction of System States from Natural Language Requirements*, 27th IEEE International Requirements Engineering Conference (RE'19), 2019, Jeju Island, Südkorea, ISBN 978-1-7281-3912-8, http://dx.doi.org/10.14279/depositonce-8717

### JOURNALBEITRÄGE:

E. Cioroaica, **F. Pudlitz**, I. Gerostathopoulos, T. Kuhn: *Simulation methods and tools for collaborative embedded systems: with focus on the automotive smart ecosystems*, SICS Software-Intensive Cyber-Physical Systems, Vol. 34, Nr. 4, November 2019, ISSN 2524-8510, http://dx.doi.org/10.14279/depositonce-9461

**F. Pudlitz**, F. Brokhausen, A. Vogelsang: *What Am I Testing and Where? Comparing Testing Procedures based on Lightweight Requirements Annotations*, Empirical Software Engineering, Vol. 25, Nr. 4, Mai 2020, ISSN 1382-3256, http://dx.doi.org/10.14279/depositonce-9856.2

# KURZFASSUNG

Die Softwareentwicklung in nahezu allen Bereichen basiert auf Software- und Testspezifikationen. Die stetig wachsende Komplexität von Softwarefunktionen führt zu einem wachsenden Anforderungs- und Testmanagement. Daraus resultiert eine Vielzahl von Herausforderungen um zu gewährleisten, dass durchgeführte Tests alle zuvor gestellten Anforderungen erfüllen. Der Einsatz von natürlichsprachlichen Anforderungen ermöglicht eine uneingeschränkte Spezifizierung der Software, erschwert jedoch die automatisierte Weiterverarbeitung für den Entwicklungs- und Testprozess, da die Vielfalt der Sprache nur unzureichend von heutigen Algorithmen verarbeitet werden kann. Manuelle Vorgehensweisen sind daher bis heute unersetzbar. Es fehlen einheitliche Methoden um natürlichsprachliche Anforderungen mit Testdurchläufen zu verknüpfen und Testergebnisse im Kontext der Anforderungen zu beurteilen.

In dieser Arbeit präsentiere ich einen Testansatz, der es ermöglicht Informationen mithilfe von Textmarkierungen aus natürlichsprachlichen Anforderungen zu extrahieren und für Tests weiterzuverarbeiten. Die Testergebnisse werden anschließend in den Anforderungen angezeigt und können von Anwender:innen beurteilt werden. Dadurch werden durchgeführte Tests direkt mit Anforderungen verknüpft und können speziell für das zu testende System analysiert werden.

Der erste Beitrag dieser Arbeit umfasst die Entwicklung einer Markierungssprache, die es den Anwender:innen ermöglicht, gezielt Texte in Anforderungsdokumenten zu markieren. Die Sprache besteht aus vier unterschiedlichen Detailstufen, die jeweils in differenzierteren Auswertungen resultieren. Die markierten Textpassagen bilden den Ausgangspunkt für alle weiteren Schritte des Testansatzes.

Der zweite Beitrag vergleicht zwei Vorgehensweisen zur Entwicklung einer automatisierten Erkennung von Systemzuständen in natürlichsprachlichen Anforderungen. Der resultierende Algorithmus kann die Anwender:innen beim Markieren von Textpassagen unterstützen und basiert auf Methoden des maschinellen Lernens. Dadurch können beliebige Softwarespezifikationen in Betracht gezogen werden.

Als dritten Beitrag präsentiere ich eine umfassende Analysemethode des Softwareverhaltens. Hierbei basieren die Auswertungen auf den markierten Textstellen der Anwender:innen und werden in den Softwarespezifikationen angezeigt. Die Anwender:innen haben die Möglichkeit die Testresultate im Kontext der Anforderungen zu beurteilen. Die Analysemethode erlaubt eine automatisierte parallele Auswertung von mehreren Testdurchläufen und einen Vergleich zweier unterschiedlicher Teststufen.

Der vierte Beitrag untersucht die Nutzbarkeit der entwickelten Markierungssprache anhand einer Interviewstudie mit Probanden aus der Praxis. Hierbei testeten die Probanden den Testansatz und beurteilten ihn. Die Ergebnisse wurden anschließend ausgewertet und in den Vergleich zu heutigen Praktiken gesetzt.

## ABSTRACT

Software development in almost all areas is based on software and test specifications. The ever-increasing complexity of software functions leads to growing requirements and test management. It results in a multitude of challenges to ensure that executed tests meet all previously specified requirements. The use of natural language requirements enables an unrestricted possibility of specifying the software, but complicates the automated further processing for the development and test process, since the variety of the language can be processed only insufficiently by today's algorithms. Manual procedures are therefore irreplaceable to this point. There is a lack of uniform methods to link natural language requirements with test executions and to evaluate test results in the context of the requirements.

In this paper, I present a testing approach that allows information to be extracted from natural language requirements using text marks and then beeing processed for testing. The test results are then displayed in the requirements and can be assessed by the user. Thus, tests performed are directly linked to requirements and can be analyzed specifically for the system under test.

The first contribution of this work involves the development of a markup language that allows the user to selectively mark up text in requirements documents. The language consists of four different levels of detail each resulting in more detailed analyses. The marked text passages form the starting point for all further steps of the test approach.

The second contribution compares two approaches for developing an automated recognition of system states in natural language requirements. The resulting algorithm can support the user in marking text passages and is based on machine learning methods. As a result, arbitrary software specifications can be considered.

As a third contribution, I present a comprehensive analysis method of software behavior. Here, the evaluations are based on the user's marked text passages and are displayed in the software specifications. The user has the possibility to evaluate the test results in the context of the requirements. The analysis method allows an automated parallel evaluation of several test runs and a comparison of two different test levels.

Our fourth contribution examines the usability of the developed marking language by means of an interview study with participants from the industry. Here, the subjects were able to test and assess the different aspects of the testing approach. Subsequently, we analyzed the results and discussed them in comparison to current practices.

# DANKSAGUNG

Mein besonderer Dank gilt Prof. Dr. Andreas Vogelsang, der mich von Anfang an unterstützte, mir während des gesamten Entstehungsprozesses zur Seite stand und mich in vielen Gesprächen inspirierte.

Herzlich danken möchte ich auch Prof. Dr. Glinz, den ich auf meiner ersten Konferenz als einen offenen und angenehmen Gesprächspartner kennenlernte. Auf weiteren Konferenzen hat mich der Austausch mit ihm in meiner Arbeit bestärkt und motiviert.

Danken möchte ich auch meinen ehemaligen Kollegen des Fachgebiets; besonders Florian Brokhausen hat eng mit mir zusammengearbeitet und mich in vielen Teilaspekten meiner Arbeit unterstützt.

Diese Arbeit wäre ohne die jahrelange Unterstützung meiner Familie, insbesondere meiner Frau Vera, nicht möglich gewesen. Danken möchte ich ihr für die vielen anregenden Gespräche, ihr großes Interesse für mein Thema und vor allem für ihre Geduld und ihr Verständnis für die teils stressige Zeit vor den Deadlines.

# INHALTSVERZEICHNIS

# 1 EINLEITUNG

Das Thema der Arbeit ist ein Testansatz für Softwarefunktionen basierend auf einer Multilevel Markup Language, die natürlichsprachliche Softwarespezifikationen und Testdurchläufe miteinander verbindet. In diesem Kapitel wird das Thema eingeführt und der Testansatz für komplexe Softwaresysteme motiviert (Kapitel 1.1). Anschließend folgt die Problembeschreibung (Kapitel 1.2) bezogen auf bisherige Testansätze. In Kapitel 1.3 erläutere ich die generelle Funktionsweise des Testansatzes und fasse die wichtigsten Beiträge dieser Arbeit in Kapitel 1.4 zusammen, bevor in Kapitel 1.5 die Gliederung der Arbeit folgt.

## 1.1 KONTEXT

In vielen Bereichen werden Softwaresysteme durch den Einsatz offener Systeme, hoch automatisierter oder vernetzter Geräte immer komplexer. Die Komplexität führt zu einer steigenden Anzahl von Anforderungen, die oft in natürlicher Sprache ausgedrückt werden [53, 67]. In mehreren Bereichen schreiben Entwicklungsstandards die Angleichung von Anforderungen und Testverfahren vor um sicherzustellen, dass softwareintensive Systeme effektiv getestet werden (vgl. ISO26262 ([50]) für Automobilsysteme, EN 50128 ([21]) für Eisenbahnsysteme oder DO-178C ([77]) für Avioniksysteme).

Softwareintensive Systeme werden auf verschiedenen Ebenen getestet um einen schrittweisen Testprozess zu ermöglichen und das Vertrauen in die Funktionalität des Systems zu erhöhen. In den letzten Jahren hat sich das V-Modell als Vorgehensmodell für die Softwareentwicklung etabliert [14]. Es beschreibt die fortschreitenden Phasen der Entwicklung zusammen mit den zugehörigen Qualitätssicherungs- und Testmaßnahmen. Die empfohlenen Testphasen reichen von Unit- und Komponententests über den Test von Subsystemen bis hin zum Integrationstest. Zur Validierung und Verifikation werden insbesondere Techniken wie Software-in-the-Loop (SiL), Hardware-in-the-Loop (HiL) und Human-in-the-Loop-Tests eingesetzt. Jede Testphase hat ihre eigenen Vor- und Nachteile und Einschränkungen. SiL-Tests sind beispielsweise günstig, flexibel und leicht skalierbar. Andererseits stützen sie sich stark auf realistische Umgebungsmodelle und berücksichtigen in der Regel keine Auswirkungen, die mit dem Verhalten von Hardware-Komponenten zusammenhängen. Human-in-the-Loop-Tests hingegen sind am realistischsten, aber teuer und schwer zu skalieren.

Ein Softwaresystem, das das System-under-Test (SuT) mit spezifischen Eingaben stimuliert, es ausführt und die resultierenden Ausgaben auf Modelle der voraussichtlichen Umgebung des Systems abbildet, wird als Simulator bezeichnet. Simulatoren sind in den letzten Jahren immer komplexer geworden, um das Systemverhalten in vielfältigen Umgebungen zu antizipieren. Die Verwendung dieser Simulatoren hat mehrere Vorteile gegenüber konventionellen Softwaretests wie Unit- oder Komponententests. Konventionelle Tests liefern nur ein binäres Testergebnis (erfüllt oder nicht erfüllt) und enthalten nur wenige Informationen über die kontextuelle Situation. Simulatoren sind flexibler, um verschiedene Variationen des Kontextverhaltens abzudecken. Simulationen bieten auch ein hohes Maß an Anpassungsfähigkeit hinsichtlich der Abstraktion der Umgebung oder des Kontextes des Systems, weshalb sie in allen Phasen der Entwicklung des SuT eingesetzt werden können. Zusätzlich bieten Simulationen im Vergleich zu statischen Softwaretests wie Unit-Tests die Möglichkeit, zeitabhängige Analysen durchzuführen. Durch die zeitkomprimierende Ausführung einer Simulation ist es möglich, das Verhalten eines SuT in seinem Kontext über Stunden oder Tage hinweg in wesentlich kürzerer Zeit zu untersuchen.

Der Einsatz sowohl von Simulationen als auch von bisherigen Teststrategien stellt die Ingenieur:innen im Abgleich mit gestellten Anforderungen speziell für komplexe Softwaresysteme vor große Herausforderungen.

## 1.2 PROBLEMBESCHREIBUNG

In der gegenwärtigen Praxis und insbesondere in großen Unternehmen stellt die Abstimmung von Test- und Anforderungsaktivitäten eine Herausforderung dar [13, 32]. Simulationsszenarien werden oft nicht direkt aus Anforderungen abgeleitet, sondern von spezialisierten Ingenieur:innen auf der Grundlage ihrer eigenen Fachkenntnisse der Problemdomäne handgefertigt [46]. Außerdem werden die Ergebnisse von Simulationsläufen oft nicht auf die Ebene der Anforderungen zurückgeführt. Somit können die Anforderungsingenieur:innen nicht von den durch die Durchführung der Simulation gewonnenen Erkenntnissen profitieren. Dieser Versatz hat mehrere Gründe: Erstens wird das Requirements Engineering und das Testmanagement oft in verschiedenen Abteilungen durchgeführt. Zweitens handelt es sich bei Simulatoren um komplexe Systeme, die von Simulationsexperten konfiguriert werden müssen. Dies macht es den Anforderungsingenieur:innen schwer sie anzuwenden. Drittens befinden sich Anforderungen und Simulationen auf unterschiedlichen Abstraktionsebenen, was erschwert, die von der Simulation erzeugten Ereignisse mit den Anforderungen in Verbindung zu bringen - insbesondere, wenn sie in natürlicher Sprache geschrieben sind. Zudem sind Simulationsszenari-

en oft unrealistisch und stellen nicht sicher, dass alle Anforderungen abgedeckt und verifiziert werden. Eine mögliche Methode diesen Problemen zu begegnen ist die Modellierung von Anforderungen. Sie kann helfen, die Lücke zwischen Anforderungen und Simulation zu schließen. Wenn die erforderlichen Modelle jedoch zu formal sind, zögern die Anforderungsingenieur:innen, sich um die Modellierung der Anforderungen zu bemühen.

Die in dieser Arbeit vorgestellte Multilevel Markup Language sowie der zugehörige Ansatz, diese Sprache mit Testausführungen zu verbinden, bieten den Anforderungsingenieur:innen ein Werkzeug, um spezifiziertes Systemverhalten besser zu beobachten. Sie erhalten umfassendes Feedback zur Verifizierung der Anforderungen. Testingenieur:innen haben die Möglichkeit, Testszenarien besser zu koordinieren und sie spezifischer auf die Anforderungen hin zu entwickeln. Dies führt zu einer Harmonisierung beider Seiten und fördert den Austausch, das Verständnis und die Zusammenarbeit in allen Bereichen des Entwicklungsprozesses.

## 1.3 LÖSUNGSIDEE UND HERAUSFORDERUNGEN

Für die Verknüpfung von natürlichsprachlichen Softwarespezifikationen mit Testdurchläufen von Softwarefunktionen werden derzeit unterschiedliche Zwischenschritte unternommen. Zu Beginn werden wichtige Informationen, die für Testdurchläufe notwendig sind, überwiegend aus den Anforderungen manuell herausgelesen. Diese Informationen nutzen die Testingenieur:innen in einem nächsten Schritt für die Erstellung von Testspezifikationen. Die Testspezifikationen umfassen eine Vielzahl von durchzuführenden Tests. Zusätzlich sind Informationen beispielsweise zu Vorbedingungen, Ablauf, erwartetem Ergebnis und Testauswertung enthalten. Bei wachsender Komplexität der Softwaresysteme wächst auch die Anzahl der Anforderungen. Daraus resultiert eine steigende Anzahl von Softwaretest mit umfassenderen Testspezifikationen. Die manuelle Erstellung stößt hier an die Grenzen der Realisierbarkeit. Zwar widmet sich ein eigenes Forschungsfeld der automatisierten Erstellung von Softwaretest [2, 31, 71, 110], dennoch können die erforschten Methoden nur bedingt mit dem Einsatz von natürlichsprachlichen Softwarespezifikationen umgehen. Daraus folgt, dass automatisierte Methoden die vollständige Erfüllung der Spezifikation nicht ausreichend gewährleisten können. Automatisierte Ansätze unterstützen die Anwender:innen, sodass die Erstellung von Softwaretests zunehmend semi-automatisiert abläuft.

Nach dem Durchlauf der spezifizierten Tests können die Testingenieur:innen die Erfüllung der Tests beurteilen. Die Rückverfolgung vom Testergebnis zu den Anforderung ist eine weitere Herausforderung die derzeit toolgestützt umgesetzt wird. Hierbei sind die digitalen Anforderungen und Tests miteinander verknüpft. So kann ein Test

mehrere Anforderungen adressieren oder eine Anforderungen von mehreren Tests abgedeckt werden. In komplexen Softwareprojekten wächst die Anzahl an Verknüpfungen enorm an. Daraus resultiert, dass die Erfüllung von Tests im Vordergrund der Teststrategien stehen und nur in seltenen Fällen eine Rückverfolgung zu den Anforderungen stattfindet.

Mit diesem Hintergrundwissen entwickelten wir einen vierteiligen Ansatz, der Informationsextraktion, Signalabbildung, Datenauswertung und Ergebnisdarstellung umfasst. Kern der Arbeit ist eine Markierungssprache, mithilfe derer die Ingenieur:innen testrelevante Informationen in Softwarespezifikationen - den Anforderungen - markieren und aus diesen extrahieren können. In der Signalabbildung werden die natürlichsprachlichen Textelemente auf konkrete Signale des Testsystems abgebildet. Während einer Testdurchführung wird das Softwareverhalten aufgezeichnet und im Anschluss ausgewertet. Die Auswertungen werden mit den anfänglich gestellten Softwarespezifikationen verknüpft und ermöglichen den Ingenieur:innen eine Bewertung des Testdurchlaufs im Kontext der Softwarespezifikation. Hierdurch entstehen zwei Auswertungsmöglichkeiten. Erstens können Testergebnisse zu markierten Textstellen unmittelbar in den Spezifikationen angezeigt werden. Zweitens stehen Auswertungen bezogen auf die gesamte Softwarespezifikation zur Verfügung. Hierdurch lässt sich beispielsweise ableiten, in welchem Umfang die Softwarespezifikation erfüllt ist. Dieser Ansatz verbindet die Anforderungen mit den Tests und reduziert den manuellen Aufwand bei der Erstellung von Testspezifikationen.

## 1.4 BEITRÄGE DER ARBEIT

Abbildung 1.1 zeigt die vier Hauptbeiträge dieser Arbeit.

Beitrag A beschreibt die Entwicklung einer Markierungssprache mit derer Elementen beliebige Textpassagen in natürlichsprachlichen Softwarespezifikationen markiert werden. Der Einsatz dieser Sprache dient zur Extraktion von Informationen, die für den weiteren Testprozess relevant sind. Um eine möglichst hohe Nutzbarkeit zu erreichen, liegt der Fokus auf einem leichtgewichtigen aber dennoch flexiblen Ansatz, der das manuelle Lesen ergänzt. Die Markierungssprache besteht aus vier Levels, die jeweils unterschiedliche Detailstufen abbilden. Die Sprache umfasst Markierungselemente, die den Textpassagen zugeordnet werden. Dieser manuelle Ansatz ermöglicht einen flexiblen Einsatz, unabhängig von der Struktur, enthaltenen Fehlern oder vom Abstraktionsgrad der Softwarespezifikation.

Beitrag B beschreibt eine mögliche Unterstützung dieses Markierungsprozesses. Wir präsentieren eine automatisierte Erkennung von spezifischen Elementen der Markierungssprache basierend auf Methoden des maschinellen Lernens. Hierfür verglichen wir zwei Vor-

| Kapitel | Name | Hauptbeitrag |
|---|---|---|
| Kapitel 3 | Anforderungsbasierter Testprozess | |
| | 3.2 Multilevel Markup Language | A |
| Kapitel 4 | Teil 1: Annotationsprozess | |
| | 4.2 Studie zur automatisierten Extraktion von Elementen | B |
| Kapitel 5 | Teil 2: Abbildungsprozess | |
| Kapitel 6 | Teil 3: Auswertungsprozess | |
| | 6.2 Studie zur automatisierten Logdatenanalyse | C |
| Kapitel 7 | Teil 4: Darstellungsprozess | |
| | 7.2 Studie zur Nutzbarkeit von Annotationen | D |

Abbildung 1.1: Hauptbeiträge dieser Arbeit im Überblick

gehensmodelle mit unterschiedlichem manuellen Aufwand, die zur Entwicklung des Algorithmus führen. Um die Ergebnisse des Vorgehensmodells und des resultierenden Algorithmus zu evaluieren, führten wir eine Studie mit realen Anforderungsdokumenten durch und diskutieren die Ergebnisse bei der Erkennung von Systemzuständen.

Beitrag C befasst sich mit der Auswertung von Testdurchläufen. Der neuartige Ansatz basiert auf den Markierungselementen des Beitrags A und ermöglicht eine umfassende vierteilige Analyse des Systemverhaltens und des eingesetzten Testverfahrens. Zudem besteht die Möglichkeit verschiedene Testverfahren miteinander zu vergleichen. Alle Auswertungen sind stets bezogen auf die gestellte Softwarespezifikation und ermöglicht Aussagen über das Systemverhalten in spezifischen Testdurchläufen.

Beitrag D untersucht die Nutzbarkeit der Textannotationen und die praktische Relevanz des entwickelten Testansatzes. Hierfür wurde eine Interviewstudie mit Probanden aus der Industrie durchgeführt. Die Teilnehmer wendeten die verschiedenen Aspekte an einem Beispiel an und beurteilten die Eigenschaften des Ansatzes. Die Ergebnisse lassen Rückschlüsse auf die Vor- und Nachteile des Testansatzes zu und erlauben die Einordnung im Vergleich zu heutigen Praktiken.

## 1.5 AUFBAU DER ARBEIT

Die vorliegende Arbeit hat acht Kapitel. Die Hauptbeiträge sind in Kapitel 1.4 erläutert und grafisch den Methodenkapiteln dieser Arbeit zugeordnet.

In Kapitel 2 führe ich grundlegende Begriffe und Methoden ein, die für das weitere Verständnis der Arbeit notwendig sind. In Abschnitt 2.1 beschreibe ich das grundsätzliche Vorgehen bei der Erhebung von Soft-

warespezifikationen. Diese Spezifikationen sind der Ausgangspunkt für die Softwareentwicklung, die im Anschluss in verschiedenen Teststufen validiert werden. In Kapitel 2.2 zeige ich etablierte Testansätze und legen den Fokus auf die Herausforderung in Testphasen zukünftiger, wachsender Softwarefunktionen. In Kapitel 2.3 beschreibe ich die wichtigsten Methoden zur Verknüpfung von Spezifikationen und Systemausführungen. Verwandte Arbeiten die ähnliche Ziele verfolgen, sind in Kapitel 2.4 dargestellt.

Kapitel 3 widmet sich dem entwickelten anforderungsbasierten Testprozess und gibt im Abschnitt 3.1 einen Überblick über die Gesamtmethode. Der Testprozess basierte auf einer Markierungssprache, die mehrere Elemente zum Markieren von Textpassagen enthält. Abschnitt 3.2 erläutert die Sprachelemente, deren Einsatz- und Auswertungsmöglichkeiten. Da der Testansatz manuelle Komponenten enthält und der Einsatz Systemkenntnisse voraussetzt, beschreibe ich in Abschnitt 3.3 das notwendige Domänenwissen und erläutern den Einfluss auf den Testansatz.

Kapitel 4 widmet sich dem ersten von vier Prozessschritten, dem Annotationsprozess. In Abschnitt 4.1 beschreibe ich das manuelle Vorgehen beim Markieren von Textpassagen und erläutere wichtige Kerneingenschaften der Markierungssprache. Im folgenden Abschnitt 4.2 zeige ich das Potenzial einer automatisierten Unterstützung im Annotationsprozess anhand einer durchgeführten Studie basierend auf realen Softwarespezifikationen.

In Kapitel 5 stelle ich den Abbildungsprozess vor. Dies ist der zweite Prozessschritt des Testansatzes. Hierbei liegt der Fokus in Abschnitt 5.1 auf der Verknüpfung von natürlichsprachlichen Textpassagen und Signalnamen aus Testausführungen. Abschnitt 5.2 beschreibt anschließend, wie Tests durchgeführt werden und welche Möglichkeiten der Aufzeichnung des Softwareverhaltens bestehen.

Kapitel 6 beschreibt den dritten der vier Prozessschritte und zeigt die Auswertungsmöglichkeiten von Systemverhalten bezogen auf die entwickelte Markierungssprache. Abschnitt 6.1 zeigt zunächst, wie Aufzeichnungen des Systemverhaltens interpretiert werden. Anschließend zeigt die Studie in Abschnitt 6.2 die möglichen Auswertungs- und Vergleichsmöglichkeiten von zwei realen Testdurchläufen unter Berücksichtigung der Softwarespezifikation.

Das Kapitel 7 beschreibt den vierten Prozessschritt, den Darstellungsprozess. Abschnitt 7.1 zeigt, wie die resultierenden Ergebnisse aus dem Auswertungsprozess visualisiert werden. Der folgende Abschnitt 7.2 erläutert die durchgeführte Studie zur Nutzbarkeit der entwickelten Markierungssprache mit Fokus auf die Darstellungsformen.

In Kapitel 8 fasse ich die Ergebnisse der Arbeit zusammen und geben einen Ausblick auf weiterführende Arbeiten.

# 2 GRUNDLAGEN UND VERWANDTE ARBEITEN

Dieses Kapitel erläutert die Kernthemen der Arbeit, stellt bisherige Forschungsarbeiten vor und erklärt grundlegende Themen, die für das bessere Verständnis der Arbeit nützlich sind. Kapitel 2.1 definiert den Begriff Requirements Engineering und erläutert die Grundlagen zur Erhebung von Softwarespezifikationen. Diese sind Ausgangspunkt für die Entwicklung der Systeme, die im Anschluss verschiedene Testphasen durchlaufen. Kapitel 2.2 beschreibt die verschiedenen Phasen heutiger Vorgehensmodelle und erläutert insbesondere zukünftige Methoden für das Testen von eingebetteten Systemen. Für die Verknüpfung von Spezifikationen und Systemausführungen mithilfe von Tests werden verschiedenen Ansätze in Kapitel 2.3 vorgestellt und bezogen auf heutige Herausforderungen bewertet. In Kapitel 2.4 stellen wir verwandte Arbeiten vor, die eine inhaltliche Nähe zum hier präsentierten Testansatz aufweisen.

## 2.1 SOFTWARESPEZIFIKATION

Das Requirements Engineering umschreibt die Erstellung und Verwaltung von Spezifikationen und wird vom International Requirements Engineering Board wie folgt definiert [34]:

> Die systematische und disziplinierte Vorgehensweise, bei der Spezifikation und Verwaltung von Anforderungen mit dem Ziel, die Wünsche und Bedürfnisse der Stakeholder zu verstehen und das Risiko zu minimieren, ein System zu liefern, das diese Wünsche und Bedürfnisse nicht erfüllt.

Das Requirements Engineering ist die Basis für alle darauffolgenden Entwicklungsphasen der Softwareentwicklung [27]. Daher ist ein qualitatives Requirements Engineering bedeutend für ein erfolgreiches Entwicklungsprojekt und ist gleichzeitig eine der größten Herausforderungen [3, 47].

Ausgangspunkt sind formulierte Softwarespezifikationen, respektive Anforderungen innerhalb des Requirements Engineerings, die das Verhalten des zu entwickelnden Systems möglichst gut abbilden. Diese werden ebenfalls vom International Requirements Engineering Board definiert [34]:

> Eine systematisch dargestellte Sammlung von Anforderungen, typischerweise für ein System oder eine Komponente, die bestimmte Kriterien erfüllt.

> In manchen Situationen wird zwischen einem Kundenlastenheft (typischerweise vom Kunden geschrieben) und einem Systemlastenheft oder Softwarelastenheft (vom Lieferanten geschrieben) unterschieden.
>
> Anforderungsspezifikation kann auch die Aktivität der Spezifikation (Erhebung, Dokumentation und Validierung) von Anforderungen bezeichnen.

Wichtige Merkmale bei der Erstellung von Spezifikationen sind unter anderem *Korrektheit*, *Eindeutigkeit*, *Nachprüfbarkeit* und *Rückverfolgbarkeit* [49]. Um diesen Kriterien gerecht zu werden, werden komplexe Softwareanforderungen in viele einzelne Anforderungen aufgeteilt, um sie besser den Softwarefunktionalitäten zuzuordnen. Diese systematische Darstellung wird von zusätzlichen Informationen wie Erklärungen oder Abbildungen begleitet. Tabelle 2.1 zeigt eine mögliche strukturierte Darstellungsform, die alle wichtigen Aspekte für den vorgestellten Testansatz enthält.

Tabelle 2.1: Exemplarische Darstellung einer Systemspezifikation

| **ID** | **Text** | **Typ** |
|---|---|---|
| R1 | **Kapitel** | Überschrift |
| R2 | Anforderung 1 | Requirement |
| R3 | Anforderung 2 | Requirement |
| R4 | Information 1 | Information |
| R5 | Anforderung 3 | Requirement |
| R6 | Information 2 | Information |
| R7 | Anforderung 4 | Requirement |

Die *ID* gibt einen eindeutigen Bezeichner an und verhindert doppelte Einträge innerhalb des Requirements Engineering Prozesses. Die Spalte *Text* enthält alle Kapitelüberschriften, Anforderungstexte und zusätzlichen Informationen. Mithilfe der Spalte *Typ* werden die Texte einem eindeutigen Typen zugeordnet. Das ermöglicht eine Unterscheidung zwischen geforderten umsetzbaren Anforderungen (*Requirement*) und zusätzlichen Informationen (*Information*), welche die Verständlichkeit erhöhen sollen. Zur besseren Strukturierung werden Kapitelüberschriften eingesetzt (*Überschrift*), die eine besseren Orientierung in umfangreichen Dokumenten ermöglicht. In realen Softwarespezifikationen sind weitere Einträge möglich, beispielsweise *s-send* und *s-receive* für aus- und eingehende Signale oder *Level* für die Angabe der hierarchischen Struktur des Dokuments. Für den entwickelten Testansatz stehen diese Elemente jedoch nicht im Vordergrund.

Um die unterschiedlichen Requirements Engineering Aufgaben zu bewältigen, ist eine Toolunterstützung notwendig [85]. Da diese Eigenschaften implementiert werden müssen, hat sich eine vielfältige Toollandschaft im Requirements Engineering entwickelt [33, 51]. Die Fallstudien dieser Arbeit basieren auf den Softwarespezifikationen, die mit dem Tool IBM DOORS verwaltet wurden [48]. Der vorgestellte Ansatz ist nicht auf die Datenbasis dieses Tools begrenzt und setzt lediglich eine textuelle Darstellung der Spezifikation voraus.

### 2.1.1 *Natürlichsprachliche Anforderungen*

Eine verbreitete Methode, um Softwarespezifikationen oder Anforderungen zu erheben, sind natürlichsprachliche Texte [53, 67, 95]. Die Erstellung von natürlichsprachlichen Anforderungen erfordert nur wenig initialen Aufwand und zeichnet sich als eine besonders flexible und vielseitige Erhebungsmethode aus, da die volle Sprachvielfalt zur Verfügung steht [72]. Der Einsatz von natürlicher Sprache ist jedoch problembehaftet und geht mit großen Anstrengungen, qualitativ hohe Anforderungen zu formulieren, einher [12]. Dennoch sind die Resultate oft nicht zufriedenstellend [87].

Es hat sich ein Forschungsfeld etabliert mit dem Ziel die Verarbeitung natürlichsprachlicher Softwarespezifikationen zu verbessern [29]. Das beinhaltet unter anderem die automatisierte Analyse von natürlichsprachlichen Anforderungen, um Schwachstellen zu erkennen [10, 28, 30, 58, 102, 103]. Des Weiteren umfasst es Methoden, um die Anforderungen automatisiert weiterzuverarbeiten [1, 4, 23, 45].

Um diese Aufgaben des Requirements Engineering stetig zu verbessern, fokussiert sich ein Teil der Arbeiten auf die Erforschung und Weiterentwicklung der zugrundeliegenden Techniken und wird im Bereich Natural language processing (NLP) zusammengefasst. Dieses Feld umfasst computergestützte Techniken zum Zweck des Lernens, Verstehens und der Produktion menschlicher Sprachinhalte [41]. Die Arbeit von Liddy definiert NLP wie folgt [61]:

> Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications [61].

Dabei werden drei Typen in der Entwicklung der Technologien unterschieden:

erstens die Entwicklung von Techniken zur praktischen Durchführung von NLP-Aufgaben, wie beispielsweise Part-of-speech Tagging, Lemmatisierung oder Tokenization [60, 65];

zweitens Softwaresysteme, die eine oder mehrere Techniken verknüpfen, beispielsweise Stanford CoreNLP[1], NLTK[2] oder OpenNLP[3];

drittens die Erstellung von Datensätzen zur Unterstützung der NLP Techniken wie WordNet[4] oder FrameNet[5].

Ein zweiter Forschungszweig ist die Analyse von natürlicher Sprache mithilfe von Methoden des maschinellen Lernens. Grundlage des Ansatzes ist es, einem Algorithmus mithilfe vieler Rohdaten die Erkennung oder Klassifizierung von Merkmalen anzutrainieren [60]. Dennoch erfordern diese Ansätze manuellen Einsatz, entweder während des Trainingsprozesses oder zur anschließenden Nachbesserung initial unzureichender Ergebnisse.

Maschinelle Lernalgorithmen werden bei der Analyse natürlicher Sprache speziell für Softwarespezifikationen eingesetzt und dienen zur Unterstützung ganzer Requirements Engineerig Prozesse [56]. Insbesondere die Klassifizierung von Spezifikationen ist Teil umfassender Forschung, beispielsweise bei der Unterscheidung zwischen funktionalen und nicht-funktionalen Anforderungen [15, 22, 108]. Auch die automatisierte Unterscheidung zwischen Anforderungen und Informationen unterstützt die Erhebung und Qualitätssicherung in der Anforderungserhebung [98].

Die maschinelle Erkennung, beziehungsweise Extraktion von bestimmten Kategorien unterstützt die nachfolgenden Schritte in der Softwareentwicklung und wird als Named-Entity Recognition (NER) bezeichnet. Diese Kategorien müssen vordefiniert sein. Seit die Conference on Natural Language Learning 2003 (CoNLL2003) Aufgaben zur sprachunabhängigen NER verteilt, gibt es viele Implementierungen dieser Methode mit maschinellen Lernalgorithmen. Konventionell, wie in der CoNLL2003-Aufgabe vordefiniert, sind die zu erkennenden Merkmale Personen, Organisationen, Orte oder Ähnliches. Bei der Bewältigung dieser Aufgabe werden vielversprechende Ergebnisse geliefert und durch intensive Forschung signifikante Leistungsverbesserungen erzielt [16, 20, 59, 105].

Die Übertragung der Ansätze auf die Anforderungsdomäne ist für viele Bereiche relevant [54]. Da Software oft zustandsbasiert arbeitet, können die extrahierten Terme in anschließenden Entwicklungsschritten für die automatisierte Erstellung von Zustandsautomaten nützlich sein [83, 96, 106]. Gleichzeitig wird die Dokumentation im Requirements Engineering, durch die Erkennung von Glossarbegriffen, verbessert [24].

Eine besondere Herausforderung für den Einsatz von NER-Algorithmen ist die Auswahl der Trainingsdaten. Bootstrapping-Ansätze wurden ursprünglich als Methode zur Extraktion von Begriffen durch

1 **https://nlp.stanford.edu/software/**.
2 **https://www.nltk.org**.
3 **https://opennlp.apache.org**.
4 **https://wordnet.princeton.edu**.
5 **https://framenet.icsi.berkeley.edu/fndrupal/**.

die Erkennung von Mustern verwendet [40, 80, 89]. Diese Methode lässt sich darüber hinaus zur automatisierten Markierung von Textmerkmalen nutzen [57, 88]. Diese Daten werden dann für die NER verwendet. Die Ungenauigkeit des Bootstrappers verringert die Qualität des NER-Algorithmus, reduziert jedoch den manuellen Aufwand bei der Erstellung domänenspezifischer Trainingsdaten [93, 100].

Das Ziel der Forschung sowohl im Bereich NLP als auch im Bereich maschinellen Lernens ist eine menschenähnliche Analyseleistung, die im Bereich des Requirements Engineering die manuellen Prozesse des Menschen unterstützt [61]. Der Mensch als Teil des Requirements Engineering ist nicht gänzlich ersetzbar, wird aber durch die heutige Rechnerleistung in vielen datenintensiven Aufgaben effektiv entlastet [11].

### 2.1.2 *Strukturierte Anforderung*

Die automatisierte Analyse von unstrukturierter natürlicher Sprache kann durch das Verfassen von Softwarespezifikationen mithilfe einer definierten Struktur unterstützt werden [54]. Jede Art von (halb-)automatisierter Analyse profitiert von Softwarespezifikationen, die eine bestimmte Struktur aufweisen oder einem bestimmten Modelltyp entsprechen [26]. Allerdings ist das Ausdrücken von Anforderungen in Form von Modellen in der Regel mit zusätzlichem Aufwand für die Erstellung der Modelle verbunden und die Ergebnisse sind möglicherweise weniger gut für die Kommunikation mit nicht-technischen Stakeholdern geeignet (insbesondere bei formalen Modellen). Daher wurden verschiedene Ansätze vorgeschlagen, um die Anforderungsmodellierung zu vereinfachen [107]. Einige konzentrieren sich auf die Unterstützung von Ingenieur:innen bei der Erstellung von grafischen Modellen [101]. Andere zielen darauf ab, das Erscheinungsbild der natürlichen Sprache beizubehalten und dennoch automatisierte Schlussfolgerungen mit sogenannter eingeschränkter natürlicher Sprache zu ermöglichen [66].

Die Verwendung von eingeschränkter natürlicher Sprache ist ein Ansatz zur Erstellung von Anforderungsmodellen unter Beibehaltung des Aussehens natürlicher Sprache. Neben dem Vorteil, dass Anforderungen einheitlich formuliert werden, reichern die Anforderungsmuster Teile der Anforderung mit Informationen über die Semantik an. Diese Informationen können genutzt werden, um Informationen aus den Anforderungen zu extrahieren. Beispielsweise ist die Erstellung konzeptioneller Modelle in User Stories dadurch möglich [63].

So ermöglicht der Gherkin-Ansatz eine strukturierte Spezifikationsdarstellung unter Verwendung von natürlicher Sprache und der Vorgabe von Schlüsselwörtern [82]. Abbildung 2.1 zeigt eine Beispielspezifikation einer Funktion, die zwei Zahlen addiert.

```
Funktion: Taschenrechner

Einfache Berechnug durch Addition von zweier Zahlen.

Scenario Addiere zwei Ziffern.
Given Ich tippe <First> in den Taschenrechner.
And Ich tippe <Second> in den Taschenrechner.
When Ich drücke Addieren.
Then Auf dem Bildschirm erscheint <Result>.

    Beispiele:
    | First | Second | Result |
    | 50    | 70     | 120    |
    | 30    | 40     | 70     |
```

Abbildung 2.1: Beispiel einer Gherkin Spezifikation

```
When an Order is shipped and Order Terms are not prepaid,
the system shall create an Invoice.

  Trigger When an Order is shipped
  Precondition Order Terms are not prepaid
  Actor the system
  Action create
  Object an Invoice
```

Abbildung 2.2: Beispiel einer EARS Spezifikation

Im Beispiel der Gherkin Spezifikation können die wechselnden Parameter `First` und `Second` automatisiert mit dem erwarteten Ergebnisse `Result` abergeglichen werden. Eine weitere Möglichkeit zur Strukturierung von Softwarespezifikationen ist eine feste Vorgabe von Satzmustern. Diese haben ein natürlichsprachliches Erscheinungsbild, sind aber strukturiert formuliert. Der Ansatz Easy Approach to Requirements Syntax (EARS) von Mavin et al. definiert eine feste Satzstruktur mit dem Ziel alle Anforderungen fomulieren zu können [66]. Die generische Syntax hat beispielsweise folgende Form:

[Trigger] [Precondition] Actor Action [Object]

Hierbei beschreibt der `Trigger` die auslösende Aktion, die `Precondition` gibt Vorbedingungen an, der `Actor` beschreibt, wer oder was die `Action` ausführt und das `Object` bezeichnet das Objekt der Ausführung. In Abbildung 2.2 ist eine Beispielanforderung an ein Bezahlsystem unter Verwendung der vorgegebenen Syntax dargestellt.

Vorteil dieses Vorgehens und von Satzmustern im Allgemeinen ist zum einen eine strukturierte Anforderungserhebung mit qualitativen Anforderungen [25]. Zum anderen erleichtert es die automatisierte Verabeitung für weitere Entwicklungsschritte. Ein allgemeiner Nachteil aller Ansätze mit vorgegebenen Satzmustern ist die Einschränkung

der Anforderungsingenieur:innen bei der Formulierung. Des Weiteren arbeiten viele Unternehmen derzeit mit unstrukturierten natürlichsprachlichen Anforderungen [30]. Ein Umstieg ist speziell für große oder komplexe Systeme kostenintensiv und zeitaufwändig. Daher stehen Analysemethoden von unstrukturierten natürlichsprachlichen Systemspezifikationen, wie in Kapitel 2.1.1 beschrieben, im Fokus vieler Forschungsarbeiten. Methoden wie EARS oder Gherkin zeigen jedoch Vorteile von Sprachelementen und Strukturierungsmöglichkeiten bei der Erstellung von Softwarespezifikationen auf.

Der hier präsentierte Ansatz verknüpft die Ideen und Herangehensweisen der vorgestellten Methoden aus Kapitel 2.1.1 mit der Verwendung von Satzmustern. Ausgangspunkt sind die unstrukturierten natürlichsprachlichen Systemspezifikationen, die mithilfe von Elementen einer Multilevel Markup Language markiert werden. Dabei wurde die Markierungssprache eigenständig entwickelt, die bisherigen Ansätze sind jedoch in die Überlegungen bei der Entwicklung der Sprache eingeflossen.

## 2.2 TESTPROZESSE VON SOFTWAREFUNKTIONEN

Die Entwicklung von Softwarefunktionen wird von einem umfassenden Requirements- und Testmanagement begleitet. Die Testprozesse unterscheiden sich jedoch stark in Abhängigkeit zum testenden System, beispielsweise Software für Endanwender:innen oder für eingebettete Systeme. Der vorgestellte Testansatz ist generell nicht auf einen spezifischen Softwaretyp begrenzt, ist aber durch die vorherrschenden Entwicklungsmethoden im jeweiligen Bereich prädestiniert für die Entwicklung von eingebetteten Systemen.

### 2.2.1 *Teststufen der Softwareentwicklung*

Das Testen von Software ist die Verifizierung, dass ein Softwareprodukt das in seinen Anforderungen spezifizierte Verhalten erfüllt [8].

Der herkömmliche Entwicklungs- und Testprozess für eingebettete Systeme basiert auf dem V-Modell, das den Entwicklungsprozess in Phasen der Dekomposition der Systemelemente und deren anschließender Integration strukturiert. Jede Anforderung, die auf einer bestimmten Abstraktionsebene spezifiziert ist, wird durch einen Testfall zur Überprüfung der korrekten Anforderungsumsetzung auf der gleichen Ebene abgebildet. Abbildung 2.3 zeigt die unterschiedlichen Phasen des V-Modells [92].

Im Fokus dieser Arbeit stehen insbesondere die System-Spezifikationsphase auf der linken Seite und die Teststufen auf der rechten Seite des Modells. Die System-Spezifikationsphase umfasst das Requirements Engineering und enthält Methoden und Vorgehensweisen zur Spezifizierung. Alle folgenden Phasen bis zum Sofwareentwurf

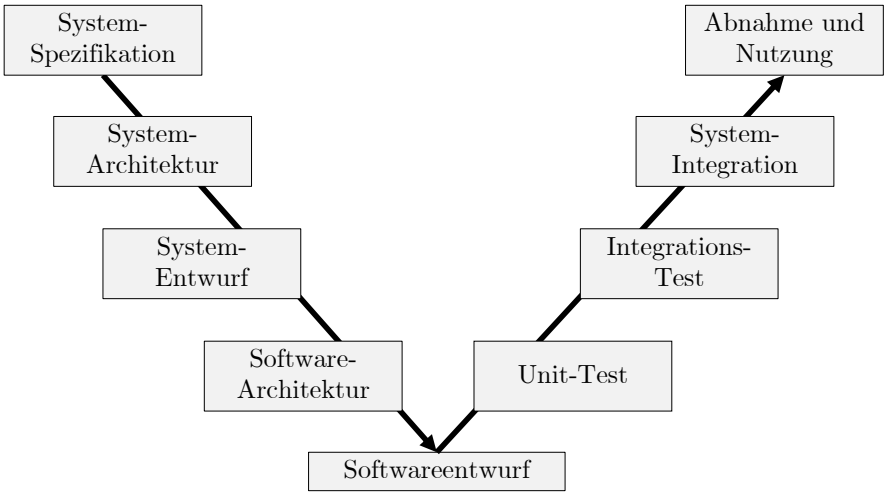



Abbildung 2.3: V-Modell der Softwareentwicklung

beschreiben weitere Spezifikationsphasen von Teilmodulen der Software. Die rechte Seite des V-Modells gliedert sich in vier Testphasen, die detailliert bei einzelnen Modultests, den Unit-Tests, beginnen und im Verlauf der Entwicklung stetig größere und komplexere Teile der Software in Betracht ziehen. Schlussendlich erfolgt die Abnahme und die Nutzung der Software, in der das Gesamtsystem mit allen Funktionen betrachtet und gegen die System-Spezifikation abgeglichen wird. Hierbei werden auf allen Teststufen explizite Testfälle für die Absicherung der Fuktionen definiert.

### 2.2.2 *Softwaretest für eingebettete Systeme*

Die zunehmende Komplexität der Systeme, die große Anzahl möglicher Testfälle und die Ungewissheit über den Systemkontext stellen diesen konventionellen Testprozess in Frage. Daher wird der Einsatz von Simulationen immer beliebter, um das Testen von Software in unterschiedlichen und komplexen Umgebungen zu erleichtern [17]. Auf diese Weise können verschiedene Verhaltensweisen der Software automatisch untersucht werden.

Eine Simulation ist die Nachahmung des Betriebs eines realen Prozesses oder Systems [6]. Um etwas zu simulieren, muss zunächst ein Modell entwickelt werden, das die wichtigsten Eigenschaften, das Verhalten und die Funktionen des ausgewählten physischen oder abstrakten Systems oder Prozesses enthält. Ein Simulator ist ein Programm, das in der Lage ist eine Simulation auszuführen. Jeder Simulationslauf ist eine Ausführung der Simulation.

Softwaretests mit Simulationsunterstützung unterscheiden sich je nach Anwendungsbereich. In den letzten Jahren hat der Einsatz von Simulationen in vielen Bereichen zugenommen. Dabei unterstützen Simulationen alle Testpahsen des herkömlichen Testmanagements aus

dem V-Modell [86]. Simulationen bilden die reale Welt mit einem gewissen Detaillierungsgrad virtuell ab [6]. Sie ermöglichen das virtuelle Testen aller Anforderungen bei vergleichsweise geringen Hardwarekosten, da viele umfangreiche Testfälle abgedeckt werden können.

Wenn die Simulation in einem Systementwicklungsprozess verwendet wird, besteht das Modell in der Regel aus einem Teilmodell, das das zu entwickelnde System - das System-under-Development (SuD) - beschreibt, und einem oder mehreren Teilmodellen, die die Betriebsumgebung des SuDs beschreiben. Die Simulation stellt den Betrieb des SuDs in seinem betrieblichen Kontext über die Zeit dar.

Ein Simulationsszenario definiert die anfänglichen Eigenschaften und Vorbedingungen eines Simulationslaufs und erstreckt sich über eine bestimmte Zeitspanne. Das Szenario definiert die globalen Parameter des Betriebskontextmodells. Das Modell des SuD wird durch die Definition des Simulationsszenarios nicht beeinflusst. Daher kann ein Simulationsszenario mit einem Testfall in einem herkömmlichen Testverfahren verglichen werden. Gegenstand des Testmanagements ist die Überprüfung darauf, ob sich das SuD in einer Menge von repräsentativen Simulationsszenarien entsprechend seiner Spezifikation verhält.

Durch den Einsatz von Simulationen haben sich neue Herausforderungen im Testmanagement herauskristalisiert, bestehend aus der Entwicklung adäquater Simulationsszenarien [37] und der eigentlichen Auswertung im Sinne eines Abgleichs der Systemausführungen mit den initialen Anforderungen.

## 2.3 ANFORDERUNGS- UND TESTANGLEICHUNG

Bisherige Arbeiten zeigen, dass ein fehlender Abgleich zwischen Anforderungs- und Testaktivitäten in vielen Softwareprojekten einer der Hauptgründe für Softwarefehler ist [43, 79]. Daher sind Ansätze zur Verknüpfung von Anforderungen mit Testdurchläufen ein wesentlicher Aspekt, der zur Verbesserung der Softwareentwicklung beiträgt [13, 90, 91]. Neben dem technischen Abgleich von Anforderungen und Testfällen stehen Methoden zur Verfügung, die zur besseren Kommunikation innerhalb eines Softwareprojekts beitragen [42, 44]. Hierbei werden Änderungen in Anforderungen automatisiert erkannt und Änderungsvorschläge für Akzeptanztests erstellt. Ein Teil der Praktiken, die die Verbindung zwischen Spezifikationen und Tests verbessern, involviert verschiedene Testrollen [91]. Die verwendeten Methoden zielen vorrangig auf umfassende Kommunnikation und Interaktion ab, unter anderem durch frühzeitige Beteiligung von Tester:innen, Richtlinien zur Nachvollziehbarkeit, die Berücksichtigung von Feature-Anfragen von Tester:innen und die Verknüpfung von Test- und Anforderungsexpert:innen. Dabei sind die Verbesserung der in-

terpersonellen Kommunikation und die Verknüpfung von Dokument gleichermaßen wichtig [91].

Der technische Abgleich von Anforderungen und Testfällen ist ein etablierter Forschungsbereich mit mehreren existierenden Lösungen. Eine Vielzahl von bisherigen Ansätzen befasst sich mit modellbasiertem Testen und formalen Methoden zur Beschreibung von Anforderungen mit Modellen oder Sprachen [8]. Eine weitere Möglichkeit Anforderungen und Testfälle zu verknüpfen, ist die Herstellung von Dokumentenverknüpfungen (Traceability-Links) zwischen den Spezifikations- und Testdokumenten [78].

Im vorgestellten Testansatz steht die technische Verknüpfung von Spezifikation und Systemausführung im Vordergrund.

### 2.3.1 *Traceability von Anforderungen und Tests*

Ein Abgleich von Anforderungen und Testfällen kann durch das Einrichten von Traceability-Links zwischen den Dokumenten erreicht werden [35, 91]. Diese Traceability-Links sind ein Werkzeug, um Anforderungsartefakte mit der Information anzureichern, dass es ein Testdokument gibt, das die durch die Anforderung spezifizierte Funktionalität umfasst. Traceability-Links können mit verschiedenen Bedeutungen verknüpft werden, um Ingenieur:innen bei der Durchführung von Verhaltensanalysen oder der Erhöhung der Testabdeckung zu unterstützen [91]. Zu den Herausforderungen bei der Nachverfolgbarkeit gehören die Volatilität der nachverfolgten Anforderungen, informelle Prozesse mit unklaren Verantwortlichkeiten für die Nachverfolgung, Kommunikationslücken, unzureichende Zeit und Ressourcen für die Aufrechterhaltung von Nachverfolgungen in Kombination mit einer Praxis, die als nicht kosteneffizient angesehen wird und ein Mangel an Schulung [18]. Darüber hinaus sind Traceability-Links statische Verknüpfungen, die lediglich eine oder keine Verknüpfung zwischen Anforderungen oder Testfällen herstellen.

Unser Testansatz verknüpft Informationen dynamischer Ausführungen mit Systemspezifikationen. Dies erleichtert die Bewertung von Anforderungen auf eine detailliertere Art und Weise, indem nicht nur das Vorhandensein des Testfalls für die Anforderung gezeigt wird. Tatsächlich wird geprüft, dass die Anforderungen erfolgreich in einer Systemausführung verifiziert werden.

### 2.3.2 *Test Coverage*

Ein wichtiges Maß zur Beurteilung von Testansätzen ist der Begriff ihrer Abdeckung, der *coverage* [111]. Vor allem werden Testfälle und Testmethoden anhand ihrer Codeabdeckung bewertet. Diese umfasst den Anteil des Codes, der bei der Ausführung eines Testfalls ausgeführt wird.

Speziell für Black-Box-Tests wurde das Konzept der Anforderungsabdeckung definiert, um den Anteil der Anforderungen zu bewerten, die von einer Testmethode „ausgeführt“ werden [97]. Ähnlich wie bei Ansätzen zum Abgleich von Anforderungen und Tests kann die Anforderungsabdeckung auf Basis von (formalen) Anforderungsmodellen oder auf Basis von statischen Traceability-Links zwischen Testfällen und Anforderungsdokumenten berechnet werden [78, 97].

In dieser Arbeit liegt der Fokus auf Black-Box-Tests und der Berechnung von Anforderungsabdeckung. Wir stützen uns jedoch nicht auf formal definierte Anforderungen sondern betrachten natürlichsprachliche Spezifikationen. Dennoch sind detailliertere Bewertungen der Beziehung zwischen Anforderungen und Tests möglich, die über die statischen Beziehungen, die durch einen Trace-Link angezeigt werden, hinausgehen.

## 2.4 VERWANDTE ARBEITEN

Einen vergleichbaren Ansatz um Anforderungen innerhalb einer Test- und Simulationsumgebung durchgängig zu verifizieren, stellt das Werkzeug Stimulus der Softwarefirma Argosim[6] dar. Mit Stimulus können die Anwender:innen formalisierte Anforderungen definieren und das zu entwickelnde System mit Zustandsautomaten und Blockdiagrammen anreichern, um Verhaltens- bzw. Architekturinformationen einzubeziehen. Mithilfe einer eingebauten Testumgebung können Signale aus der Umgebung, von der das System abhängt und auf die es reagiert, simuliert werden. Innerhalb dieser Simulationen werden das Systemverhalten im Hinblick auf seine durch die Anforderungen spezifizierten Randbedingungen bewertet und Verstöße erkannt. Zu den Hauptmerkmalen gehört die Erkennung von widersprüchlichen und fehlenden Anforderungen.

Dieser Ansatz weist jedoch einige wesentliche Unterschiede zum hier gezeigten Testansatz auf. Die von Stimulus geforderte Form der Anforderungen ist stark formalisiert.

Der hier vorgestellte Ansatz hingegen ermöglicht es Anforderungsingenieur:innen, natürlichsprachliche Anforderungen intuitiv zu markieren. So bleiben die implizit enthaltenen Informationen für Verifizierungszwecke innerhalb einer Systemausführung erhalten.

Zweitens hängt die von Stimulus bereitgestellte Testfunktion davon ab, dass die Anwender:innen Eingaben für das System definieren und diesen einen Wertebereich für die Testausführung zuweisen. Dieser Schritt kann durch unseren Testansatz automatisiert werden, da testrelevante Informationen in den Anforderungen enthalten sind und markiert werden. In den Informationen der Markierungen sind Anforderungen an die Testumgebung enthalten.

6 www.argosim.com

Die Arbeit von Beck et al. befasst sich mit der Darstellung von Ergebnissen in natürlichsprachlichen Texten [9]. Hier wird ebenfalls die fehlende ganzheitliche Auswertung von Programmverhalten auf Basis natürlichsprachlicher Texte adressiert. Im Vordergrund steht die Auswertung von Methoden in Softwareprogrammen mit einer natürlichsprachlichen Ergebnisrepräsentation. Hierbei wird der resultirende Text jedoch nach der Analyse automatisiert generiert und nicht in vorhandene Texte integriert. Die Arbeit zeigt jedoch die Vorteile der Ergebnisdarstellung in natürlichsprachlichem Kontext.

Der hier präsentierte Testansatz fokussiert sich auf die heutigen eingesetzten natürlichsprachlichen Softwarespezifikationen und verknüpft sie mit Systemausführungen, die den gesamten Enwicklunsprozess begleiten. Dabei werden Analysemethoden aus der automatisierten Textverarbeitung (NLP) integriert, ohne die Spezifikationserhebung einzuschränken. Die entwickelten Analysemethoden werten Logdaten beliebiger Testphasen aus und ermöglichen insbesondere eine anforderungsbezogene Auswertung von Simulationen, deren Bedeutung für das Testmanagement stetig wächst.

# 3

# ANFORDERUNGSBASIERTER TESTPROZESS

Dieses Kapitel gibt einen Überblick über den entwickelten Testprozess und stellt die eingesetzte Sprache vor. Zu Beginn wird in Kapitel 3.1 der Ablauf eines Testprozesses vorgestellt und die enthaltenen Teilprozesse eingeführt. Anschließend werden die Level und Elemente der eingesetzten Sprache in Kapitel 3.2 im Detail beschrieben. Auf dieser Sprache basiert der gesamte Testprozess und ermöglicht den Nutzer:innen die Verknüpfung von Anforderungen und Testausführungen. Da der Testprozess manuelle Schritte enthält, ist Domänenwissen der Nutzer:innen erforderlich. Welche Voraussetzung notwendig und welches Domänenwissen erforderlich ist, wird in Kapitel 3.3 erläutert.

Das Kapitel basiert auf dem bereits veröffentlichtem Beitrag von Pudlitz et al. [76].

## 3.1 ÜBERBLICK ÜBER DEN TESTPROZESSES

### 3.1.1 *Definition der Nutzer:innen*

Der Ansatz dieser Arbeit enthält manuelle Anteile, die es ermöglichen den gesamten Testansatz individuell auf den Entwicklungsprozess der Softwarefunktion anzupassen. Die Durchführung dieser manuellen Schritte ist von verschiedenen Faktoren abhängig, beispielsweise von der Aufteilung des Entwicklungsprozesses, vom Zeitpunkt des Einsatzes im Entwicklungsprozess und von den involvierten und verantwortlichen Entwickler:innen. Zusätzlich kann der Verwendungszweck mancher Teilprozesse des Ansatzes variieren und damit die resultierenden Verantwortlichen, die in die manuellen Schritte involviert sind.

Durch die bisherigen Entwicklungsprozesse haben sich unterschiedliche Berufsqualifikationen etabliert. So gibt es in vielen Branchen Anforderungsmanager:innen, die vorrangig mit der Erstellung von Softwarespezifikationen betraut sind und Testmanager:innen, die das Testen der Software verantworten. Der vorgestellte Ansatz dieser Arbeit kann in allen Bereich des Entwicklungsprozesses eingesetzt werden und tangiert, bedingt durch die manuellen Schritte, verschiedene Bereiche und Personengruppen. Welches Wissen für bestimmte Teile des Ansatzes notwendig ist, wird in Kapitel 3.3 genauer erläutert.

In dieser Arbeit findet keine Unterscheidung in Hinblick auf die involvierten Berufsgruppen statt, da diese variieren können und nicht im Zentrum dieser Ausführung stehen. Daher schreibe ich im Fol-

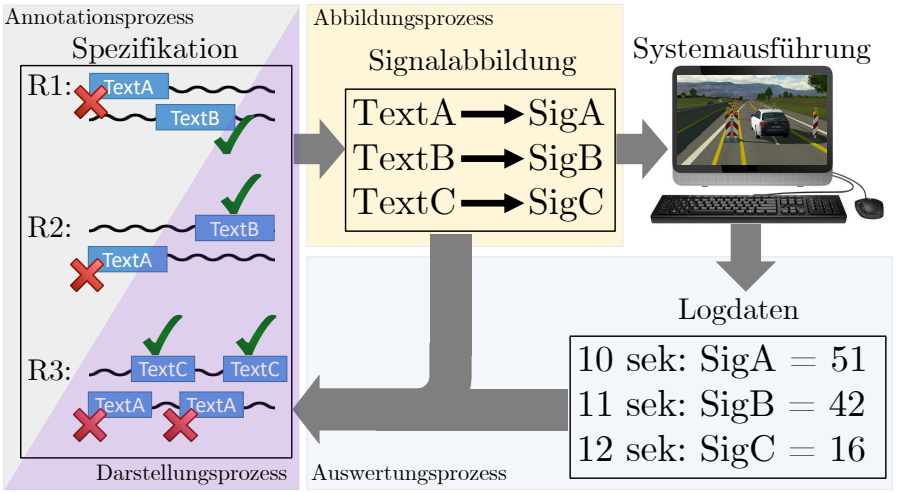



Abbildung 3.1: Schematische Repräsentation des anforderungsbasierten Testablaufs und die Unterteilung der Teilprozesse

genden stets von *Nutzer:innen*, wenn manuelle Schritte im Testansatz durchgeführt werden.

### 3.1.2 *Ablauf eines Testdurchlaufs*

Der Testansatz ist schematisch in Abbildung 3.1 dargestellt. Ausgangspunkt sind natürlichsprachliche Anforderungen, welche exemplarisch in Abbildung 3.1 als R1, R2 und R3 gezeigt sind. Die Anforderungen beschreiben Softwarespezifikationen und sind ohne Einschränkung der Sprache oder Vorgaben formuliert, insbesondere ohne vorgegebene syntaktische Form. Daher kann nicht ausgeschlossen werden, dass beispielsweise Rechtschreib- und Grammatikfehler sowie Synonyme enthalten sind.

Mit Elementen der entwickelten Markierungssprache können Nutzer:innen einzelne Wörter oder Textpassagen annotieren. Diese Markierungen werden als Annotation bezeichnet und sind wie folgt definiert:

> *Eine Annotation bezeichnet eine Funktion, die einen Text auf ein Element der Multilevel Markup Language abbildet.*

Die Multilevel Markup Language ist in Kapitel 3.2 genauer spezifiziert. Die Annotationen sind jeweils als *TextA*, *TextB* und *TextC* in den Anforderungen R1, R2 and R3 exemplarisch dargestellt. Beispielsweise sind systemrelevante Textstellen wie *Motor*, *Tür*, *Geschwindigkeit* oder *Beschleunigung* in Anforderungen für Systeme aus dem Automobilkontext enthalten. Jede Annotation wird anschließend manuell von den Nutzer:innen mit einem spezifischen Wert der Implementierung, der ein bestimmtes Systemverhalten beschreibt, im Abbildungsprozess verknüpft. Bezogen auf das Beispiel wird TextA dem Signal SigA zugeordnet (analog dazu TextB und TextC). Die Annotationen enthalten

zum einen systembezogene Informationen die in der Testausführung zu beobachten sind. Zum anderen können in Annotationen relevante Informationen über die Systemumgebung, beispielsweise *Wetter* oder *Straßenbeschaffenheit*, enthalten sein um die Erstellung von Testszenarien zu unterstützen.

Die Testdurchführung ist zunächst unabhängig von den Annotationen. Hierbei wird das zu testende System ausgeführt. Die dargestellte Fahrzeugsimulation in Abbildung 3.1 zeigt nur eine von vielen möglichen Systemausführungen. Während des Testdurchlaufs speichert die Testumgebung alle auftretenden Ereignisse. Diese Datenaufzeichnungen werden Logdateien genannt und im Anschluss an den Test automatisiert ausgewertet. In den Logdateien sind die aufgezeichneten Signal- und Zustandsänderungen des Systems enthalten. Die dargestellte Logdatei in Abbildung 3.1 zeigt einen exemplarischen Ausschnitt der Sekunden 10 bis 12 mit den auftretenden Signalnamen und den dazugehörigen Werten. Eine automatisierte Auswertung, beispielsweise von definierten Bedingungen oder kausalen Zusammenhängen des Systems, basierend auf den zuvor erstellten Annotationen, erfolgt im Auswertungsprozess auf den Logdaten des Testdurchlaufs. Die Auswertungsergebnisse der Annotationen werden anschließend im Darstellungsprozess grafisch in den natürlichsprachlichen Anforderungen und tabellarisch dargestellt. Hierbei ermöglichen Informationen aus der Zuweisung von Textphrasen zu den Signalen die Zuordnung der Ergebnisse zu den Annotationen.

Diese Vorgehensweise ermöglicht den Nutzer:innen das Verhalten des Systems, unabhängig von dessen Komplexität, zu evaluieren. Dabei steht das Erreichen von zwei Hauptzielen im Vordergrund. Erstens die Gewährleistung der Erfüllung der Anforderungen mit gewünschtem Detaillierungsgrad, unabhängig von der Komplexität des Systems. Zweitens bereits in frühen Entwicklungsphasen das Verständnis des Systemverhaltens in komplexen Situation im Kontext der Anforderungen zu erhöhen.

Der vorgestellte Testansatz ist hierbei unabhängig von der gewählten Teststufe und erlaubt eine Auswertung in jeder Entwicklungsstufe. Die Ergebnisdarstellung in den Ausgangsanforderungen ermöglicht eine Systemevaluation im Zusammenhang des ursprünglichen Kontexts. Es gibt hierbei keine Informationsverluste durch Anforderungsformalisierungen oder anderen Übersetzungen.

## 3.2 MULTILEVEL MARKUP LANGUAGE

Für die Annotation von Softwarefunktionen und Umgebungsbedingungen haben wir eine Markierungssprache entwickelt um natürlichsprachliche Anforderungen und Logdaten aus den Testdurchläufen zu verknüpfen. Hierzu werden die folgenden fünf Ziele definiert:

**Extraktion aus Spezifikationen:** Die entwickelte Sprache dient als eine leichtgewichtige, intuitive Methode um Objekte aus natürlichsprachlichen Softwarespezifikationen zu extrahieren.

**Extraktion von Umgebungseigenschaften:** Die Sprache ermöglicht eine Extraktion von wichtigen Umgebungseigenschaften, die in einem Testdurchlauf auftreten müssen.

**Erstellung von Zusammenhängen:** Mit der Sprache können sowohl einfache als auch komplexe Beziehungen zwischen Informationen in Softwarespezifikationen hergestellt werden, ohne eine formale Überführung durchzuführen.

**Beobachtung von Softwareverhalten:** Mithilfe der Sprache besteht die Möglichkeit Softwareverhalten während des gesamten Entwicklungsprozesses zu beobachten.

**Vergleich von Teststufen:** Die Bestandteile der Sprache dienen als Ausgangspunkt zur Extraktion von Bedingungen, die dem Vergleich von Testszenarien und Teststufen dienen.

Basierend auf diesen Zielen wurde eine Sprache bestehend aus vier Levels entwickelt, die jeweils Markierungselemente gruppieren. Das ermöglicht eine differenzierte Informationsextraktion in Abhängigkeit vom Ziel der Nutzer:innen. Die vier Levels der Sprache enthalten individuelle Elemente zur Textannotation. Das Annotieren wird von Nutzer:innen durchgeführt und ist bewusst ein manueller Vorgang. Dadurch ist der Ansatz unabhängig von den Anforderungen und für unterschiedliche Anforderungstypen geeignet. Abhängig vom gewünschten Zeitaufwand sind sowohl einfache und somit schnelle Auswertungen als auch komplexe, zeitaufwändigere Systemevaluationen möglich. Während des Annotationsprozesses wählen die Nutzer:innen ein Element abhängig vom Kontext und der gewünschten Evaluation aus. Das dazugehörige Level wird hierbei nur indirekt von den Nutzer:innen ausgewählt. Der aufeinander aufbauende Charakter der Level gibt den Nutzer:innen die Flexibilität den Annotationsprozess zu einem späteren Zeitpunkt, beispielsweise bei voranschreitendem Entwicklungsverlauf, detaillierter fortzuführen.

Der Annotationsprozess wird detailliert in Kapitel 4 beschrieben. In den folgenden Kapiteln 3.2.1 und 3.2.2 sind die Elemente, die dazugehörigen Level und ihre Auswertungsmöglichkeiten beschrieben.

### 3.2.1 *Definition von Elementen*

Die Elemente sind Grundbestandteil der Sprache und sind Bezeichnungen, die den Annotationen zugeordnet werden. Die Nutzer:innen wählen selbst, welches Element für die Annotation genutzt wird. Abhängig vom Kontext des Systems kommen nur ausgewählte Elemente für eine Annotation in Frage. Das richtige Verständnis der Elemente ist hierbei entscheidend, da die Auswahl des Elements die Art der

automatisierten Auswertung beeinflusst. Jedes Element wird einem von vier Levels zugeordnet, die den Detaillierungsgrad der Auswertung definieren. Der aufeinander aufbauende Charakter der Levels ist auch durch die Namensgebung der Elemente ersichtlich. Die Elementnamen sind entweder eindeutig angegeben oder durch einen zusätzlich tiefgestellten Elementnamen genauer spezifiziert. Die {L1}, {L2} und {L3} Bezeichnungen in den Elementnamen stehen für den Elementnamen des nächstniedrigeren Levels.

Die Namenskonvention soll nun am Beispiel des Elementnamens $\text{Value}_{\{L1\}}$ gezeigt werden. In der Level 2 Annotation $\text{Value}_{\text{System}}$ zeigt das tiefgestellte `System` den Bezug zur System-Annotation aus Level 1. Der Bezug kann auf zwei Wegen hergestellt werden. Einerseits kann eine Textstelle zuerst als System markiert und anschließend in eine $\text{Value}_{\text{System}}$-Annotation umgewandelt werden. Andererseits ist eine direkte Markierung als $\text{Value}_{\text{System}}$ möglich.

Die zweiteilige Definition der Elemente umfasst zum einen, welche semantische Bedeutung geeignet für ein Element ist. Zum anderen die möglichen Auswertungen. Diese sind fest definiert und beeinflussen unter Umständen die Nutzer:innen in der Wahl des Elements für eine Textphrase.

Die **System**-Annotation beschreibt systembezogene Textpassagen, die für die funktionale Ausführung erforderlich sind. Diese Annotationen sollen auf Textstellen angewendet werden, die Informationen bezüglich des Systems beinhalten. Hierzu zählen Eigenschaften, Begriffe oder Parameter des zu testenden Systems. Im Anschluss an den Markierungsprozess werden die Annotationen hinsichtlich ihrer Anzahl beziehungsweise der Häufigkeit der dazugehörigen Signale in den Systemausführungen evaluiert.

Die **Environment**-Annotation bezeichnet testrelevante Textpassagen, die sich auf alle Informationen außerhalb der Systemgrenzen beziehen, beispielsweise auf die Umgebung, die Testplattform oder weitere Systeme. In der Auswertung der *Environment* Annotationen findet eine Überprüfung auf das Auftreten der Signale statt.

Die $\textbf{Value}_{\{L1\}}$-Annotation beschreibt Textstellen, die einen Wert beschrieben, der einem kontinuierlichen Verlauf folgt. Die Auswertungen zeigen entsprechend den Verlauf des Werts über die gesamte Testzeit an.

Die $\textbf{State}_{\{L1\}}$-Annotation umfasst alle Textstellen, die im Testverlauf einen definierten Zustand einnehmen. Die markierten Textstellen bezeichnen jeweils den Zustandsnamen. Diese können im Testverlauf mehrere aber disjunkte Zustandswerte annehmen. Beispielsweise kann der Zustandsname „Tür“die Zustandswerte „geschlossen“oder „geöffnet“annehmen. Die Auswertung der Tests zeigt entsprechend alle auftretenden Zustandswerte für die markierte Textstelle an.

Die $\textbf{Event}_{\{L1\}}$-Annotation bezeichnet einmalig oder sporadisch auftretende Signalereignisse. Sie treten kurzzeitig und je Signalnamen

mit fest definierten Werten auf. Die Ergebnisse zeigen das Auftreten aller Eventwerte und deren Häufigkeit.

Die **Time**-Annotation ist eine Markierung, die für Zeitaspekte wie Zeiträume, Angaben von Dauer oder Zeitbeschränkungen Anwendung findet. Die Markierung besteht vorrangig aus einer Zahl und wird als einzelne Markierung nicht ausgewertet. Die Annotation kann anschließend um einen Bedingungsoperator erweitert werden und wird dann als *Time-Condition* bezeichnet.

Die Nutzer:innen haben die Möglichkeit die zuvor erläuterten Elemente in Bedingungen umzuwandeln. Dazu werden weitere Informationen in den natürlichsprachlichen Anforderungen markiert und mit bereits bestehenden Annotationen mithilfe eines logischen Operators verbunden. Die möglichen Operatoren unterschieden sich abhängig vom gewählten Element. Ausgehend vom Typ der Annotation werden aus $\text{Value}_{\{L1\}}$, $\text{State}_{\{L1\}}$, $\text{Event}_{\{L1\}}$ und Time jeweils $\text{Value}_{\{L1\}}$-Condition, $\text{State}_{\{L1\}}$-Condition, $\text{Event}_{\{L1\}}$-Condition und Time-Condition.

Die **$\text{Value}_{\{L1\}}$-Condition** beschreibt prüfbare Bedingungen basierend auf kontinuierlichen Werteverläufen. Die Annotationen bestehen aus den drei Teilen Annotationsnamen, Operator und Annotationswert. Die logischen Operatoren $<, \leq, =, >, \geq$ und $\neq$ stehen für die Verknüpfung mit dem Annotationswert zur Auswahl. Die Werte sind vorrangig Zahlen oder Parameter und können entweder ebenfalls in der Spezifikation annotiert oder von den Nutzer:innen selbst festgelegt werden. In den Auswertungen erfolgt die Angabe der Dauer dieser erfüllten Bedingung und der prozentuale Anteil bezogen auf die Gesamttestdauer.

Die **$\text{State}_{\{L1\}}$-Condition** dient zur Beschreibung einer Bedingung, die im Zusammenhang mit fest definierbaren Zuständen steht. Mit dieser Annotation erfolgt eine Verknüpfung zwischen dem Element $\text{State}_{\{L1\}}$ und einem möglichen Zustand mithilfe der Operatoren $=$ oder $\neq$. Die drei Bestandteile werden State-Name, State-Operator und State-Wert genannt. Die Nutzer:innen können den State-Wert in der Spezifikation annotieren oder selbst frei wählen. Die Ergebnisdarstellung der Annotation zeigt Häufigkeit und die Dauer der erfüllten Bedingung absolut und prozentual an.

Die **$\text{Event}_{\{L1\}}$-Condition** ist eine Annotationen, die das Auftreten von einmaligen oder sporadisch auftretenden Signalereignissen prüft. Bei der Erstellung wird dem Eventnamen, der zuvor mit dem Element $\text{Event}_{\{L1\}}$ markiert wurde eine 1 oder 0 als Eventwert, zugeordnet. Damit kann das Auftreten (1) oder das Nichtauftreten (0) der Signalwerte in den Testläufen überprüft werden. Es besteht zusätzlich die Möglichkeit einen selbstgewählten Eventnamen zu vergeben und überprüfen zu lassen. In der Testauswertung findet eine entsprechende Zählung der auftretenden Events und die Auswertung der durchschnittlichen Zeit zwischen zwei zugeordneten Werten statt.

Die **Time-Condition** fügt dem vorhandenen markierten Element *Time*, welches vorrangig aus Zahlen besteht, einen Bedingungsoperator hinzu. Dieser kann einer der logischen Operatoren $<, \leq, =, >, \geq$ oder ein natürlichsprachlicher Ausdruck wie *länger*, *kürzer* oder *innerhalb* sein. Für die *Time-Condition* findet keine eigenständige Auswertung statt. Sie dient der Erweiterung von anderen Annotationen um eine zeitliche Zusatzbedingung und muss dazu mit diesen verknüpft werden.

Die Auswertung der bisher vorgestellten Elemente findet unabhängig voneinander statt. Eine Möglichkeit komplexere Sachverhalte zu annotieren ist die Verknüpfung von bestehenden Annotationen miteinander. Dazu stehen die bisher vorgestellten Elemente $\text{Value}_{\{L1\}}$-Condition, $\text{State}_{\{L1\}}$-Condition, $\text{Event}_{\{L1\}}$-Condition zur Verfügung. Diese können jeweils mit einer Time-Condition zusätzlich annotiert sein. Bei der Verknüpfung zweier Elemente wird ein kausaler Zusammenhang hergestellt und zwei zuvor erstellte Annotation werden gemeinsam ausgewertet.

Ein **{L3}-Trigger** dient zur Beschreibung einer ereignisauslösenden Bedingung und besteht aus einer oder mehreren zuvor erstellten Condition-Annotationen. Ein {L3}-Trigger muss mit dem Element {L3}-Action verknüpft werden und setzt beide in einen kausalen Zusammenhang. Die Verbindung zeigt, dass die Bedingung des {L3}-Triggers erfüllt sein muss, um die Bedingung der {L3}-Action zu überprüfen. Bei der Erstellung eines {L3}-Triggers mit mehreren Annotationen muss einer der beiden Operatoren AND oder OR zur Verkettung angegeben werden. Bei einer AND-Verknüpfung gilt der {L3}-Trigger als erfüllt, wenn alle enthaltenen Bedingungen innerhalb einer Zeitspanne (meist nur wenige Millisekunden) erfüllt sind. Im Gegensatz dazu gilt ein {L3}-Trigger mit OR-Verknüpfung als erfüllt, wenn mindestens eine enthaltene Bedingung erfüllt ist. Nach einem Testdurchlauf wird die Häufigkeit der erfüllten Annotation ausgewertet.

Eine **{L3}-Pre-Condition** beschreibt Bedingungen, die als Voraussetzung für eine {L3}-Action gelten. Sie besteht aus einer oder mehreren vorhandenen Condition-Annotationen, die gemeinsam mit der {L3}-Action ausgewertet werden. Zu jedem Zeitpunkt der Auswertung einer {L3}-Action wird parallel überprüft, ob die Bedingungen der {L3}-Pre-Condition erfüllt sind. Eine {L3}-Pre-Condition kann aus einer oder mehreren Bedingungen bestehen, die jeweils mit AND oder OR verkettet sind. Analog zum {L3}-Trigger muss in einer OR-Verkettung mindestens eine Bedingung erfüllt sein, bei der Verwendung des AND Operators alle Bedingungen. Die Testauswertung evaluiert die Häufigkeit der erfüllten {L3}-Pre-Condition.

Die **{L3}-Action** umfasst Bedingungen, die abhängig von den Elementen {L3}-Trigger und {L3}-Pre-Condition ausgewertet werden. Die Annotation besteht ebenfalls aus bereits existierenden Condition-Annotationen und wird durch das Hinzufügen eines {L3}-Trigger

oder einer {L3}-Pre-Condition in eine {L3}-Action umgewandelt. Eine Verkettung von mehreren {L3}-Actions ist an dieser Stelle nicht möglich. Mehrere {L3}-Actions können getrennt annotiert werden. Die Auswertung zeigt die Häufigkeit der erfüllten Annotation.

### 3.2.2 *Gruppierung der Annotationen in Level*

Die vorgestellten Markierungselemente sind in vier Gruppen eingeteilt, die jeweils für einen Detaillierungsgrad der möglichen Auswertungen stehen. Diese Stufen werden als Level 1 bis Level 4 bezeichnet. Sie enthalten jeweils die Elemente, mit denen eine Markierung von einzelne Wörtern oder Textpassagen möglich ist. Die Level bauen aufeinander auf, da aus den Elementen eines Levels (z.B. $\text{State}_{\{L1\}}$) durch zusätzliche Markierungen ein Element eines höheren Levels entsteht (z.B. $\text{State}_{\{L1\}}$-Condition).

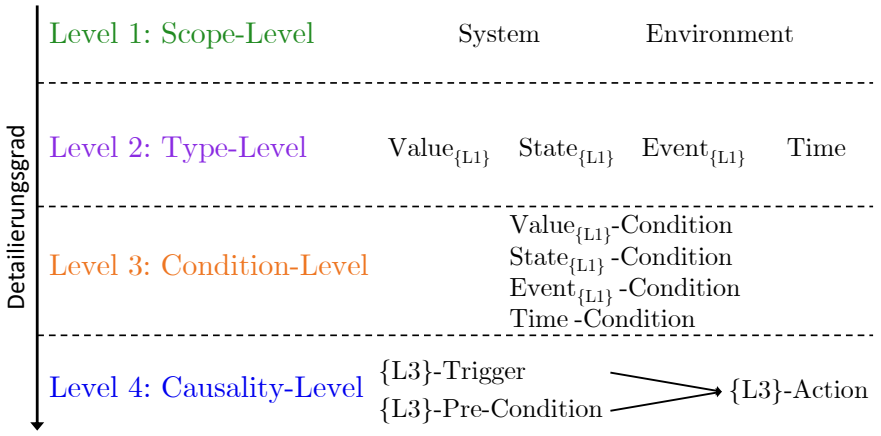



Abbildung 3.2: Übersicht aller Level und Elemente

Das *Scope-Level (Level 1)* dient zur Unterscheidung von Informationen nach *System* und *Environment*. Die Einteilung in diese zwei Kategorien ermöglicht eine Zuordnung der Annotation auf Anforderungen, die sich auf das System oder auf die Umgebung beziehen. Eine Unterteilung in diese zwei Kategorien findet sich auch in den Arbeiten von Gunter et al. ([38]) und Zave et al. [109] wieder, in denen die Unterscheidung zwischen System und Umwelt für den Entwicklungsprozess von Software entscheidend ist. Systemrelevante Informationen in Anforderungen werden vorrangig annotiert um Texte der Systemspezifikation semantisch dem zu entwickelnden System zuzuordnen. Daraus lassen sich im Verlauf des Testprozesses Rückschlüsse auf das Systemverhalten ziehen. Testrelevante Informationen beziehungsweise Textphrasen, die mit dem Element *Environment* annotiert wurden, werden semantisch der Systemumwelt zugeordnet. Diese Systemumwelt bezeichnet alles außerhalb der Funktionalität des Systems.

Mit dem *Type-Level (Level 2)* besteht die Möglichkeit, Textstellen in Typen einzuteilen. Die Typen sind abhängig vom Verhalten des Systems. Alle Typen sind entweder der Kategorie *System* oder *Environment* aus Level 1 zugeordnet. Mit der Wahl des jeweiligen Typs passt sich auch die Auswertungsmethode innerhalb des Testprozesses an. Die Typen aus Level 2 sind die Basis für alle weiteren Level.

Das *Condition-Level (Level 3)* ermöglicht eine Erweiterung der Level 2 Annotationen um eine logische Operation und einen Wert. Es entsteht so eine Bedingung. Welche logischen Operationen möglich sind, hängt vom jeweiligen Typen ab. Die erstellten Bedingungen stehen dabei in keinem Zusammenhang.

Das *Causality-Level (Level 4)* stellt eine Beziehung zwischen den Bedingungen des Level 3 her und schafft kausale Zusammenhänge. Dies setzt detaillierte Systemkenntnisse voraus. Der notwendige Annotationsprozess, der von den Nutzer:innen manuell durchgeführt wird, ist hierbei am zeitaufwendigsten. Die Evaluation stellt jedoch umfassende Ergebnisse bereit und umfasst den umfangreichsten Kontext.

## 3.3 DOMÄNENWISSEN DER NUTZER:INNEN

Die Qualität des beschriebenen Testansatzes in Kapitel 3.1.2 hängt sowohl von der Qualität der Annotationen als auch von der manuellen Durchführung der Signalabbildung ab. Um eine adequate Durchführung des Annotations- und Abbildungsprozesses zu gewährleisten, ist Fachwissen der Nutzer:innen über das zu testende System erforderlich. Wir teilen das benötigte Wissen in drei Kategorien auf:

**Wissen über das Gesamtsystem** ist bei der Erstellung von Annotationen notwendig. Die Nutzer:innen müssen entscheiden welche Markierungen für ihr gewünschtes Ziel am besten zutreffende ist. Hierbei ist neben der gewünschten Detailstufe der Auswertung auch spezifisches Domänenwissen notwendig um den Kontext der Anforderungen korrekt zu interpretieren und die Annotationen speziell ab Level 3 korrekt zur erstellen.

**Wissen über Signalbedeutung** ist während des manuellen Abbildungsprozesses zur Erstellung der Abbildungstabelle notwendig. Hier werden die natürlichsprachlichen Textbausteine mit konkreten Signalen des Systems verknüpft. In komplexen Systemen kann die Anzahl der zur Verfügung stehenden Signalnamen sehr umfangreich sein. Je nach System ist der Aufwand entsprechend groß die Signalabbildung korrekt zu erstellen. Treten Fehler während des Abbildungsprozesses auf, folgt daraus eine Auswertung der falschen Signale und die fehlerhafte Ergebnisanzeige in den Ausgangsanforderungen.

**Wissen über die Logdaten** ist für die Erstellung und Interpretation der Annotationen notwendig. Auswertungen finden nur auf den vorhandenen Logdaten statt. Die angezeigten Ergebnisse resultieren aus der Analyse der Logdaten und spiegeln dadurch das Verhalten des Systems wider. Findet keine Aufzeichnung des Systemverhaltens statt, werden dieses nicht ausgewertet. In der Ergebnisanzeige findet nicht aufgezeichnetes Systemverhalten daher keine Berücksichtigung. Daraus lässt sich keine mangelnde Funktionalität des zu testenden Systems ableiten. Beispielsweise führt die Markierung des Begriffs *Tür* als $\text{State}_{\{L1\}}$ zu der Anzeige aller auftretenden Zustände. Sind in den Logdaten nur die Zustände *Fehler* und *bereit* vorhanden, jedoch keine Aufzeichnungen zu *geöffnet* oder *geschlossen*, obwohl diese während des Testlaufs eingenommen werden, so treten diese auch nicht in den Ergebnissen auf. Die Nutzer:innen könnten von einer Fehlfunktion ausgehen und den gesamten Testlauf falsch interpretieren. Es ist daher notwendig zu wissen, welche Auswertungen auf den Logdaten möglich sind und wie diese anschließend zu interpretieren sind.

Das benötigte Wissen der Nutzer:innen muss im praktischen Einsatz des Ansatzes nicht zwangsläufig in einer Person gebündelt sein. In den Testansatz können mehrere Personen involviert sein, die jeweils das Wissen für ihren Prozess, beispielsweise Annotationsprozess oder Abbildungsprozess, benötigen. Bei der Annotation der Anforderungen sind trotz des benötigten Wissens keine Einschränkungen bezüglich der Annotationen von Texten gesetzt. Die Auswahl der Annotationen, insbesondere die Wahl des Elements aus dem jeweiligen Level, liegt ohne Einschränkung bei den Nutzer:innen. Die Ergebnisse müssen unter Berücksichtigung der Auswertungsmöglichkeiten der einzelnen Elemente interpretiert werden.

## 3.4 TOOLUNTERSTÜTZUNG

Die grafischen Bestandteile des entwickelten Testansatz wurden in einem eigenständigen Prototypen umgesetzt, der Software Requirements and Test Manager (SRT-Manager) heißt. Die Kernfunktionalitäten sind die Annotation von Textphrasen in Softwarespezifikationen und die Darstellung der Ergebnisse aus den Systemausführungen. Der Prototyp wird von den Nutzer:innen im Annotationsprozess und im Darstellungsprozess des Testansatzes genutzt. Beide Prozessschritte nutzen unterschiedliche Funktionen.

**Annotationsprozess:** Im Annotationsprozess steht das Annotieren der Softwarespezifikation im Vordergrund. Die grafische Oberfläche integriert hierbei die Multilevel Markup Language und ermöglicht den Nutzer:innen das Markieren von Textpassagen und die Zuordnung zu den Elementen der Sprache. Die folgenden Funktionen stehen über

das Annotieren von Textpassagen hinaus für den Annotationsprozess zur Verfügung:

- Einlesen von Softwarespezifikationen im Comma-separated values (CSV)-Format
- Einlesen von Softwarespezifikationen eines DOORS-Exports
- Paralleles Öffnen von mehreren Softwarespezifikationen in Tabs
- Eingefärbte Darstellung erstellter Annotationen
- Browsen durch die Softwarespezifikation, gegebenenfalls mit annotierten Textpassagen
- Speichern von Annotationen je Level im CSV oder Extensible Markup Language (XML) Format
- Einlesen existierender Annotationen

Der gesamte Annotationsprozess kann mithilfe des Prototypen durchgeführt werden. Die Nutzer:innen können Softwarespezifikationen in das Tool einlesen, Textpassagen annotieren und anschließend die erstellten Annotationen extrahieren und abspeichern. Dadurch wird ein Zwischenspeichern ermöglicht, das den Nutzer:innen erlaubt denn Annotationsprozess zu unterbrechen und zu einem späteren Zeitpunkt fortzusetzen. Das XML Format dient als Speicherformat, um eine möglichst hohe Kompatibilität zu weiteren Tools oder weiteren Forschungsarbeiten zu ermöglichen.

**Darstellungsprozess:** Im Darstellungsprozess werden die Ergebnisse des Auswertungsprozesses gemeinsam mit den natürlichsprachlichen Softwarespezifikationen angezeigt. Die detaillierten Darstellungsmöglichkeiten werden in Kapitel 7.1 erläutert. Neben den für den Annotationsprozess verfügbaren Funktionen stehen den Nutzer:innen folgende Funktionalitäten zur Verfügung:

- Einlesen einer Ergebnisdatei im XML-Format, die alle Ergebnisse der Annotationen des Auswertungsprozesses enthält
- Browsen durch die Softwarespezifikation, gegebenenfalls mit annotierten Textpassagen, die abhängig vom verfügbaren Ergebnis eingefärbt sind
- Darstellung mehrerer Softwarespezifikationen mit eingelesenen Ergebnissen in mehreren Tabs

# 4

# TEIL 1: ANNOTATIONSPROZESS

In diesem Kapitel wird der erste Teil des Testprozesses beschrieben. Dieser befasst sich mit der manuellen Erstellung der Annotationen durch die Nutzer:innen und der teilautomatisierten Erkennung und Annotation von Zuständen in Anforderungen. Kapitel 4.1 zeigt die manuelle Erstellung von Annotationen anhand eines Beispiels aus dem Automotive Kontext. Anschließend wird die praktische Umsetzung des Annotationsprozesses anhand des entwickelten Prototypen beschrieben. Die Annotierung von Anforderungen kann durch automatische Erkennung von Elementen in Anforderungen unterstützt werden. Kapitel 4.2 beschreibt zwei Ansätze zur Erkennung von $\text{State}_{\{L1\}}$-Elementen. Darüber hinaus wird der eingesetzte selbstlernende Algorithmus beschrieben, der für weiter Automatisierungsschritte angewendet werden kann.

Dieses Kapitel basiert teilweise auf dem bereits veröffentlichten Beitrag von Pudlitz et al. [74].

## 4.1 MANUELLE ERSTELLUNG VON ANNOTATIONEN

Das Markieren von Textphrasen in natürlichsprachlichen Anforderungen wird manuell von den Nutzer:innen durchgeführt. Das ermöglicht einen flexiblen Einsatz der Annotationen für unterschiedliche Ziele.

Die *Überwachung und das Testen von Softwarefunktionen*, der *Vergleich von Teststufen* und die *Erstellung von Testanforderungen* sind die drei wesentlichen Ziele, die durch die Annotation erfüllt werden sollen.

**Überwachung und Testen von Softwarefunktionen** ist ein wesentlicher Aspekt im Bereich der Softwareentwicklung. Ziel ist eine möglichst genaue und präzise Aussage über das Softwareverhalten in allen vorkommenden Situationen der eingesetzten Softwarefunktion. Die erstellten Annotationen ermöglichen die Bereitstellung detaillierten Feedbacks des Softwareverhaltens. Die Einbettung der Testergebnisse in die Anforderungsdokumente gibt den Nutzer:innen Auskunft über erfüllte und nicht erfüllte Anforderungen begleitet von Analysen zum Systemverhalten während der Systemausführung.

Der **Vergleich von Teststufen** ist besonders in großen und komplexen Softwareprojekten bedeutsam. Hier findet das Testen auf mehreren Teststufen statt, die unterschiedliche Aspekte der Systemfunktionalität untersuchen. Die Teststufen der Softwareentwicklung orientieren

sich an dem vorgestellten V-Modell aus Kapitel 2.2 und müssen von der Software durchlaufen werden. Der Vergleich der Ergebnisse aus den Teststufen gibt Aufschluss über das Auftreten von Fehlern. Beispielsweise kann eine Simulation vollständig erfolgreich durchgeführt werden, während die gleiche Software in einer Realfahrt zu Problemen führt. Dies kann durch den Abstraktionsgrad in der Simulation begründet sein, beispielsweise wenn kein realer Verkehr simuliert wurde. Durch die erstellten Annotationen können die fehlerhaften Szenarien identifiziert werden. Die Nutzer:innen entscheiden durch die Erstellung der Annotationen, welche Aspekte der Anforderungen verglichen werden.

Die **Erstellung von Testanforderungen** gewinnt durch die wachsende Komplexität der Softwarefunktionalität zunehmend an Bedeutung. Sie verhindert durch die Vielzahl an möglichen Szenarien die vollständige Testabdeckung [104]. Daher gibt es verschiedene Ansätze gezielt Testszenarien zu erzeugen ([5, 69, 70]), die in den unterschiedlichen Teststufen die Softwarefunktionalität prüfen. Es fehlen jedoch Ansätze zur Identifizierung von Testeigenschaften, die eine Grundlage zur systematischen Erstellung sind. Durch die Möglichkeit Annotationen in Anforderungen zu erstellen, können relevante Beschreibungen, beispielsweise zur Umwelt oder zu anderen Systemen, markiert und anschließend extrahiert werden. Auf der Basis dieser extrahierten Annotationen können Testszenarien erstellt werden, die die gestellten Anforderungen abbilden.

### 4.1.1 *Annotation von Anforderungen*

Die Multilevel Markup Language erleichtert das Hervorheben wichtiger Informationen und die Beobachtung der markierten Textpassagen in Systemausführungen mit einem flexiblen Detaillierungsgrad. Die Annotationen werden manuell von Nutzer:innen erstellt und variieren daher je nach Anwendungsfall und individuellen Zielen der Nutzer:innen. Dieses Kapitel zeigt exemplarisch die Einbettung der Markierungen in die natürlichsprachlichen Systemanforderungen an einem Beispiel aus dem Automotive Kontext.

Abbildung 4.1 zeigt drei Beispielanforderungen (B-1, B-2, B-3) einer Systemspezifikation des Totwinkel-Assistenten (TA)[1]. Die annotierten Textpassagen sind jeweils unterstrichen dargestellt. Unterhalb der Anforderungen werden die Details der Annotation wie Level, Typ und gegebenenfalls Annotationswerte mit Operatoren angezeigt.

1 Das System warnt den Fahrer vor anderen Verkehrsteilnehmern im toten Winkel. In seinem Fahrzeugen leuchtet eine rote Warnleuchte im Außenspiegel. Bei zeitgleicher Betätigung des Blinkerhebels ertönt zusätzlich ein Warnton. Die rote Warnleuchte blinkt währenddessen.

Softwarespezifikation: Totwinkel Assistent

| ID | Object Text |
|---|---|
| B-1<br>L1/L2 | Der TA ist für Geschwindigkeiten zwischen 0 km/h und 150 km/h verfügbar.<br>L1 TA: System<br>L2 Geschwindigkeiten: $Value_{System}$ |
| B-2<br>L3 | Nähert sich ein Fahrzeug von hinten an das Systemfahrzeug und befindet sich in einem Abstand von 3 m, erreicht der TA die Warnstufe 1.<br>L3 $Value_S$ Abstand ≤ 3 m<br>L3 $State_S$ TA = Warnstufe 1 |
| B-3<br>L3/L4 | Ist der TA in Warnstufe 2 und der Blinker wird auf der gleichen Seite betätigt während ein anderes Fahrzeug erkannt wird, ertönt ein akustisches Warnsignal.<br>L4 Trigger: TA = Warnstufe 2 && Blinker = betätigt<br>Action: akustisches Warnsignal = ertönt<br>L3 $State_S$ Fahrzeug = erkannt |

Abbildung 4.1: Beispielanforderungen eines Totwinkelssistenten mit Annotationen auf allen vier Level.

B-1 enthält Level 1 und Level 2 Annotationen. Die *System*-Annotation von `TA` zeigt die Systemzugehörigkeit dieser Textpassage und ermöglicht die Analyse, ob ein entsprechendes Signal in der Systemausführung vorhanden ist. Dies kann je nach der verwendeten Simulation oder dem Entwicklungsstand des Systems variieren. Mit der Level 2 Annotation $Value_{\{System\}}$ wird die Textpassage `Geschwindigkeit` semantisch zu einem Begriff mit kontinuierlichem Verlauf. Das ermöglicht eine Darstellung der Werte über den gesamten Testzeitraum.

In B-2 wurden alle Annotationen bis zu Level 3 annotiert. Die Wörter `Abstand` und `TA` sind jeweils mit einem bestimmten Wert aus der Anforderung verknüpft. Zu diesem Zweck wird ein Operator ausgewählt, der je nach Typ definiert wird. Für Zustände ist es möglich, $=$ und $\neq$ zu verwenden, für Werte die zusätzlichen Vergleichsoperatoren $<, \leq, >$ und $\geq$.

B-3 enthält eine Level 3 und eine Level 4 Annotation. Die Level 3 Annotation besteht aus dem Bedingungsnamen `Fahrzeug` und dem zugehörigen Bedingungswert `erkannt`. Sie sind mit einem Gleichheitsoperator verknüpft. Level 4 Annotationen setzen sich aus mehreren Level 3-Annotationen zusammen. Die Level 4-Annotation in B-3 zeigt, dass zwei Level 3 Annotationen (`TA = Warnstufe 2` und `Blinker = betätigt`) als Auslöser verwendet werden. Der Begriff `akustische Warnung` ist ebenfalls eine Annotation des Level 3. Trigger, Pre-Condition und Action können jeweils aus einer Liste von Level 3 Annotationen bestehen. Diese werden von den Nutzer:innen durch *AND* oder *OR* verknüpft und später entsprechend ausgewertet.

### 4.1.2 *Weitere Eigenschaften der Multilevel Markup Language*

Die Erstellung von Annotationen ist bewusst ein manueller Prozess, der den Nutzer:innen die Entscheidung über den Detaillierungsgrad der Auswertung ermöglicht. Weitere Eigenschaften der Multilevel Markup Language sind im Folgenden aufgeführt.

Die **levelunabhängige Erstellung von Annotationen** ist durch die direkte Annotierung auf Level 1 bis Level 3 möglich. Es ist nicht zwingend notwendig auf Level 1 zu beginnen und levelweise vorzugehen. Die Nutzer:innen können Texte in Anforderungen direkt mit Elementen höherer Level markieren. Abhängig vom Element des gewählten Levels werden alle Typen (Level 2) automatisch der Annotation zugewiesen. Die Nutzer:innen müssen lediglich das passende Level 1 Element auswählen. Ein Spezialfall ist Level 4, da dieses Level ausschließlich aus Elementen des Level 3 besteht, müssen diese Annotationen zuvor existieren. Die Erstellung von Level 4 Annotationen setzt mindestens zwei Level 3 Annotationen voraus.

Das Erstellen der Annotationen aus mehreren Textpassagen unterschiedlicher Anforderungen ist durch die **anforderungsübergreifende Annotierung** möglich. Die Nutzer:innen können Textpassagen entweder innerhalb einer oder aus unterschiedlichen Anforderungen auswählen, um eine Beziehung zwischen ihnen herzustellen.

Es ist möglich ein annotiertes Anforderungsdokument zu erstellen, welches sowohl eine **diverse Levelverteilung innerhalb des Dokuments** als auch eine **diverse Levelverteilung innerhalb einer Anforderung** aufweist. Diese Flexibilität ermöglicht den Nutzer:innen eine an die Teststrategie angepasste Anforderungsannotierung. Abhängig vom Entwicklungsstand der zu testenden Software, der eingesetzten Testverfahren im Ausführungsprozess oder der beschriebenen Zielen der Nutzer:innen können Annotationen in einem Anforderungsdokument erstellt werden.

In einem Anforderungsdokument können Textpassagen mehrfach auftretet. Um doppelte Markierungen der Nutzer:innen zu verhindern und um die Übersicht vorhandener Annotationen zu erleichtern, werden Annotationen aus Level 1 bis Level 3 für das gesamte Dokument übernommen und den Nutzer:innen angezeigt. Diese **indirekte Erstellung der globalen Annotationen** zeigt die Auswertungen der Testdurchläufe im gesamten Dokument und setzt die angezeigten Ergebnisse auch ohne erneute manuelle Annotierung in den jeweiligen Kontext.

## 4.2 STUDIE ZUR AUTOMATISIERTEN EXTRAKTION VON ELEMENTEN

Die Annotationen in einer Anforderungsspezifikation werden von Nutzer:innen manuell erstellt. Das ermöglicht einen flexiblen Einsatz

der Multilevel Markup Language. Diese kann abhängig vom Entwicklungsstand der zu testenden Software und den jeweiligen Zielen der Nutzer:innen unterschiedlich variabel innerhalb einer Anforderungsspezifikation angewendet werden. Der Umfang der erstellten Annotationen ist abhängig von der Komplexität der Software und entsprechend aufwendig. Um den Annotationsprozess zu unterstützen, wurde ein automatisierter Ansatz zur Extraktion von Elementen aus der Multilevel Markup Language entwickelt. Dieser verknüpft Techniken des maschinellen Lernens mit minimalem Aufwand der Nutzer:innen und stellt ein Modell zur Verfügung, welches im Annotationsprozess eingesetzt werden kann. Für das Training des Algorithmus werden zwei Ansätze mit unterschiedlichem Aufwand für die Nutzer:innen verglichen. Die resultierenden Ergebnisse werden anschließend anhand der Extraktion von $\text{State}_{\{L1\}}$-Elementen diskutiert.

### 4.2.1 *Kontext*

Die automatisierte Erkennung von Elementen der Multilevel Markup Language bietet eine erhebliche Zeitersparnis bei der Annotierung von Anforderungen. Besonders die Elemente der Level 1 und Level 2 sind überwiegend kontextunabhängig und können unabhängig von den Zielen der Nutzer:innen identifiziert werden. Im Gegensatz dazu sind die Elemente der Level 3 und Level 4 mehr vom Entwicklungsstand der Software abhängig. Der manuelle Annotationsprozess wird zusätzlich durch die automatisierte Analyse der Anforderungen unterstützt und erleichtert oder beschleunigt die Erstellung von Annotationen.

Die automatisierte Analyse von natürlichsprachlichen Anforderungen ist ein stetig wachsendes Forschungsfeld [23, 56, 84]. Neben der Formalisierung von Anforderungen ([54]) und der Generierung von Testfällen ist die Extraktion von Informationen Fokus vieler Forschungsarbeiten. Insbesondere in den letzten Jahren haben sich Methoden des maschinellen Lernens zunehmend etabliert [24, 59, 105]. Für den Einsatz dieser Methoden und Algorithmen muss das eingesetzte Modell zuvor trainiert werden. Für die Erkennung und Extraktion von Elementen der Multilevel Markup Language aus natürlichsprachlichen Anforderungsdokumenten liegen derzeit keine trainierten Modelle vor. Das Training benötigt vormarkierte Daten, mit derer der Algorithmus die Erkennung erlernt. Dieser Prozess ist manuell sehr zeitaufwendig.

Wir vergleichen zwei Ansätze um Trainingsdaten für einen maschinellen Lernansatz zu erzeugen. In beiden Ansätzen werden die Nutzer:innen unterschiedlich stark eingebunden und die Auswirkungen auf die Erkennungsqualität untersucht. Für die Evaluation des Trainingsansatzes und das daraus resultierende Modell stehen die $\text{State}_{\{L1\}}$-Elemente der Multilevel Markup Language im Fokus der Untersuchung. Software im Allgemeinen und Fahrerassistenzsyste-

me in Fahrzeugen im Speziellen arbeiten zustandsbasiert und stehen daher auch im Fokus in Anforderungsdokumenten. In der automatischen Erkennung der Zustände liegt daher das größte Potenzial zur Unterstützung der Nutzer:innen im Annotationsprozess.

Für die automatische Erkennung der $\text{State}_{\{L1\}}$-Elemente in Anforderungen ergeben sich zwei wesentliche Vorteile im Gesamttestprozess. Zum einen können die Nutzer:innen ein Anforderungsdokument automatisiert vorannotieren lassen. Hierbei werden alle $\text{States}_{\{L1\}}$ aus Level 2 automatisch annotiert und im Lastenheft farblich dargestellt. Darauf aufbauend können die Nutzer:innen weitere Annotationen erstellen oder vorhandene Annotationen anpassen. Zum anderen besteht die Möglichkeit, die automatisch erkannten $\text{States}_{\{L1\}}$ unverändert für die anschließenden Teilprozesse (Abbildungsprozess, Auswertungsprozess, Darstellungsprozess) zu nutzen. Diese schnelle und leichtgewichtige Art Teile eines Anforderungsdokuments zu evaluieren, erspart den Nutzer:innen Zeit und personellen Aufwand.

Die manuelle Erstellung von $\text{States}_{\{L1\}}$ ermöglicht die uneingeschränkte, individuelle Auswahl von Textpassagen. Für die automatische Erkennung und anschließende Evaluation des Trainings- und Erkennungsprozesses wurde der zu erkennende Zustand genauer definiert.

#### 4.2.1.1 *Definition von Zuständen*

Systemzustände beschreiben das veränderbare Verhalten eines Systems. Je nach Sichtweise, Domäne oder System können diese Zustände unterschiedlich definiert sein. Die Definition eines Zustands kann ferner von seiner spezifischen Verwendung abhängen. Mit dem Fokus auf Software und der Unterscheidung zwischen Laufzeit und Kompilierzeit sind nicht alle änderbaren Eigenschaften einer Software als Zustände zu definieren. Parameter, Merkmale und Designentscheidungen können vor jedem Durchlauf geändert werden, bleiben aber zur Laufzeit konstant. Systemzustände ändern sich während der Laufzeit und haben zur Kompilierzeit eine feste Menge möglicher Werte. Formal ist der $\text{State}_{\{L1\}}$ als Funktion definiert:

$$State_{\{L1\}} := Name_x \rightarrow \{Value_{xy}\}$$

Aus der Definition geht hervor, dass einem Zustandsnamen immer eine Menge von Zustandswerten zugeordnet werden (die leere Menge ist hierbei ausgeschlossen), die in den Anforderungen explizit enthalten sind. Das alleinige Vorkommen des Namens ist nicht ausreichend um der Definition eines Zustandes zu entsprechen. Wenn jedoch die Zuweisung eines Zustandsnamen zu einem Zustandswert einmalig in der Anforderungsspezifikation enthalten ist, wird jedes Vorkommen dieses Zustandsnamens in der Spezifikation erkannt und entsprechend

gekennzeichnet. Die Zustandswerte beschreiben immer diskrete Merkmale und zeigen keinen kontinuierlichen Verlauf. Sie beschreiben Änderungen, die im Testprozess beobachtbar sind. Tabelle 4.1 listet zwei beispielhafte Anforderungen mit kurzer Erklärung auf. Darunter befinden sich Zustände und Texte, die nicht der Zustandsdefinition entsprechen.

Tabelle 4.1: Beispiel einer Zustandsdefinition

| **Anforderung** | **Zuordnung** | **Zustand** |
|---|---|---|
| Der Startknopf startet und stoppt den Motor. | $Name_1 = Motor$<br>$Wert_{1,1} = startet$<br>$Wert_{1,2} = stoppt$ | Ja<br>$Motor \rightarrow \{startet, stoppt\}$ |
| | $Name_2 = Startknopf$<br>$Wert_{2,1} = \{\}$ | Nein, konkreter Wert fehlt |
| Ab einer Geschw. über 150 km/h schaltet sich das System aus. | $Name_3 = Geschw.$<br>$Wert_{3,1} = \{\}$ | Nein, kontinuierlicher Verlauf |
| | $Name_4 = System$<br>$Wert_{4,1} = aus$ | Ja<br>$System \rightarrow \{aus\}$ |

Die Beispiele zeigen zwei Zustandsnamen (*Motor*, *System*) mit entsprechenden Werten. Im Gegensatz dazu werden die Begriffe *Startknopf* und *Geschw.* nicht als Zustände bezeichnet, da sie nicht der Definition entsprechen. Diesen Zustandsnamen werden keine Zustandswerte zugeordnet. Jedoch wird das Vorkommen der Zustandsnamen *Motor* und *System* im übrigen Anforderungsdokument automatisiert als Zustand gekennzeichnet, unabhängig vom Vorhandensein von entsprechenden Werten in der jeweiligen Anforderung.

In natürlichsprachlichen Anforderungen kann es zur Verwendung von Synonymen kommen. Diese werden in der Erkennung als unabhängige Zustände betrachtet und entsprechend der Definition identifiziert.

Die hier verwendete Definition unterscheidet sich von der Arbeit von Vogelsang et al. [94]. Zwar wird die Identifikation eines Zustands durch die Zuordnung eines spezifischen Werts zu einem Namen vorausgesetzt. Dieser Wert wird jedoch nicht in die Zustandsdefinition aufgenommen. Darüber hinaus wird jede Nennung eines einmalig identifizierten Zustandsnamens in den Anforderung als Zustand definiert. Vogelsang et al. [94] berücksichtigt nur die explizite Kombination

von Name und Wert. Daher würde in dessen Arbeit, der Zustand *Motor* in einer anderen Anforderung, ohne explizit erwähnte Werte, nicht als ein Zustand identifiziert werden. Die Entscheidung, ob ein Begriff als Zustand angesehen wird, ist für die Nutzer:innen subjektiv. Im Vergleich dazu ist unsere Definition strenger, da wir zuweisbare Werte verlangen, damit ein Zustand als solcher identifiziert werden kann.

### 4.2.2 *Zielsetzung*

Ziel der automatisierten Erkennung ist die Identifikation von Zustandsnamen in natürlichsprachlichen Anforderungen. Hierbei vergleichen wir zwei Trainingsansätze mit unterschiedlichem Aufwand für die Nutzer:innen. Beide Ansätze erzeugen Trainingsdaten für einen maschinell lernenden Erkennungsalgorithmus und werden anschließend auf ein Anforderungsdokument aus dem Automobilbereich angewendet.

Für die Analyse stehen zwei wesentliche Fragestellungen im Vordergrund. Wie gut lassen sich Zustandsnamen mithilfe von maschinell lernenden Erkennungsalgorithmen identifizieren? Und wie wirkt sich der Aufwand für die Nutzer:innen bei der Erstellung der Trainingsdaten auf die Qualität der Erkennung aus?

Ein wesentlicher Vorteile von selbst trainierten Ansätzen ist die Adaptierbarkeit an die zu erkennenden Objekte. Da derzeit keine vortrainierten Algorithmen zur Erkennung von Zustandsnamen existieren, können die eingesetzten Algorithmen auf die verwendeten Anforderungsdokumente optimiert werden. Die vorgestellten Trainingsansätze sind ebenso auf weitere Elemente der Multilevel Markup Language übertragbar und können so durch die zusätzliche Automatisierung weitere Erleichterung für die Nutzer:innen bringen.

### 4.2.3 *Methodik*

Wir untersuchen zwei Ansätze zur Extraktion von Zustandsnamen mit unterschiedlichem Grad an Automatisierung und Interaktion der Nutzer:innen. Abbildung 4.2 und Abbildung 4.3 visualisieren schematisch den halbautomatischen und den nahezu automatisierten Ansatz. Als Ausgangspunkt der Ansätze wird ein kleiner Teil verschiedener Ausgangszustandsnamen benötigt. Diese werden durch manuelles Scannen von wenigen zufälligen Anforderungen nach enthaltenen Zustandsnamen erstellt. Das Ergebnis dient anschließend als Ausgangspunkt für die weitere Verarbeitung.

Der erste Ansatz bezieht die Anwender:innen iterativ in den Trainingsprozess ein. Zunächst werden die Trainingsdaten in mehrere Teile aufgeteilt. Mit den Ausgangszuständen, wie oben erwähnt, werden die relevanten Datensätze aus der ersten Trainingsuntermenge extrahiert und für das Training des NER-Modells verwendet. Das

trainierte Modell wird dann angewendet, um Zustände in der zweiten Teilmenge der Trainingsdaten zu erkennen. Die erkannten Zustände werden den Nutzer:innen zur Überarbeitung angezeigt. Die überarbeitete und so reduzierte Liste der erkannten Zustandsnamen wird dann verwendet um einen annotierten Trainingssatz aus der Kombination der ersten und zweiten Teilmenge der Trainingsdaten zu erstellen. Dieses Verfahren wird wiederholt, bis die Gesamtanzahl der aufgeteilten Trainingsdaten erreicht ist.

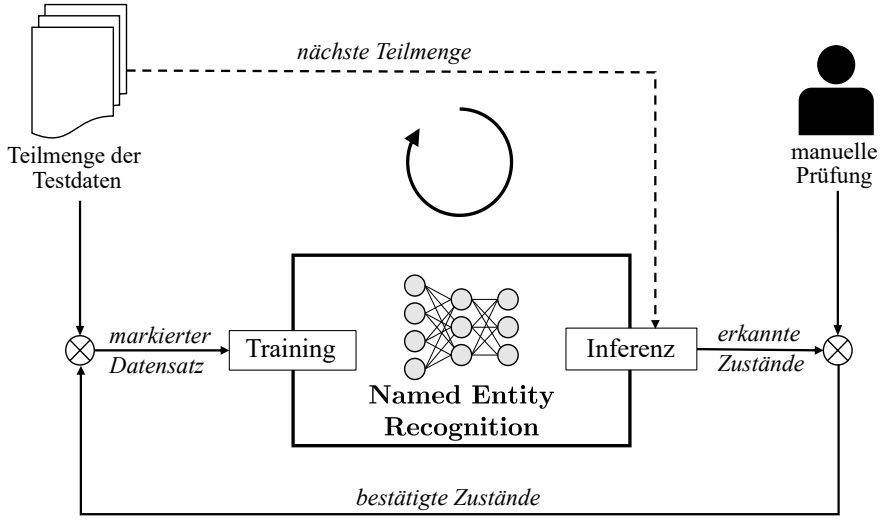



Abbildung 4.2: Schematische Übersicht des halb-automatisierten Ansatzes

Der zweite Ansatz ist fast vollständig automatisiert und erfordert nur minimale Interaktion der Nutzer:innen. Aus Gründen der Lesbarkeit wird dieser Ansatz im weiteren Verlauf dieser Arbeit als automatisiert bezeichnet. Die identifizierten Ausgangszustandsnamen werden verwendet, um einen Bootstrapper, der in 4.2.3.1 genauer erläutert wird, zu initialisieren. Dieser Bootstrapper erkennt weitere Zustände in den Trainingsdaten. Die resultierende Liste der Zustandsnamen wird von den Nutzer:innen überprüft, wobei alle Phrasen, die nicht der Zustandsdefinition entsprechen, gelöscht werden. Die konditionierten Zustandsnamen werden zur Erstellung eines Trainingssatz für das Modell eines Named-Entity Recognizers verwendet, der in 4.2.3.2 erläutert wird. Alle Anforderungen im Trainingsset, die einen oder mehrere der Zustände enthalten, werden extrahiert. Diese Daten stehen dem einmaligen Training des NER-Modells zur Verfügung.

#### 4.2.3.1 *Bootstrapping Algorithmus*

Für viele Anwendungen, die auf dem Training von Modellen basieren, ist eine große Anzahl von Trainingsdaten für eine valide Auswertung der Funktion erforderlich. Für spezielle Anwendungen, bei denen keine Trainingsdaten vorhanden sind, müssen zunächst Trainingsdaten generiert werden. Dieser Prozess kann mit Bootstrapping-Algorithmen automatisiert werden. Mit einem Bootstrappingansatz kann ein großes

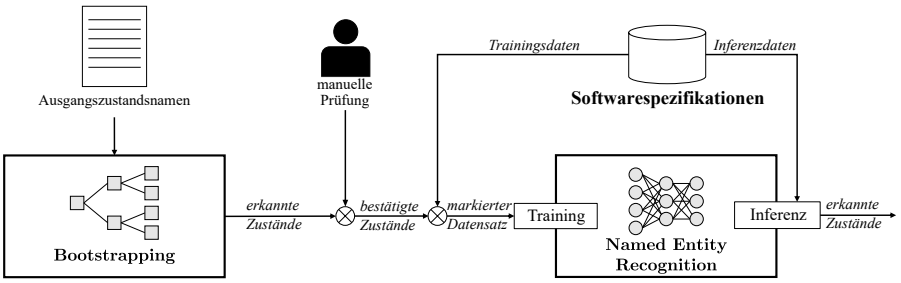



Abbildung 4.3: Schematische Übersicht des automatisierten Ansatzes

Trainingsset mit einem kleinen Teil von Ausgangsdaten zusammengestellt werden. Die resultierenden Daten werden als Input für die weitere Verarbeitung verwendet.

In dieser Arbeit wird der Algorithmus von Gupta et al. [39] auf Anforderungen aus dem automobilen Kontext angewendet. Der Ansatz extrahiert Einträge nach gelernten Mustern in Texten. Er wird zur automatischen Erkennung von Zustandsnamen anhand von Mustern, die in Fahrzeuganforderungen vorkommen, angewendet. Die Liste der erkannten Zustandsnamen wird als Input für die darauffolgende Named-Entity Recognition verwendet. Ausgangspunkt für die Erkennung ist eine kleine Auswahl von Zustandsnamen, die durch manuelle Analyse von zufällig ausgewählten Anforderungen extrahiert werden. Für die Erkennung von Zustandsnamen in allen Anforderungen mit dem Bootstrapping-Algorithmus ist die Extraktion dieser Ausgangszustände der einzige manuelle Schritt. Danach durchläuft der Algorithmus die folgenden drei Schritte auf vollständig automatisierte Weise:

**Erkennung von Mustern:** Alle zuvor identifizierten Ausgangszustandsnamen werden in allen Anforderungen markiert. Jede Anforderung die einen markierten Zustandsnamen enthält, wird auf ein vorhandenes textuelles Muster untersucht. Um ein Muster zu erkennen, wird der Kontext um das markierte Wort in einem Bereich von beispielsweise zwei bis vier Wörtern vor und nach dem Zielwort untersucht. Die folgenden Muster veranschaulichen die Auswahl des untersuchten Textes, wobei **X** dem untersuchte Zustandsnamen entspricht:

*der Status des* ***X*** oder *das Signal, das den* ***X*** *beschreibt*

**Bewertung der Muster:** Die erkannten Muster werden bewertet. Für die Berechnung der Bewertungsfunktionen werden unterschiedliche Faktoren (Features) in Betracht gezogen. Unter anderem werden die Begriffe in Vektoren umgewandelt und die Distanzen zu bekannten Zustandsnamen (positive Distanz) und zu nicht zutreffenden Begriffen (negative Distanz) berechnet. Ein weiterer Faktor ist das Clustern (Gruppieren) von ähnlichen Begriffen. Alle Begriffe aus dem selben

Cluster wie der Zustandsname erhalten eine höhere Punktzahl. Für weitere Details sowie die genaue Formel der Bewertungsfunktion soll an an dieser Stelle auf die Arbeit von Gupta et al. [39] verwiesen werden, da die Entwicklung beziehungsweise Optimierung des Bootstrapping Ansatzes nicht im Fokus dieser Arbeit steht. Die am besten bewerteten Muster werden in die Auswahl der gelernten Muster aufgenommen.

**Erkennung von Zustandsnamen:** Mit den neu gelernten Mustern werden neue Kandidaten für Zustandsnamen in den Texten identifiziert und anschließend bewertet. Ein Bewertungsalgorithmus erstellt eine Rangliste der Kandidaten und fügt die besten Begriffe einer Liste hinzu.

Diese drei Schritte werden iterativ wiederholt. In jedem Schritt erfolgt eine Erkennung neuer Zustandsnamen und Muster. Der Algorithmus beendet die Analyse, wenn keine neuen Muster erkannt werden oder die festgelegte Anzahl von Iterationen erreicht ist. In unserer Anwendung erkennt der Bootstrapping-Ansatz automatisch Zustandsnamen in den Anforderungsdaten mit geringer Interaktion der Nutzer:innen. Die Nutzer:innen müssen zum Start eine kleine Anzahl von Ausgangszuständen festlegen. Um die Qualität dieser Ausgangsdaten zu erhöhen, integrieren wir in den Prozess einen Domänenexperten, der die erkannten Zustandsnamen des Bootstrappers einmalig bewertet. Das genaue Beurteilungsverfahren wird in Kapitel 4.2.3.4 näher erläutert.

#### 4.2.3.2 *Named Entity Recognition Model*

Das in dieser Arbeit verwendete Modell der NER basiert auf der Arbeit von Chiu et al. [16] und wird mit den zuvor selbsterstellten Trainingsdaten trainiert. Die allgemeine Architektur ist in Abbildung 4.4 dargestellt.

Das Modell integriert sowohl `word` und `character embeddings` in die Darstellung des Inputs. Die word embeddings sind nach dem Skip-Gramm-Modell, wie es von Mikolov et al. [68] vorgestellt wird, vortrainiert. Da die Domäne der Automobilanforderungen über einen speziellen Satz von Begriffen und Phrasen verfügt, wird die Einbettung an den Anforderungen selbst trainiert. Zusätzlich wird die Einbettung während des Trainings des NER-Modells angepasst.

Die Zeicheneinbettung wird mit Zufallsvektoren für jedes Zeichen initialisiert und erst während der Ausbildung am NER-Modell angepasst. Um die zu einem Wort gehörenden mehrfachen Zeicheneinbettungen zu reduzieren, wird ein Convolutional Neural Network (CNN) verwendet. Die Zeichenanzahl im Netzwerk ist einheitlich festgelegt, sodass jedes Wort aufgefüllt oder gekürzt wird. Mit einer Faltungs- und einer Poolingschicht wird diese Matrix von Zeicheneinbettungen auf einen einzigen Vektor reduziert, der der Zeichenzusammensetzung der Wörter entspricht.

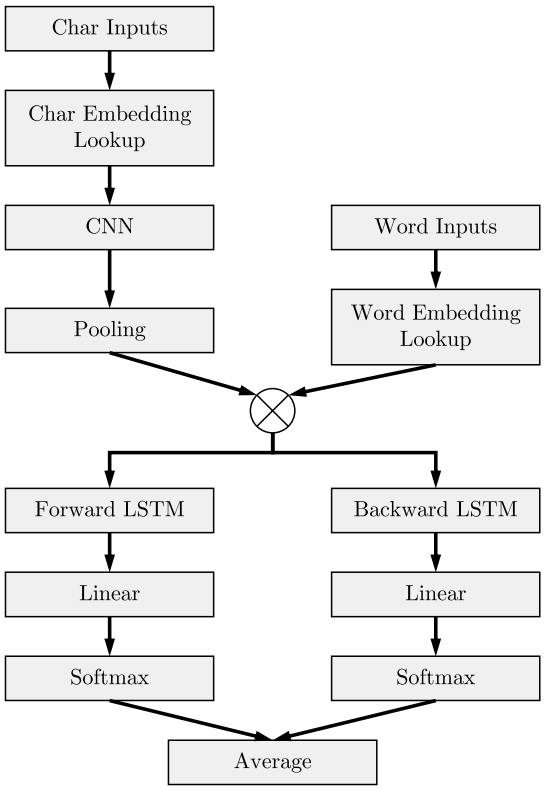



Abbildung 4.4: Layerstruktur des Named Entity Recognition Modells

Die beiden Eingaben werden dann für jedes Wort verkettet, um eine Sequenz von Wortvektoren für jede Eingabeanforderung zu erzeugen.

Das eigentliche Modell der NER wird mit einem Bidirectional Long short-term memory (BiLSTM) implementiert. Diese Netzwerkschicht verarbeitet die Eingabesequenz in Vorwärts- und Rückwärtsrichtung. Diese Berechnung berücksichtigt die sequentiellen Eigenschaften der Sätze sowie die inhärenten semantischen Verbindungen zwischen den Wörtern. Zusätzlich werden aufgrund der Vorwärts- und Rückwärtsverarbeitung die Verbindungen zwischen den Wörtern in Bezug auf den vorhergehenden und nachfolgenden Kontext berücksichtigt [36]. Die Vorwärts- und Rückwärtsberechnungen resultieren jeweils in einer neuen Sequenz in gleicher Länge wie die Eingabe. Für die Vorwärts- und Rückwärtsverarbeitung wird diese Ausgabesequenz dann einer linearen, vollständig verbundenen Schicht zugeführt, um ein Tupel pro Wort in der Eingabesequenz zu erzeugen, das die entsprechende Kategorie des Wortes repräsentiert. In dieser Anwendung ist diese Ausgabe ein Tupel, da das Modell nur zwischen Zuständen und Nicht-Zuständen unterscheidet. Um Beschriftungen zwischen 0 und 1 zu erzeugen, wird ein Softmax auf die Tupel der Sequenz angewendet. Zuletzt werden die beiden vorwärts und rückwärts verarbeiteten Tupel durch Mittelwertbildung ihrer jeweiligen Werte kombiniert. Diese letzte Schicht weicht dem vorgestellten Modell aus der Arbeit von Chiu et al. [16] ab, da sie sich für die Anwendung dieser Arbeit besser

eignet. Bei dieser letzten Berechnung werden sowohl die Vorwärts- als auch die Rückwärtsverarbeitung der Eingabe gleich gewichtet und kombiniert, um eine Ausgabe zwischen 0 und 1 zu erzeugen. Die einzelnen Ergebnisse der Vorwärts- und Rückwärtsberechnung werden nicht analysiert.

Die Ausgabe des Netzes ist ein Tupel, wobei das erste Element dem Wort *Zustandsname* und der zweite Wert dem Wort *kein Zustandsname* entspricht. Im Vergleich zu diesem Vorgehen schnitt ein Modell mit Vorhersage nur eines Werts mit 1 - Zustandskennzeichen und 0 - kein Zustandszeichen deutlich schlechter ab.

#### 4.2.3.3 *Parametrisierung des Trainingsprozesses*

Das NER-Modell hat mehrere Hyperparameter, die in Tabelle 4.2 dargestellt sind. Diese werden sowohl im Verlauf der Trainingsiterationen als auch für den automatisierten Ansatz auf konstante Werte gehalten.

Tabelle 4.2: Parameter des NER Modells

| **Parameter** | **Wert** | **Parameter** | **Wert** |
|---|---|---|---|
| LSTM size | 275 | Optimizer | Adam |
| LSTM dropout | 0.68 | Epochs | 50 |
| Maximum sentence length | 50 | Batch size | 5 |
| Maximum word length | 15 | Learning rate | 0.002 |
| Detection threshold | 0.3 | | |

Die maximale Anzahl der Epochen ist auf 50 begrenzt. Während des Trainings wird das Modell durch einen vorzeitigen Abbruch ergänzt: das Training wird abgebrochen, wenn sich der Validierungsverlust im Verlauf der letzten fünf Epochen nicht verbessert hat. Die Lernrate wurde empirisch ermittelt. Abgesehen von der Batchgröße entsprechen die übrigen Parameter der Konfiguration aus dem Ansatz von Chiu et al. [16]. Die Batchgröße wurde mit geringeren Werten als empfohlen festgesetzt, da es - insbesondere bei frühen Iterationen - weniger Trainingsstichproben als bei vergleichbaren Datensätzen gibt.

Die Parameter für die maximale Satz- und Wortlänge werden so festgelegt, dass so viele Informationen wie möglich einbezogen werden und dennoch eine Berechnung möglich ist. Um eine angemessene Grenze zu bestimmen, werden die Histogramme der Stichproben- und Wortlängen in Abbildung 4.5 analysiert. Durch das Kürzen längerer Stichproben entsprechend der angegebenen Maximallänge werden 93,94 % der Daten verwendet. Die maximale Wortlänge schränkt die Anzahl der Zeichen ein, die von der Zeichenkodierung CNN aufgenommen werden. Der angegebene Wert macht 99,48 % aller Zeichen in den Daten aus.

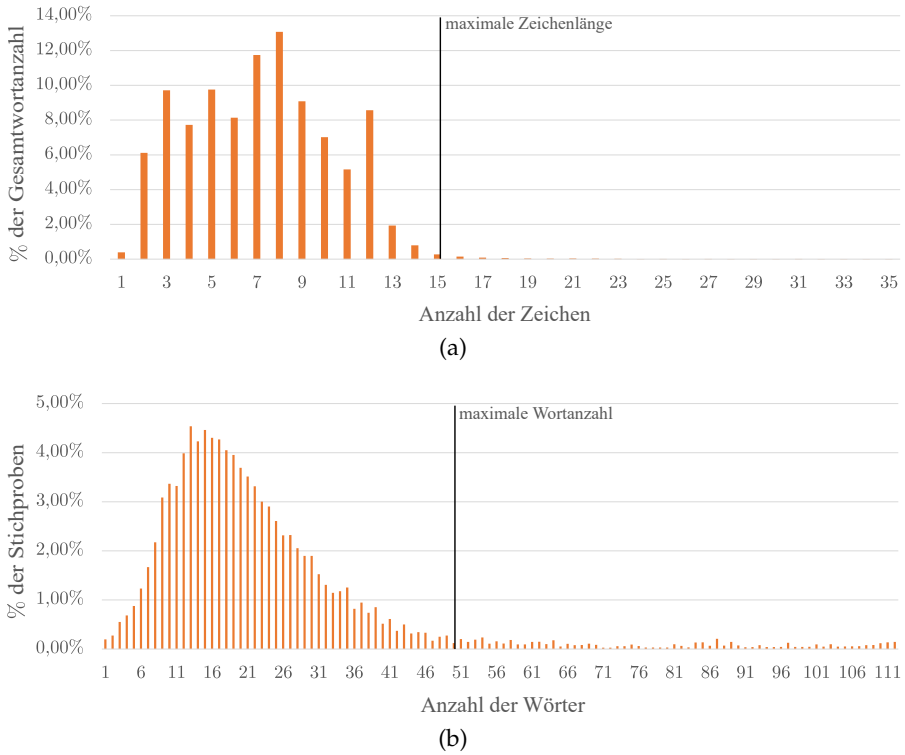



Abbildung 4.5: Histogramm über die Anzahl von Zeichen je Wort (a) und Anzahl der Wörter je Anforderung (b)

Der Erkennungsschwellenwert definiert die Grenze, ab der die Modellausgabe als Zustandsangabe interpretiert wird. Der Wert wird empirisch festgelegt, indem die Precision und der Recall auf dem Testsatz für wechselnde Werte beobachtet werden.

#### 4.2.3.4 *Interaktion der Nutzer:innen*

Beide vorgestellten Ansätze dienen der Unterstützung der Extraktion von Zustandsnamen. Die unterschiedlichen Möglichkeiten zur Generierung der Trainingsdaten erfordern einen unterschiedlichen Zeitaufwand für die Nutzer:innen. Der Aufwand und die Qualität der Modelle werden dann mit der rein manuellen Extraktion verglichen.

Der automatisierte Ansatz erfordert den geringsten Zeitaufwand. Während des Trainingsprozesses müssen die Nutzer:innen die vorgeschlagene Liste des Bootstrapping-Algorithmus überarbeiten. Die Nutzer:innen beurteilen ob die einzelnen Einträge der Definition in Kapitel 4.2.1.1 entsprechen. Begriffe, die keine Zustandsnamen sind, werden aus der Liste gestrichen, während tatsächliche Zustandsnamen unverändert in der Liste verbleiben. Da an dieser Stelle nur der Zustandsname und nicht der Zustandswert erscheint, ist Domänenwissen erforderlich, um die Beurteilung vorzunehmen. Dieser Aufwand ist während des gesamten Trainingsprozesses nur einmal erforderlich. Nach dem Training des NER-Modells mit den überarbeiteten

Zuständen des Bootstrappers können die Zustände ohne zusätzlichen Zeitaufwand in neuen Daten erkannt werden.

Der halbautomatische Ansatz beginnt mit einer limitierten Auswahl von Anforderungen und extrahierten möglichen Zustandsnamen. Die Nutzer:innen müssen nun die Kandidatenliste analog zum automatisierten Ansatz überarbeiten. Einträge, die nicht der Definition entsprechen, werden gelöscht; korrekt identifizierte Zustandsnamen verbleiben unverändert in der Liste. Anschließend wird ein neuer Teil der Daten entsprechend den überarbeiteten Zustandsnamen markiert. Der NER wird neu trainiert und auf einen weiteren neuen Datensatz angewendet. Die Ausgabe ist wieder eine Liste von Zustandskandidaten. Der Zyklus beginnt erneut. Die Nutzer:innen sind hier in jedem Zyklus beteiligt, jedoch reduziert sich der Zeitaufwand durch zunehmende Optimierung des NER-Modells. Der qualitative Einfluss der investierten Zeit ist Bestandteil der Auswertung in Kapitel 4.2.4.

#### 4.2.3.5 *Limitationen*

Während des Trainings werden die Nutzer:innen möglichst wenig in den Trainingszyklus einbezogen. Dennoch haben sie starken Einfluss auf das Endergebnis. Der Fokus der Erkennung liegt auf Anforderungsdokumenten aus dem Automobilbereich und erfordert Domänenwissen, um eine hohe Qualität der Vorschläge zu gewährleisten. Die Einbeziehung der Nutzer:innen birgt die Gefahr, dass ein schlecht trainiertes Modell in der praktischen Anwendung nicht effizient arbeitet. Drei potenzielle Schwachstellen können die Ergebnisse beeinflussen:

- **Inkorrekte Einschätzung:** Die Beurteilung der vorgeschlagenen Kandidaten kann durch subjektive Sichtweisen auf das System beeinflusst werden. Trotz eindeutiger Definitionen ist eine inkorrekte Zuordnung durch die Nutzer:innen möglich.
- **Ungenügende Domänenkenntnisse:** Für die Beurteilung der vorgeschlagenen Zustände ist ein solides Verständnis des Systems notwendig, um eine korrekte Bewertung vornehmen zu können.
- **Systematischer Fehler:** Durch übliche Anwendungsfehler wie Unkonzentriertheit oder fehlerhafte Sorgfalt können korrekte Zustände gelöscht werden oder fehlerhafte Positionen in der Liste inkorrekt verbleiben.

Um den Einfluss von menschlichen Fehlinterpretationen zu minimieren, sollten mehrere Personen in den Trainingsprozess einbezogen werden.

Eine weitere Einschränkung der vorgestellten Ansätze sind die inhärenten Inkonsistenzen innerhalb der Anforderungsdokumente. Synonym verwendete Begriffe werden nicht zuverlässig als solche erkannt oder verbunden, sondern als individuelle Zustände betrachtet.

#### 4.2.4 *Ergebnisse*

In den folgenden Abschnitten werden die Ergebnisse erläutert. Zu Beginn sind in Abschnitt 4.2.4.1 die verwendeten Metriken erklärt. Daran anschließend ist der Versuchsablauf in Abschnitt 4.2.4.2 dargestellt, bevor in Abschnitt 4.2.4.3 genauer auf den verwendeten Datensatz eingegangen wird. Abschnitt 4.2.4.4 zeigt abschließend die Auswertungen.

##### 4.2.4.1 *Metriken*

Für die Qualität der Ansätze stehen Vollständigkeit und Genauigkeit der Ergebnisse im Vordergrund. Aus der Ergebnisliste resultieren die Annotationen in den Anforderungsdokumenten.
Die Vollständigkeit wird mithilfe des *Recall*-Werts ermittelt. Bezogen auf die Evaluation des Ansatzes errechnet sich der Recall mit folgender Formel:

$$Recall = \frac{Anzahl\ extrahierter\ tatsächlicher\ Zustandsnamen}{Anzahl\ tatsächlicher\ Zustandsnamen\ aus\ manueller\ Analyse}$$

Durch den Recall lässt sich eine Aussage über die Vollständigkeit der extrahierten Zustandsnamen in Hinblick auf die tatsächliche Anzahl vorhandener Zustandsnamen treffen. Der Wert liegt zwischen 0 für kein Zustandsname ist enthalten und 1 für alle möglichen Zustandsnamen sind enthalten. Der Recall ermöglicht jedoch keine Aussage über die Länge der Liste oder die Anzahl von irrelevanten Begriffen.

Um die Genauigkeit, beziehungsweise die Qualität der Liste zu beurteilen, wird die *Precision* berechnet. Das Maß setzt die Anzahl der extrahierten tatsächlichen Zustandsnamen ins Verhältnis zur Gesamtanzahl der extrahierten Begriffe. Die Precision errechnet sich durch die folgende Formel:

$$Precision = \frac{Anzahl\ extrahierter\ tatsächlicher\ Zustandsnamen}{Gesamtanzahl\ extrahierter\ Begriffe}$$

Die Werte liegen zwischen 0 für kein Zustand ist enthalten und 1 für alle Begriffe in der Liste sind Zustandsnamen.

Ein harmonisches Mittel der beiden Metriken wird durch die Berechnung des $F_1 - Scores$ ermittelt. Der $F_1 - Scores$ berechnet sich aus der Formel:

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

In der Regel sinkt die Precision bei steigendem Recall und umgekehrt.

#### 4.2.4.2 *Versuchsablauf*

Um die beiden Ansätze, wie sie in Kapitel 4.2.3 vorgestellt werden, zu evaluieren und zu bewerten, wird folgende Strategie verfolgt:

Die Extraktion der Ausgangszustandsnamen erfolgt an 50 zufällig ausgewählten Anforderungen. Diese manuelle Extraktion ergab elf Ausgangszustandsnamen. Dieser Umfang wurde gewählt, um die anfängliche Arbeit so gering wie möglich zu halten und dennoch genügend Daten für eine ordnungsgemäße Ausführung zu liefern.

Die vorverarbeiteten 14 000 Anforderungen im Datensatz werden nach dem Zufallsprinzip unabhängig von ihrer Quellspezifikation gemischt. Das geschieht unter der Annahme, dass die Herkunft der Anforderung keinen Einfluss auf die nachfolgende Verarbeitung hat. Das Modell behandelt die Anforderungen unabhängig vom Ursprung der Anforderungen.

Der Trainings-, Validierungs- und Testteil wird nach folgendem Schema durchgeführt: unter den genannten Annahmen können die gemischten Daten auf beliebige Weise aufgeteilt werden. 2000 der zufällig verteilten Anforderungen werden abgespalten, um als Testteil zur Bewertung des Modells zu dienen. Für den automatisierten Ansatz werden die verbleibenden 12 000 Anforderungen verwendet, um das NER-Modell zu trainieren. Während des Trainings dienen 10 % der Daten als Validierungsdaten. Für den halbautomatischen Ansatz werden die 12 000 Anforderungen nach dem Zufallsprinzip in 10 disjunkte Teilmengen aufgeteilt, um die Ausführung von Trainingsiterationen zu ermöglichen. Innerhalb jeder Trainingsiteration werden die nachlaufenden 10 % der Daten als Validierungsdaten verwendet.

Die Testdaten wurden zusätzlich manuell annotiert, um als Goldstandard zu dienen, gegen den alle Modelle ausgewertet werden. Daher müssen alle in den 2000 Anforderungen enthaltenen Zustandsnamen identifiziert werden. Hierzu analysierten vier Experten aus dem Automobilsektor die Daten, die disjunkt zwischen ihnen aufgeteilt wurden. Nachdem jeder Experte die relevanten Zustandsnamen aus der jeweiligen Teilmenge extrahiert hat, wurde die resultierende Liste kreuzvalidiert. Dazu überprüfte jeder Experte die Zustandsliste eines anderen. In dieser zweiten Runde konnte nur die Entfernung aus der Liste erfolgen; neue Zustandsnamen wurden nicht eingeführt. Daher enthält die resultierende Goldstandardliste nur Zustandsnamen, auf die sich zwei der Experten einigten.

Die Goldstandardliste mit den Zustandsnamen erleichtert die Kennzeichnung der Testdaten nach dem Ergebnis des NER-Modells, indem jedem Vorkommen eines Zustandsnamens die jeweils korrekte Kennzeichnung zugeordnet wird. Die manuelle Extraktion der Zustände

ergibt 222 eindeutige Zustandsnamen im Testsatz. Von den 2000 Anforderungen in der Testmenge enthalten 1228 Anforderungen tatsächlich einen Zustandsnamen.

#### 4.2.4.3 *Besonderheiten des Datensatzes*

Der Ansatz konzentriert sich auf die Extraktion von Systemzustandsnamen. Die verwendeten Daten sowohl für die Trainings- als auch für die Testsätze wurden von einem großen deutschen Automobilkonzern zur Verfügung gestellt. Das Unternehmen unterscheidet zwischen Komponenten- und Systemspezifikation. Die Komponentenspezifikation beschreibt einzelne kleine Fahrzeugkomponenten wie Sensoren oder Motoren, die jeweils eine bestimmte Funktionalität abdecken. In Systemspezifikationen werden mehrere Komponenten zu einer größeren, komponentenübergreifenden Fahrzeugfunktion zusammengefasst. Ein besonderes Merkmal der automobilen Anforderungen ist die Anhäufung von Signal- und Funktionsnamen. Eine typische Anforderung bei Signalnamen ist beispielsweise „Wenn die Komponente BSM das Signal BSM_Stat_Req empfängt, wechselt der Systemzustand auf ON“. Diese werden nicht konsistent verwendet und sind daher besonders schwierig automatisiert zu verarbeiten.

Der Ansatz verarbeitet beide Arten von Spezifikationen. Alle Anforderungen sind in englischer Sprache verfasst. Die Anforderungen wurden von Anforderungsexperten des Unternehmens manuell erstellt und Qualitätssicherungsmaßnahmen unterzogen. Die Anforderungen sind in Prosaform und ohne Einschränkungen, Muster oder Vorlagen geschrieben. Daher hängt es vom Verfasser ab, ob die britische oder amerikanische englische Schreibweise verwendet wird, weshalb beide Varianten in den untersuchten Daten vorkommen. Trotz des Überprüfungsprozesses durch die Experten sind in den Daten auch syntaktische und orthographische Fehler zu beobachten.

#### 4.2.4.4 *Auswertung*

Die Ergebnisse folgen dem in Kapitel 4.2.4.2 vorgestellten Versuchsablauf.

Abbildung 4.6 zeigt die Anzahl der erkannten Zustandsnamen für jede Untergruppe von Daten sowie das Verhältnis von abgelehnten und akzeptierten Zustandsnamen. In der ersten Iteration werden nur die elf Ausgangszustandsnamen angezeigt, da die erste Teilmenge der Daten allein nur als Trainingsdaten dient. Da das Modell bei jeder Iterationen mit zunehmender Anzahl von Trainingsanforderungen trainiert wird, verringert sich die Anzahl der zusätzlich vorhergesagten Zustandsnamen. Nach der dritten Iteration stagniert diese Zahl und schwankt um Werte knapp über 100. Dennoch nimmt die Anzahl der korrekten Zustandsnamen in Bezug auf den gesamten Satz von

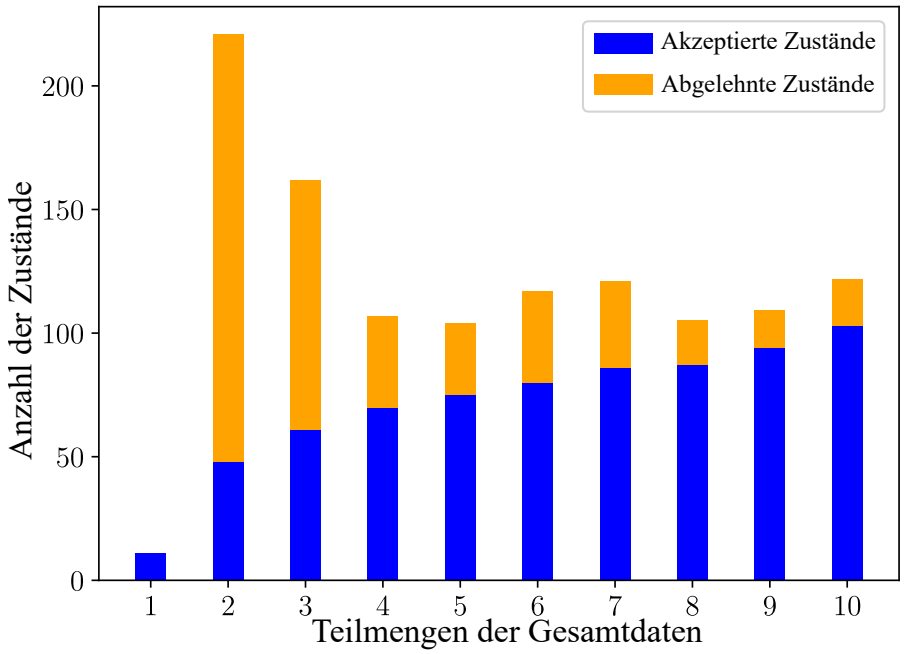



Abbildung 4.6: Erkannte Zustandsnamen je Iteration des halbautomatischen Ansatzes

Vorhersagen stetig zu. Mit jeder Iteration verbessert sich die Fähigkeit, neue valide Zustandsnamen innerhalb der Daten zu identifizieren.

Dies korreliert mit den in Abbildung 4.7 dargestellten Größen der Trainingsdaten. Bei jeder Iteration erhöht sich die Anzahl der theoretisch verfügbaren Trainingsdaten um 1200 Anforderungen, da eine neue Teilmenge einbezogen wird. Abbildung 4.7 zeigt ebenfalls den Anteil der Daten, die tatsächlich die akzeptierten Zustandsnamen der Iteration enthalten und die tatsächlich als Trainingsdaten dienen. Das Verhältnis der Trainingsdaten zu den verfügbaren Daten bleibt mit Ausnahme der ersten Iteration relativ konstant. Dort dienen nur die 11 Ausgangszustandsnamen als Eingabe, um relevante Anforderungen zu extrahieren. Daher werden nur etwa die 10 % der Daten für das Training verwendet, die diese Zustände tatsächlich enthalten. Bei den Iterationen zwei bis sieben machen die Trainingsdaten etwa 37 % der Daten aus. In den letzten drei Iterationen erhöht sich dieses Verhältnis auf circa 43 %.

Nach jeder Iteration des halbautomatischen Ansatzes werden das Modell auf die Testdaten angewendet und die Ergebnisse mit dem Goldstandard verglichen. Abbildung 4.8 zeigt die Precision-, Recall- und $F_1$-Metriken, die auf zwei verschiedene Arten berechnet werden. Abbildung 4.8a gibt die Metriken an, basierend auf der Anzahl korrekt vorhergesagten Labels in allen Anforderungen. Das heißt, je öfter ein Zustandsname in unterschiedlichen Anforderungen richtig erkannt wird, desto besser ist die entsprechende Metrik. Die dargestellten Werte sind die Durchschnittswerte aller Anforderungen im Testsatz. Dies berücksichtigt das mehrfache Vorkommen von Zuständen in den

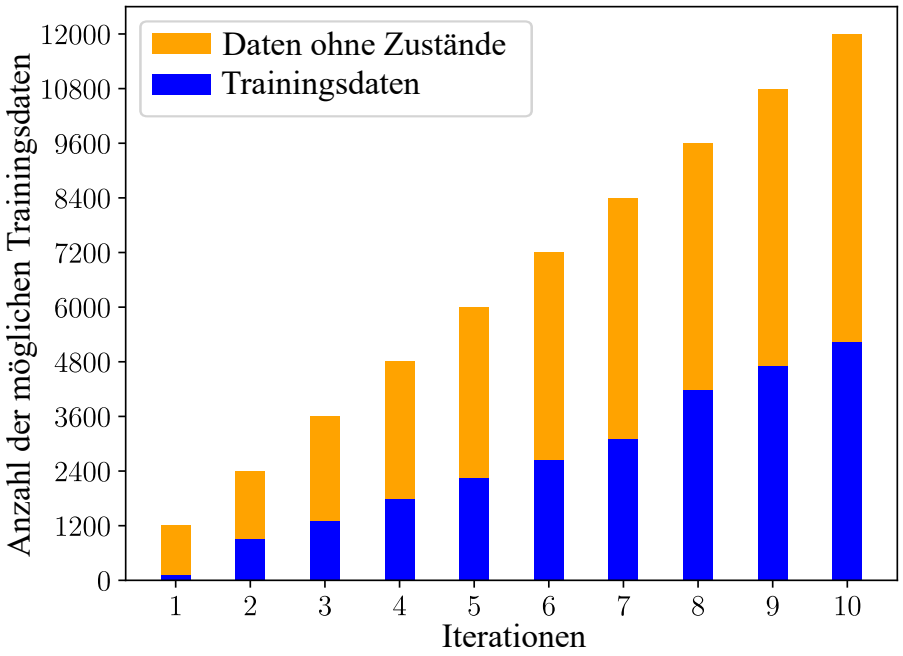


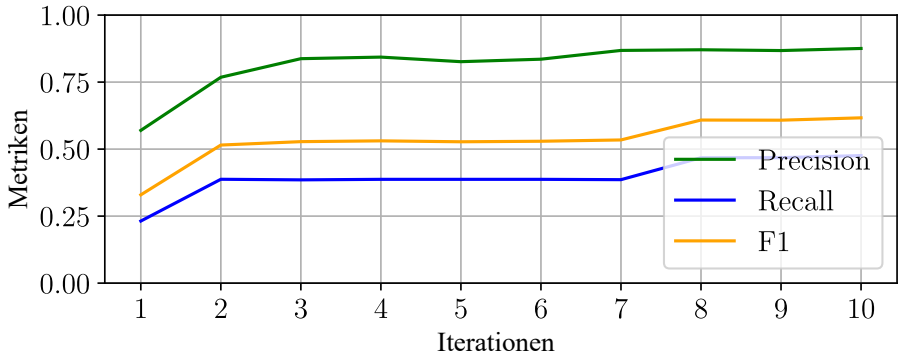

Abbildung 4.7: Trainingsdaten je Iteration

Daten, da das Modell jedes Vorkommen von Zustandsnamen zuverlässig identifiziert. Zusätzlich berücksichtigt diese Metrik auch partielle Übereinstimmungen von Zustandsnamen. Wenn ein Zustandsname aus mehreren Wörtern besteht, ist die Ausgabe des Modells auch dann relevant, wenn es nur einige der konstituierenden Wörter richtig kennzeichnet.

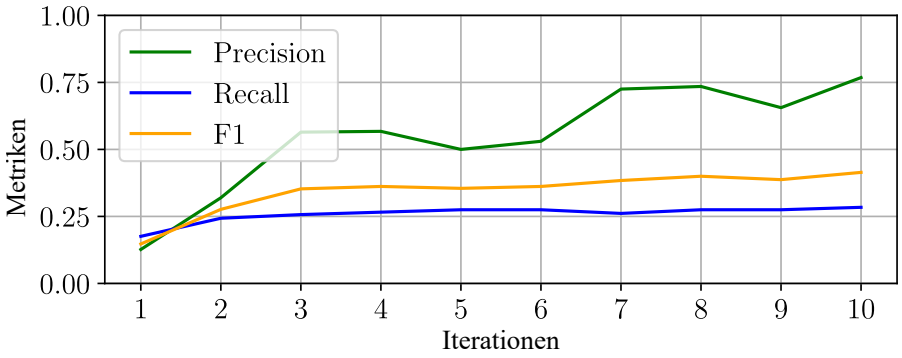
Die Grafik zeigt, dass das Modell zwei verschiedene Phasen der Verbesserung aufweist: eine zu Beginn und eine von Iterationen sieben zur Iteration acht. Die erreichte Precision von 0,88 zeigt die Zuverlässigkeit des Modells. Der Recall von 0,48 beschreibt, dass das Modell etwa die Hälfte der Zustandsbezeichnungen im Datensatz erkennt.

Die zweite Qualitätsbewertung wird in Abbildung 4.8b dargestellt. Hier beziehen sich die Metriken auf die tatsächlichen Zustandsnamen, die das Modell erkennt, verglichen mit den Zustandsnamen des Goldstandards. Dabei ist die Anzahl der vorkommenden Zustandsnamen unerheblich, lediglich ob ein Zustandsname erkannt wurde oder nicht geht in die Berechnung ein. Die Precision verbessert sich im Laufe der Iterationen fast kontinuierlich, während der Recall stagniert und sich nur um einen kleinen Betrag verbessert. Diese Auswertung zeigt das gleiche Muster wie die vorherige. Die Precision erreicht einen Endwert von 0,77. Daraus ergibt sich, dass drei Viertel der vorhergesagten Zustandsnamen korrekt sind. Der Recall erreicht einen Endwert von 0,28. Daher erkennt das Modell etwas mehr als ein Viertel der im Testsatz enthaltenen Zustandsnamen.

Der automatisierte Ansatz benutzt die gleiche Datenbasis wie der halbautomatische. Der Bootstrapper erhält die elf Ausgangszustände als Input um weitere Zustände in den 12 000 Trainingsanforderungen




(a) Metriken bezogen auf vorhandene Labels




(b) Metriken bezogen auf vorhandene Zustände

Abbildung 4.8: Qualität des halb-automatisierten Ansatzes auf den Testdaten

zu erkennen. Nach 20 Iterationen findet der Algorithmus insgesamt 200 Zustände. Nach der manuellen Überarbeitung werden 73 Zustände als gültig identifiziert und dienen als Input für das Training des NER-Modells. Beim Extrahieren der relevanten Anforderungen auf der Grundlage dieser Zustände verbleiben 4472 der 12 000 Anforderungen im Trainingssatz.

Precision und Recall sind im automatisierten Ansatz geringer als im halbautomatischen. Tabelle 4.3 gibt einen Überblick über die erreichten Werte der Metriken für beide Ansätze. Die Qualität weicht für alle Metriken um circa 0,1 von der halbautomatisierten ab. Insgesamt zeigen die Ergebnisse zur Extraktion von Zustandsnamen in Anforderungen vergleichbare Werte für den automatisierten und der halbautomatisierten Ansatz.

Der zweite Teil von Tabelle 4.3 beschreibt die Qualität der Erkennung von Zustandsnamen. Hinsichtlich der Precision unterliegt der automatisierte Ansatz dem halbautomatischen Ansatz und erkennt Zustandsnamen somit weniger zuverlässig. Der Recall hingegen ist beim automatisierten Ansatz höher; etwa ein Drittel aller Zustände in der Testmenge werden erkannt.

Tabelle 4.3: Vergleich der Evaluationsmetriken beider Ansätze

| | Metrik | Automatisierter Ansatz | Halbautomatisierter Ansatz |
|---|---|---|---|
| Labels | Precision | 0,77 | 0,88 |
| | Recall | 0,38 | 0,48 |
| | $F_1$ | 0,51 | 0,62 |
| States | Precision | 0,44 | 0,77 |
| | Recall | 0,30 | 0,28 |
| | $F_1$ | 0,36 | 0,41 |

### 4.2.5 *Validität*

Die Auswertung zeigt den Vergleich der beiden Ansätze zur Extraktion von Zuständen aus natürlichsprachlichen Anforderungen. Die Evaluationsergebnisse sind diversen kritischen Einflüssen unterworfen. Im Folgenden soll auf drei identifizierte Einflüsse auf die Gültigkeit der Evaluation und die unternommenen Bemühungen zur Minimierung dieser Einflüsse eingegangen werden.

Alle Ergebnisse beziehen sich auf die identifizierten Zustände des Goldstandards, was die Ergebnisse bereits beeinflussen kann. Da es keine öffentlichen Daten von Zuständen aus Anforderungsspezifikationen gibt, musste der Goldstandard manuell erstellt werden. Dieser Prozess ist fehleranfällig und wirkt sich direkt auf die Bewertungsergebnisse aus. Um die möglichen Fehleinschätzungen eines einzelnen Experten zu reduzieren, wurden vier Experten einbezogen. Der Goldstandard wurde in zwei Verarbeitungsschritten erstellt, einen für die Extraktion der Zustandsnamen und einen für die Überprüfung der identifizierten Kandidaten. Im Abschnitt 4.2.4.2 wird die genaue Verteilung der Daten ausführlich beschrieben.

Auch die gewählten Hyperparameter können die Ergebnisse beeinflussen. Eine Analyse des Parameterraums wurde nicht durchgeführt, da der Fokus der Untersuchung nicht auf der Parameteroptimierung lag. Sie basieren jedoch auf dem neuesten Stand der Forschung. Zukünftig könnte eine auf das Problem und die verfügbaren Daten abgestimmte Parameteroptimierung die Ergebnisse weiter verbessern.

Ein zusätzlicher externer Einfluss ist die Einbeziehung der Nutzer:innen. Sie verbessern zwar die Ergebnisse für den halbautomatischen Ansatz, können aber auch direkt negative Auswirkungen durch fehlerhafte Entscheidungen auf die Auswertung haben. Insbesondere die Ermittlung von Ausgangszuständen hat einen erheblichen Einfluss auf die Leistung beider Ansätze. Wie in Kapitel 4.2.4.2 beschrieben, wird die Interaktion der Nutzer:innen bewusst limitiert. Um einen möglichen negativen Einfluss weiter zu minimieren, wurden zwei

Experten in alle für die Evaluation erforderlichen Interaktionen einbezogen. Die Experten diskutierten die Entscheidungen unter Berücksichtigung des notwendigen Domänenwissens.

### 4.2.6 *Diskussion*

Beide vorgestellten Ansätze beziehen die Anwender:innen in den Trainingsprozess ein. Der gemeinsame Ausgangspunkt sind 11 Ausgangszustände. Diese wurden durch das Lesen von 50 zufällig ausgewählten Anforderungen extrahiert. In der anschließenden Auswertung dauert diese Extraktion 17 Minuten. In der praktischen Anwendung ist es denkbar, dass Nutzer:innen mit Domänenwissen dieses Set von Zustandsnamen erstellen, ohne eine große Anzahl von Anforderungen zu lesen. In diesem Fall ist der Aufwand vernachlässigbar gering.

Tabelle 4.4: Vergleich des Zeitaufwands

| **Manuelle Extraktion** | **Automatisierter Ansatz** | **Halbautomatisierter Ansatz** |
|---|---|---|
| 540 min | 15 min | 100 min |

Die Auswertung zeigt neben den Ergebnissen beider Ansätze auch den entsprechenden Zeitaufwand der manuellen Bearbeitung. Tabelle 4.4 weist die jeweilige Bearbeitungsdauer aus. Beim automatisierten Ansatz ist der Zeitaufwand am geringsten. Die Nutzer:innen müssen lediglich die vorgeschlagenen Kandidaten des Bootstrappers lesen und bewerten. Bei unserer Auswertung dauert es 15 Minuten, um die Liste zu prüfen und alle Begriffe auszusortieren, die nicht der Zustandsdefinition entsprechen. Alle weiteren Schritte sind vollständig automatisiert. Angewandt auf den Goldstandard wird eine Precision von 0,77 und ein Recall von 0,38 erreicht. Im Gegensatz dazu erreicht der halbautomatische Ansatz eine Precision von 0,88 und einen Recall von 0,48. Hier werden die Anwender:innen stärker in den Trainingsprozess einbezogen. In jeder Iteration müssen die Kandidaten überarbeitet werden. Im vorgestellten Szenario verbrachten die Anwender:innen insgesamt 100 Minuten für die Überprüfung von Zuständen in 10 Iterationen. An dieser Stelle ist es wichtig zu erwähnen, dass der Trainingsprozess des Modells nur einmalig stattfindet und das resultierende Modell wiederholt angewendet werden kann, um Zustände in neuen Daten zu extrahieren. Für das Lesen der 2000 Anforderungen und die Ermittlung der Zustände benötigte die Expertengruppe effektiv neun Stunden. Einerseits ist anzunehmen, dass durch dieses Vorgehen (nahezu) alle Zustände identifiziert wurden. Andererseits wird ein neuer Satz von 2000 Anforderungen weitere neun Stunden Arbeit erfordern, während das trainierte Modell

Zustände ohne zusätzlichen manuellen Aufwand in neuen Daten in kurzer Zeit extrahieren kann.

Beim Vergleich der beiden Ansätze in Bezug auf die entdeckten Zustände lassen sich signifikante Unterschiede feststellen. Der automatische Ansatz ist entsprechend dem Stand der Technik bei der Informationsextraktion eingerichtet [64, 80, 81]. Bei diesem Ansatz wird eine hohe Anzahl von Falsch-Positiven untersucht. Einige dieser Kandidaten werden als Zustände identifiziert, sind aber für eine Systemausführung des Systems nicht geeignet. Dazu gehören Namen, Verben, IDs anderer Systeme oder statische Eigenschaften des Systems. Hier sei als Beispiel das Automotive Safety Integrity Level (ASIL) ([50]) genannt, der definierte Zustandswerte hat und daher als ein Zustand charakterisiert werden könnte.

Der halbautomatische Ansatz produziert etwa 88 % weniger falschpositive Ergebnisse, von denen einige zwar sinnvoll charakterisiert sind, aber, wie bereits erwähnt, außerhalb des Anwendungsbereichs liegen. Diese verbesserte Precision ist vorhersehbar, da die Nutzer:innen die Vorhersagen des Modells kontinuierlich optimieren, um die Qualität der Trainingsdaten zu verbessern.

Der Ansatz kann Zustandsnamen vorannotieren und annnotiert idealerweise 88 %, wie die Evaluation zeigte. Ein zentraler Aspekt der Multilevel Markup Language ist die Möglichkeit manueller Annotation durch die Nutzer:innen, die dadurch entscheiden, welche vorannotierten Zustände in der Simulation beobachtet werden. Die manuelle Intervention ist jedoch optional; vorannotierte Spezifikationen könne auch ohne zusätzliche Interaktion verwendet werden.

# 5

# TEIL 2: ABBILDUNGSPROZESS

Dieses Kapitel umfasst die Beschreibung des Abbildungsprozesses und damit den zweiten Schritt des Gesamttestansatzes. Im Anschluss an den Annotationsprozess aus dem vorangegangenen Kapitel extrahieren die Nutzer:innen die Annotationen über das Tool und verknüpfen sie mit konkreten Signalen der Teststufen. Der Abbildungsprozess bildet damit die Verbindung zwischen natürlichsprachlichen Textbausteinen in Form von Annotationen und den Systemausführungen. In Kapitel 5.1 werden die verschiedenen Möglichkeiten innerhalb des Abbildungsprozesses beschrieben und deren Auswirkung auf den Gesamtprozess erläutert. Das Kapitel 5.2 umreißt kurz den weiteren Ablauf des Testprozesses nach dem Abbildungsprozess. Die Durchführung der Systemausführungen steht nicht im Fokus dieser Arbeit und wird daher nur der Vollständigkeit halber skizziert.

## 5.1 VERKNÜPFUNG VON ANNOTATIONEN UND SIGNALEN

Vor dem Start einer Systemausführung werden die Annotationen aus der Systemspezifikation extrahiert und mit konkreten Signalen und ihren zugehörigen Werten verknüpft. Dieser Vorgang ist ein manueller Vorgang, der von den Nutzer:innen durchgeführt wird. Die Signalnamen bezeichnen Signale, die das Systemverhalten abbilden. Die Verknüpfung von natürlichsprachlichen Ausdrücken mit Signalnamen des Systems wird als Abbildung bezeichnet. Alle gespeicherten Signalabbildungen werden in einer Tabelle, der Abbildungstabelle, festgehalten. Der Vorteil einer manuellen Abbildung liegt in der unabhängig nutzbaren Teststufe sowie in einem frei wählbarem Testwerkzeug. Es ist daher unerheblich, wie das System innerhalb der Teststufe repräsentiert wird. Beispielsweise können sich Signale in einer Simulation eines Anbieters von den Signalen einer HiL-Testsoftware eines anderen Anbieters unterscheiden. In diesem Fall erstellen die Nutzer:innen zwei verschiedene individuelle Signalabbildungen. Alle anderen Prozessschritte (Annotationsprozess, Auswertungsprozess, Darstellungsprozess) bleiben unbeeinflusst und in vollem Funktionsumfang ausführbar.

Der Abbildungsprozess ist ein bewusst manueller Prozess mit manuellem Mehraufwand in der ersten Systemausführung, der jedoch bei gleichbleibenden Signalen im weiteren Entwicklungsprozess entfällt. Andernfalls muss die Signalabbildung gegebenenfalls angepasst werden. Im Entwicklungsverlauf werden lediglich neu hinzukommende Signale neu abgebildet und der Abbildungstabelle hinzugefügt.

Eine erfolgreiche Abbildung ist abhängig vom Level der gewählten Annotation.

**Level 1 und Level 2:** In Level 1 ist das alleinige Vorkommen eines passenden Signals für eine erfolgreiche Abbildung ausreichend. In Level 2 ist ein adäquater Wertebereich in den verfügbaren Logdaten erforderlich. Beispielsweise können annotierte $States_{\{L1\}}$ nicht auf kontinuierliche Signale abgebildet werden. Gleiches gilt auch für $Values_{\{L1\}}$, $Events_{\{L1\}}$ und Time-Annotationen. Abbildung 5.1 zeigt exemplarisch die erfolgreiche Abbildung zweier Annotationen auf Signale und die korrekten Wertebereiche.

Tabelle 5.1: Abbildung von natürlichsprachlichen Level 1 und Level 2 Annotationen auf Signalnamen

| **Annotation** | | **Signalname** | **Wertebereich** |
|---|---|---|---|
| TA-System | → | <ta_stat> | zustandsbasiert |
| Geschwindigkeit | → | <fzg_geschw> | kontinuierlich |

**Level 3 und Level 4:** Für eine korrekte Abbildung von Level 3 Annotationen ist es erforderlich, dass Signalnamen und Signalwerte passend zugeordnet sind. Da Level 4 Annotationen ausschließlich aus Level 3 Annotationen bestehen, wird eine Level 4 Annotation nur dann erfolgreich abgebildet, wenn alle enthaltenen Level 3 Annotationen passend zugeordnet sind. Abbildung 5.2 zeigt exemplarisch korrekt zugeordnete Signalnamen und Signalwerte zu extrahierten Level 3 Annotationen. Zur Sicherstellung, dass Signalabbildungen korrekt sind, ist Domänenwissen der Nutzer:innen notwendig (Kapitel 3.3).

Tabelle 5.2: Abbildung von natürlichsprachlichen Level 3 Annotationen auf Signalnamen mit dazugehörigen Werten

| **Annotation** | | **Signalname mit Wert** |
|---|---|---|
| Distanz $\leq$ 3m | → | <fzg_abstand> $\leq$ 300 |
| TA = Warnstufe 1 | → | <ta_stat> = 1 |
| TA = Warnstufe 2 | → | <ta_stat> = 2 |

**Abbildung ist nicht möglich:** Eine Abbildung auf konkrete Signalnamen ist nicht immer möglich. Dies ist der Fall, wenn das Signal nicht vorhanden oder unbekannt ist. Tabelle 5.3 enthält eine unvollständige Abbildung für die Level 3 Annotation `Fahrzeug = erkannt`. Dies könnte im Testprozess ein Anzeichen dafür sein, dass die Testumgebung das Signal nicht bereitstellt oder ein inkorrekter Testfall gewählt wurde. Eine weitere Ursache könnte der Entwicklungsstand

der Software sein, in dem das Signal noch nicht verfügbar ist. Auch bei unvollständiger Abbildungen ist es stets möglich, die Testläufe zu starten und nur eine Teilmenge der Systemanforderungen zu verifizieren.

Tabelle 5.3: Unmögliche Signalabbildung einer Annotation

| **Annotation** | | **Signal** |
|---|---|---|
| Fahrzeug = erkannt | → | *N/A* |

**Mehrfachabbildung:** Darüber hinaus sind die Annotationen aus dem Text und die Signale aus dem Test nicht zwangsläufig 1:1 abbildbar. Daher können auch 1:n Abbildungen erstellt werden. Aus der allgemein formulierten Anforderung in Abbildung 5.1 sind die Annotationen, die in der Signalabbildung in Tabelle 5.4 berücksichtigt werden, extrahiert.

Die Ausdrücke `akustische Warnung` und `Blinksignal` in 5.4 demonstrieren eine 1:2 Abbildung. Eine Annotation wird jeweils auf zwei konkrete Systemsignale abgebildet. Die abgebildeten Signale werden standardmäßig mit einer ODER-Verknüpfung ausgewertet. Dies ermöglicht eine Signalabbildung auch dann, wenn in den natürlichsprachlichen Anforderungen allgemeine Formulierungen verwendet werden.

Ist das Blinksignal aktiviert, wird eine akustische Warnung ausgegeben.

Blinksignal = aktiviert — $State_{[System]}$-*Condition*

akustische Warnung = 1 — $Event_{[System]}$-*Condition*

Abbildung 5.1: Beispielanforderung mit Annotationen

Tabelle 5.4: Abbildung von Annotation auf mehrere Signalnamen mit dazugehörigen Signalwerten

| **Annotation** | | **Signal** |
|---|---|---|
| akustische Warnung = 1 | → | <ta_akustWarn_links> = 1 |
| | | <ta_akustWarn_rechts> = 1 |
| Blinksignal = aktiviert | → | <fzg_BlinkSig_links> = 1 |
| | | <fzg_BlinkSig_rechts> = 1 |

## 5.2 DURCHFÜHRUNG VON SYSTEMAUSFÜHRUNGEN

Nach dem Abbildungsprozess werden die Testdurchläufe durchgeführt. Die Durchführung ist unabhängig von den Annotationen der

Nutzer:innen und wird nicht von diesen beeinflusst. Die Softwaretests können auf beliebigen Teststufen erfolgen. Die Auswertung findet im darauffolgendem Auswertungsprozess statt.

### 5.2.1 *Teststufen in der Softwareentwicklung*

Der Testansatz umfasst einen Zyklus in dem die Systemausführung Teil des Gesamtansatzes ist. Dieser ist schematisch in Abbildung 3.1 in Kapitel 3.1.2 dargestellt ist. Die Ausführung der Software zu Testzwecken wird allgemein im Bereich der dynamischen Software-Testverfahren zusammengefasst [55]. Die tatsächliche Systemausführung steht jedoch nicht im Fokus dieser Arbeit, da diese frei gewählt werden kann und abhängig vom gewählten Testzweck ist. Der entwickelte Testansatz kann für beliebige Systemausführungen eingesetzt werden, unter der Voraussetzung, dass das Systemverhalten aufgezeichnet wird. Welche Anforderungen an die Systemaufzeichnung gestellt werden, die Entstehung der Systemaufzeichnung und ihre Eigenschaften sind im folgenden Kapitel beschrieben.

### 5.2.2 *Entstehung von Logdaten*

Logdaten aus Softwaretests beschreiben Signale, Zustandsänderungen und Aktionen des zu testenden Systems im Verlauf der Systemausführung. Sie werden automatisiert protokolliert und dienen dazu, das Verhalten der Software nachzuvollziehen. Zusätzlich können weitere Informationen über die Testumgebung, das Testumfeld oder den Testaufbau enthalten sein. Im Vordergrund steht dabei die Auswertung der Logdaten bezüglich der Software.

Die Logdaten liegen abhängig von der eingesetzten Testumgebung in unterschiedlichen Formaten und Darstellungen vor. Jedoch werden obligatorisch für alle Logdaten Zeitstempel, Eintragsnamen und Eintragswert erfasst.

**Zeitstempel:** Der Zeitstempel beschreibt den Zeitpunkt des Logeintrags. Dieser kann sich auf die aktuelle Uhrzeit beziehen oder auf eine systeminterne Zeit (beispielsweise auf die vergangene Zeit seit Teststart).

**Eintragsname:** Der Eintragsname beschreibt das zu protokollierende Ereignis. Das kann beispielsweise ein Systemzustand, ein Signalname oder eine Systemaktion sein.

**Eintragswert:** Der Eintragswert bezieht sich auf den Eintragsnamen und protokolliert den aktuellen Wert zur angegebenen Zeit des Zeitstempels.

Speziell im Automobilbereich entstehen Logdaten zu unterschiedlichen Zeitpunkten des Softwareentwicklungsprozesses. Die Auswertung von Logdaten erfolgt in jeder Teststufe (beispielsweise HiL oder SiL). Eine Möglichkeit dazu ist die Aufzeichnung der Signalwerte,

die zwischen den Softwarekomponenten ausgetauscht werden. Die Signale werden über eine Kommunikationsinfrastruktur im Fahrzeug ausgetauscht und für die Steuerung der Systeme genutzt. Diese Kommunikationsinfrastruktur wird abstrahiert in anderen Teststufen ebenfalls eingesetzt, um das System möglichst realitätsnah abzubilden. Daher ist es beispielsweise in Simulationen möglich, ebenfalls auf die gleichen Signale und Logdaten zuzugreifen. Abbildung 5.2 zeigt einen Auszug einer Logdatei, die während eines Simulationstestlaufs entstand.

```
2020-02-12 13:20:27,904 log: ta_stat: 0 (15.060,001s)
2020-02-12 13:20:27,909 log: ta_stat: 1 (15.080,001s)
2020-02-12 13:20:27,909 log: fzg_abstand: 10 (15.080,001s)
2020-02-12 13:20:27,909 log: fzg_geschw: 75 (15.080,001s)
2020-02-12 13:20:27,914 log: ta_AkustWarn: 0 (15.100,001s)
2020-02-12 13:20:27,914 log: ta_Lt_Ob_X: 0.0 (15.100,001s)
2020-02-12 13:20:27,914 log: ta_Rt_Ob_X: 0.0 (15.100,001s)
2020-02-12 13:20:27,914 log: fzg_geschw: 76 (15.100,001s)
```

Abbildung 5.2: Auszug einer Logdatei eines Simulationstestlaufs

Eine Zeile enthält einen Logeintrag mit den Werten: Datum, Zeitstempel (als Uhrzeit), Bezeichner, Eintragsname, Eintragswert und Simulationsdauer. Diese Logdaten werden systematisch analysiert und bezogen auf die Annotationen mithilfe der Abbildungstabelle aus dem Abbildungsprozess ausgewertet.

Der Testansatz umfasst ebenfalls die Möglichkeit die Logdaten aus mehreren Testdurchläufen gleichzeitig auszuwerten und die Ergebnisse als Gesamttest auf die Annotationen abzubilden. In Kapitel 6 ist der Auswertungsprozess anhand eines Experiments erläutert.

# 6

# TEIL 3: AUSWERTUNGSPROZESS

Dieses Kapitel umfasst den dritten Teil des Testprozesses und beschreibt die Auswertung des aufgezeichneten Systemverhaltens aus den Systemausführungen - die Logdaten. Hierfür müssen die Logdaten abhängig von der Teststufe interpretiert werden. Abschnitt 6.1 erläutert zwei verschieden Wege der Logdatenverarbeitung. Im Anschluss werden die interpretierten Logdaten in einer vierteiligen Analysemethode ausgewertet. Für die Auswertungen werden die markierten Textpassagen aus dem Annotationsprozess und die Signalabbildungen aus dem Abbildungsprozess mit einbezogen. Abschnitt 6.2 stellt eine Studie vor, in der die Analysemethoden im realen Kontext angewendet werden.

Dieses Kapitel basiert auf dem bereits veröffentlichten Beitrag von Pudlitz et al. [75].

## 6.1 INTERPRETATION VON LOGDATEN

Im Auswertungsprozess findet die systematische Analyse der Logdaten aus den Testdurchläufen statt. Die Logdaten entstehen in unterschiedlichen Teststufen des Entwicklungsprozesses und bilden das Verhalten des Systems in einem Testdurchlauf ab. Die Generierung der Logdaten wird in Kapitel 5.2.2 erläutert.

Die Auswertung der Logdaten basiert auf den Annotationen der Nutzer:innen aus dem Annotationsprozess aus Kapitel 4 und der Abbildungstabelle, die in Kapitel 5 beschrieben ist. Neben der detaillierten Annotationsauswertung lässt sie Aussagen über die Anwendbarkeit der Annotationen in der Anforderungsspezifikation, die Ähnlichkeit von Teststufen, die Passgenauigkeit von Teststufen und den Vergleich von Testfällen zu. Die vier zuletzt genannten Auswertungsmöglichkeiten werden in der Studie zur automatisierten Logdatenanalyse in Kapitel 6.2 anhand eines TA im Fahrzeug erläutert.

Die Auswertung der Annotationen kann auf zwei Arten erfolgen: Einerseits können Logdaten eines Systems aus einem Testdurchlauf ausgewertet und die Ergebnisse zu den Annotationen im Darstellungsprozess angezeigt werden. Andererseits kann eine Vielzahl von Systemausführungen zusammengefasst werden und anschließend annotationsbasiert als Gesamttestdurchlauf betrachtet werden. Diese Auswertungen unterscheiden sich von der Einzelauswertung dahingehend, dass als Resultat Durchschnittswerte angegeben werden und die Dauer die kumulierten Testdurchlaufszeiten angibt. Die Unterschie-

de der beiden Auswertungsmöglichkeiten werden in den folgenden Kapitel erläutert.

### 6.1.1 *Analyse von Annotationen bei einzelnen Systemausführungen*

In der Analyse einzelner Systemausführungen findet die Auswertung der Annotationen bezogen auf einen Testdurchlauf statt. Hierbei wird jede Annotation individuell betrachtet. Welche Auswertung je Annotation möglich ist, ist durch die Wahl des Annotationselements, beispielsweise $\text{State}_{\{L1\}}$ oder $\text{Value}_{\{L1\}}$, vorgegeben und in der Definition der Elemente in Kapitel 3.2.1 erläutert. Bei der Auswertung werden alle Testschritte und Logdateneinträge betrachtet. Neben den numerischen Ergebnissen und der grafischen Darstellung der $\text{Value}_{\{L1\}}$-Werte findet eine farbliche Klassifizierung in **Grün**, **Gelb** und **Rot** statt. $\text{Value}_{\{L1\}}$ Annotationen werden in der farblichen Einteilung nicht betrachtet.

**Grün:** Grün sind Annotationen in Level 1 bei einer vorhandenen Signalabbildung. An dieser Stelle wäre eine Systemausführung nicht notwendig um die Ergebnisse zu erhalten.

Für Level 2 Annotationen müssen Signalabbildungen und Auswertungen möglich sein. Das setzt voraus, dass eine passende Signalabbildung vorhanden ist und das abgebildete Signal in den Logdaten enthalten ist. Hierbei ist eine Signalabbildung auf einen falschen Wertebereich nicht zulässig.

Level 3 Annotationen sind in Grün dargestellt, wenn die Bedingungen in jedem Testschritt (Logeintrag), in dem der Zustandsname enthalten ist, erfüllt sind.

Für Level 4 Annotationen gilt die Annotation als erfüllt, wenn zum einen bei jedem erfüllten {L3}-Trigger auch zeitgleich die {L3}-Action erfüllt ist. Zum anderen wird bei jeder {L3}-Action überprüft, ob die dazugehörigen {L3}-Pre-Conditions ebenfalls erfüllt sind.

**Gelb:** In Gelb eingefärbt sind Annotationen im Allgemeinen, wenn keine Aussagen zum Ergebnis vorhanden sind und eine Annotation weder erfüllt noch als unerfüllt bewertet werden kann.

Level 1 Annotationen können nicht gelb dargestellt werden, da es nur die binäre Aussage erfüllt oder fehlend gibt - keine davon entspricht der gelben Kategorisierung.

Level 2 Annotationen werden bei fehlenden Signalabbildungen und daraus resultierenden fehlenden Ergebnissen gelb dargestellt.

Analog dazu sind Level 3 Annotationen gelb dargestellt, sollte es keine Auswertungen der Bedingungen in den Logdaten geben. Entweder kann aufgrund der Überprüfung der Logdaten keine Aussage zur Bedingung getroffen werden oder die Signalabbildung fehlt.

Level 4 Annotationen setzen sich aus Level 3 Annotationen zusammen. Dadurch werden Level 4 Annotation gelb eingefärbt, sobald eine der Level 3 Annotationen ergebnislos bleibt und gelb markiert wird. Zusätzlich wird Gelb verwendet, wenn der {L3}-Trigger oder

die {L3}-Action nicht eintritt und daher die gesamte Annotation zu keinem Zeitpunkt ausgewertet wird.

**Rot:** In Rot werden Annotationen auf Level 1 eingefärbt, wenn keine Signalabbildungen vorhanden sind.

In Level 2 werden Annotationen in Rot eingefärbt, falls eine Signalabbildung möglich ist, jedoch die Logdaten keine Eintragsnamen des abgebildeten Signals enthalten. Daraus folgt, dass keine Ergebnisse für das jeweilige Element vorhanden sind. Das kann durch einen unpassenden Testfall oder ein fehlendes Signal bedingt sein. Beispielsweise kann die Annotation *Regen* in der Systemausführung abgebildet werden; Regen kommt jedoch im Testfall oder als Signal in den Logdaten nicht vor.

Für Level 3 und Level 4 können konkrete Aussagen zur Erfüllung der Annotation getroffen werden, da diese Level konkrete Bedingungen zur Überprüfung enthalten. Rot sind demnach Level 3 und Level 4 Annotationen, in denen die spezifizierte Bedingung nicht oder nicht vollständig in jedem Simulationsschritt erfüllt wird.

Die Darstellung der Ergebnisse, der farblichen Annotationen und der Anforderungen wird in Kapitel 7, erläutert.

### 6.1.2 *Parallele Analyse von Annotationen von mehreren Systemausführungen*

In der Betrachtung und Analyse von Logdaten können sowohl einzelne als auch mehrere Systemausführungen ausgewertet werden. Für die Auswertung einer Summe an Systemausführungen gibt es zwei Entstehungsweisen von Logdaten: sequenziell (einzelne Logdaten nacheinander) und parallel (mehrere Logdaten zeitgleich aus einem Testszenario). Die anschließende Auswertung erfolgt stets parallel und ermöglicht komplexe und zugleich zeiteffiziente Analysen, die sich von etablierten Testverfahren abheben. Sie ist in Abbildung 6.1 schematisch dargestellt.

1. Im ersten Schritt findet eine getrennte Auswertung der Systemausführungen statt. Ausgehend von den Annotationen wird jede Logdatei einer Systemausführung individuell analog zu Kapitel 6.1.1 ausgewertet. Es entsteht für jede Logdatei eine Auswertung, die die Ergebnisse der Annotationen enthält. Bereits jetzt können Ergebnisse einzelner Systemausführungen optional in den Anforderungen angezeigt werden.

2. Im zweiten Schritt werden die Ergebnisse zusammengeführt und ausgewertet.

Die zusammengeführten Ergebnisse enthalten teils andere Auswertungen und werden im Folgenden genauer spezifiziert:

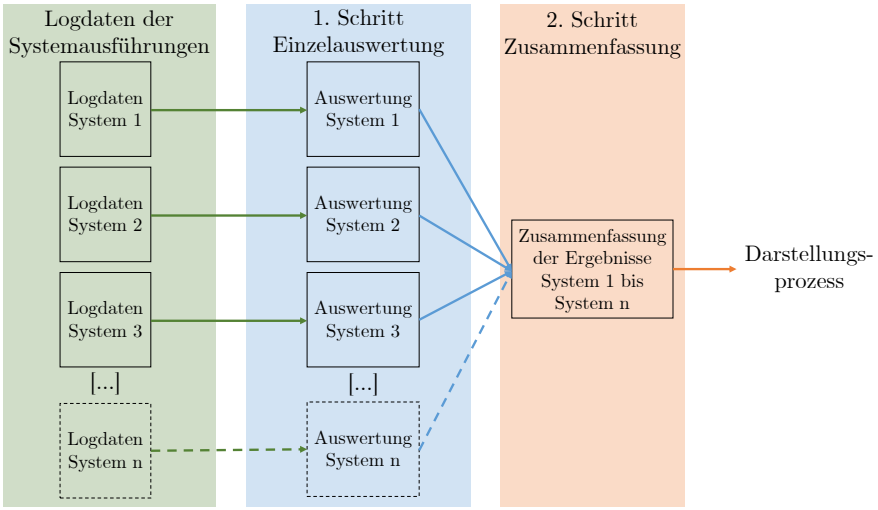



Abbildung 6.1: Auswertung von Logdaten aus parallelen Systemausführungen

Die Auswertung der **Value**$_{\{L1\}}$-Annotation zeigt keinen kontinuierlichen Verlauf, stattdessen werden arithmetisches Mittel, Minimum, Maximum und Standardabweichung über alle Testdurchläufe angezeigt.

Die **State**$_{\{L1\}}$-Annotation zeigt nach der Auswertung alle auftretende Zustandswerte und deren Verteilung über alle ausgewerteten Testläufe.

Die Auswertung der **Event**$_{\{L1\}}$-Annotation enthält alle vorkommenden Eventwerte und die Häufigkeit des annotierten Events bezogen auf alle Logdaten.

Die **Value**$_{\{L1\}}$**-Condition** zeigt die Dauer der gesamten erfüllten Bedingung in Sekunden. Zusätzlich wird das Ergebnis prozentual zur addierte Gesamtdauer aller Testdurchläufe angezeigt.

Die Auswertung der **State**$_{\{L1\}}$**-Condition** zeigt drei unterschiedliche Werte an: die Häufigkeit aus allen Testwerten; die Gesamtzeit der erfüllten Bedingung; die prozentuale Häufigkeit in Bezug zur Gesamtdauer.

Die **Event**$_{\{L1\}}$**-Condition** wird bezüglich der Anzahl der auftretenden Events und der durchschnittlichen Zeit zwischen zwei Eventwerten ausgewertet.

**Level 4 Annotationen** zeigen in der Ergebnisauswertung den prozentualen Anteil der erfüllten und nicht erfüllten Level 4 Bedingungen.

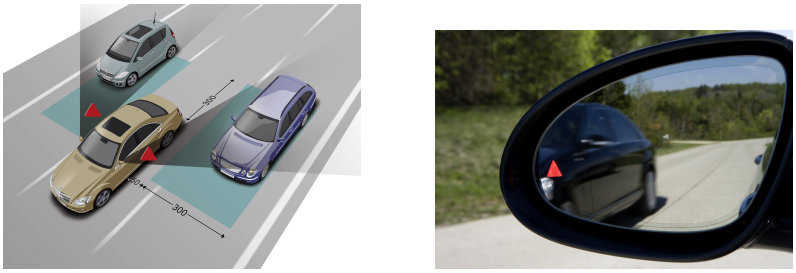
## 6.2 STUDIE ZUR AUTOMATISIERTEN LOGDATENANALYSE

Dieser Abschnitt beschreibt die experimentelle Auswertung, die das Analysepotenzial durch die vorgestellte Multilevel Markup Language unterstreicht. Der Fokus liegt hierbei auf dem Auswertungsprozess des Testansatzes, veranschaulicht an einem realen Beispiel aus dem Automobilbereich. Der Aufbau des Experiments orientiert sich an der

Richtlinie für Experimente in der Softwaretechnik von Jedlitschka und Pfahl [52]. Zu Beginn erfolgt die Beschreibung der Forschungsziele, der allgemeinen Versuchsplanung und des Kontexts, in dem die Studie durchgeführt wird. Anschließend erfolgt die Beschreibung der Datenerhebung und -analyse sowie der Ergebnisse des Experiments, welche zuletzt kritisch diskutiert werden.

### 6.2.1 *Kontext*

Diese experimentelle Auswertung basiert auf einer Anforderungsspezifikation aus dem Automobilbereich, die von einem Industriepartner zur Verfügung gestellt wurde.



(a) Überwachte Bereiche des TA-Systems

(b) Angezeigte Warnung im Außenspiegel

Abbildung 6.2: Funktionsweise des Totwinkel-Assistenten[1]

Das Anforderungsdokument definiert das Verhalten und den Aufbau eines Systems zur Überwachung des toten Winkels, das in modernen Personenkraftwagen eingesetzt wird. Abbildung 6.2 zeigt die Bereiche außerhalb des Fahrzeugs (6.2a), für die eine Warnung angezeigt wird (6.2b). Dieser TA weist eine angemessene Komplexität auf, um verallgemeinerbare Schlussfolgerungen ziehen zu können bei gleichzeitiger intuitiv verständlicher Funktionsweise.

Das System warnt den Fahrer visuell mit einer Anzeige im Seitenspiegel, wenn sich ein Objekt im toten Winkel des Fahrers befindet. Wenn dieses Szenario eintritt und der Fahrer diese Warnung nicht bemerkt und den Blinker setzt, um den Wunsch zum Spurwechsel zu signalisieren, beginnt die Anzeige zu blinken. Zusätzlich wird eine akustische Warnung ausgelöst, um den Fahrer davor zu warnen, die Spur zu wechseln und möglicherweise das Objekt im toten Winkel zu erfassen.

Intern hat der TA drei Zustände. Standardmäßig befindet sich das System in der Warnstufe 0. Wenn ein Objekt in den toten Winkel des Fahrzeugs eindringt, sich aber hinter dem Systemfahrzeug befindet, wird die Warnstufe 1 ausgelöst und die visuelle Warnung leuchtet auf.

Sobald sich das herannahende Fahrzeug neben dem Systemfahrzeug, aber immer noch im toten Winkel befindet und der Fahrtrich-

1 https://media.daimler.com/marsMediaSite/ko/de/9919190

tungsanzeiger betätigt wird, wird die Warnstufe 2 ausgelöst. Nur wenn das System auf Warnstufe 2 eingestellt ist, wird die oben beschriebene zusätzliche akustische Warnung ausgelöst.

Eine weitere Funktion des TA-Systems ist die Ausstiegswarnung. Sie ist nur aktiviert, wenn das Fahrzeug steht, beispielsweise in einer Parksituation. Diese Warnung soll das Erfassen von herannahenden Objekten wie Fahrrädern oder umstehende Gegenständen verhindern. Wird ein Gegenstand registriert, blockiert das System die entsprechende Tür.

Die Anforderungsspezifikation des TA-Systems besteht aus 443 Objekten, die in natürlicher Sprache verfasst sind. Da die Spezifikation in unstrukturierter, uneingeschränkter natürlicher Sprache geschrieben ist, weist sie eine Vielzahl unterschiedlicher Formatierungsstile auf. Die Gesamtzahl der Objekte setzt sich aus 292 Anforderungen, 60 Anmerkungen und 91 Überschriften zusammen.

### 6.2.2 *Zielsetzung*

Diese experimentelle Auswertung zielt darauf ab, die Anwendbarkeit der entwickelten Multilevel Markup Language an einem beispielhaften System aus dem Automobilbereich zu validieren und so den Auswertungsprozess zu beschreiben. Darüber hinaus werden zusätzliche Vorteile bei der Etablierung eines umfassenden Workflows mit mehreren Teststufen und umfangreichen Analysen aufgezeigt.

Die beiden in diesem Kapitel betrachteten Teststufen zeigen die Anwendungsbreite des vorgestellten Auswertungsprozesses für die Bewertung der Systemspezifikation in verschiedenen Entwicklungsstadien. Die durch die Verifikation der Anforderungen erzielten Ergebnisse werden interpretiert und kritisch diskutiert.

Zur Evaluation des entwickelten Auswertungsprozesses wurden vier Forschungsfragen definiert:

- **Forschungsfrage 1**: Wie sind Annotationen in realistischen Anforderungsspezifikationen verteilt und welcher Aufwand ist mit dem Annotationsprozess verbunden?
- **Forschungsfrage 2**: Können die Annotationen verwendet werden um die Ähnlichkeit verschiedener Testphasen in Bezug auf die Eigenschaften der Logdaten zu beurteilen?
- **Forschungsfrage 3**: Können die Annotationen verwendet werden um die Übereinstimmung der verschiedenen Teststufen mit den Merkmalen einer Anforderungsspezifikation zu beurteilen?
- **Forschungsfrage 4**: Können die Annotationen verwendet werden um die Übereinstimmung der tatsächlichen Testfälle mit den Merkmalen einer Anforderungsspezifikation zu beurteilen?

Die erste Frage adressiert die allgemeine Anwendbarkeit unseres Ansatzes durch die Anwendung der Multilevel Markup Language auf ein umfangreiches Anforderungsdokument. Die weiteren Forschungsfragen analysieren den zusätzlichen Nutzen, der sich aus der Anwendung der Multilevel Markup Language ergibt.

Forschungsfrage 3 und 4 erscheinen zunächst ähnlich; der Fokus bei Forschungfrage 3 liegt jedoch auf der Testphase, bei Forschungsfrage 4 auf dem tatsächlichen Testfall.

Der detaillierte Analyseprozess wird in Kapitel 6.2.3.2 erläutert.

### 6.2.3 *Methodik*

Die experimentelle Auswertung ist schematisch in Abbildung 6.3 dargestellt. Die gestrichelten Pfeile adressieren die Forschungsfragen. Ausgangspunkt und Basis für die Auswertung ist die Anforderungsspezifikation. Diese wurde mit der Multilevel Markup Language annotiert.

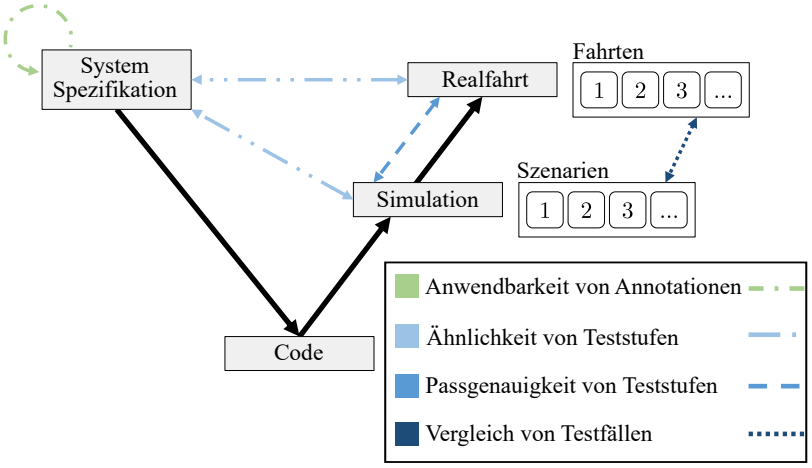



Abbildung 6.3: Schematischer Aufbau der Evaluation

Die Annotationen des Anforderungsdokuments werden mit einer Analyse auf Logdaten aus zwei Testphasen ausgewertet: erstens Logdaten aus einer umfangreichen Simulation und zweitens Logdaten aus realen Testfahrten. Als Simulator wurde Vehicle-2-X Simulation Runtime Infrastructure (VSimRTI)[2] als Simulator verwendet, da er ein umfangreiches Framework mit verschiedenen Simulatoren bietet [73].

Die realen Fahrdaten werden von einem Partner aus der Automobilindustrie zur Verfügung gestellt und entstanden aus kundenorientierten Testfahrten, die von Mitarbeitern des Unternehmens durchgeführt wurden.

2 https://www.dcaiti.tu-berlin.de/research/simulation/

#### 6.2.3.1 *Datenerhebung*

Dieser Abschnitt gibt einen Überblick über die verschiedenen Datenquellen und wie sie als Grundlage für die experimentelle Auswertung dienen.

##### ANFORDERUNGEN UND ANNOTATIONSDATEN

Um die geplanten Analysen durchzuführen, werden die Anforderungen manuell annotiert. Für die Annotation von Anforderungsdokumenten wird das selbst entwickelte Tool, welches in Kapitel 3.4 beschrieben wird, eingesetzt. Das Tool integriert die Objekte aus der Anforderungsspezifikation und stellt sie in strukturierter Form dar. Es werden Textpassagen manuell ausgewählt und durch die Wahl des Elements aus der Multilevel Markup Language eine Annotation erstellt. Die vorhandenen Annotationen zur aktuell ausgewählten Anforderung werden im Text hervorgehoben und zusätzlich in einem dafür vorgesehenen Abschnitt angezeigt.

Die Erstellung der Annotationen im Anforderungsdokument wird in zwei Schritten durchgeführt. Im ersten Schritt annotieren zwei Personen jeweils eine Hälfte des Anforderungsdokuments. Im zweiten Schritt validieren die zwei Personen die annotierte Hälfe des jeweils anderen. Unstimmigkeiten werden gemeinsam besprochen und gemeinsam entschieden. Dieses Vorgehen gewährleistet die Annotation und Validierung des gesamten Dokuments.

Im Annotationsvorgang wird das höchstmögliche Annotationslevel in jeder Anforderung angestrebt. Die detaillierte Analyse der Annotation ist Gegenstand von Kapitel 6.2.4.1.

##### ABBILDUNG VON ANNOTATIONEN

Zur Verknüpfung der annotierten Textpassagen mit Signalen eines Teststadiums wurden eine manuelle Abbildungstabelle für beide betrachteten Teststadien erstellt. Das Kapitel 5 beschreibt einen Abbildungsprozess detailliert.

Bei den realen Fahrdaten gewährleisten Konsultationen mit Domänenexperten eine qualitativ hochwertige und korrekte Abbildung der Annotationen. Die Abbildung der Textphrasen auf die Signale der Simulation wurde mit den TA-Entwicklern abgestimmt. Die Abbildungstabellen können aufgrund von Geheimhaltungsklauseln des Industriepartners nicht angegeben werden.

Die Grundlage für die Abbildungen dienen alle Level 3 Annotationen der Spezifikation. Level 1 und Level 2 Annotationen, die nicht implizit in den Level 3 Annotationen enthalten sind, werden somit im Abbildungsprozess und in nachfolgender Auswertung der Teststufen nicht berücksichtigt. Da die Annotationen auf Level 4 ausschließlich

aus Level 3 Annotationen aggregiert werden, sind diese inhärent in den Abbildungsprozess einbezogen.

Wie in Kapitel 3.2.1 beschrieben, besteht eine Annotation auf Level 3 aus einem Namen und einem zugewiesenen Wert. Somit besteht auch für die Abbildung die Voraussetzung, dass ein Signal und ein dazugehöriger Wert vorhanden sind.

#### LOGDATENGENERIERUNG

Die Logdaten werden aus zwei Quellen bezogen: aus einer Simulationsumgebung und aus einer Datenbank mit Testfahrtdaten eines Partners aus der Automobilindustrie.

**Simulationsdaten.** Der Simulator - VSimRTI - ist ein Framework, das verschiedene Simulatoren integriert. Der Verkehr wird mit dem Verkehrssimulator Simulated Urban Mobility (SUMO) ([62]) simuliert. Dieser ist bereits in das Framework VSimRTI integriert. In VSimRTI können benutzerdefinierte Funktionen implementiert werden um sie auf die Fahrzeuge im Szenario abzubilden. Alle Personenfahrzeuge im Szenario wurden mit einem eigens dazu implementierten TA-System ausgestattet.

Um das simulierte Szenario möglichst realistisch zu gestalten, wird das Luxembourg SUMO Traffic (LuST)-Szenario verwendet [19]. Dieses öffentlich zugängliche[3] Szenario modelliert den realen Verkehr in Luxemburg für die Dauer eines Tages. Die Infrastruktur von Luxemburg wird exakt in den Simulator übersetzt, mit Stadtautobahnen, Haupt- und Wohnstraßen. In die Simulation fließen offizielle demographische Daten in Bezug auf die Bevölkerungsverteilung und das Alter ein. Neben dem so modellierten Individualverkehr werden Busse des öffentlichen Nahverkehrs originalgetreu einbezogen. Obwohl die eigentliche Verifizierung des TA-Systems nicht im Fokus dieser Arbeit steht, ist die verwendete Simulation dicht an die Realität angelehnt.

Die Simulation beginnt um 8:00 Uhr morgens in der Simulationszeit, was der maximalen Anzahl simulierter Fahrzeuge entspricht. In unserer Simulation ergibt dies 1300 Fahrzeuge, die alle etwa 10 Minuten lang auf der virtuellen Karte fahren. Um die Daten aus dieser 10-minütigen Simulation zu sammeln, wurde der Simulator 5 Stunden auf einem Standard-Laptop ausgeführt. Wir sammelten Daten von 13 Signalen aus den 1300 Fahrzeugen mit einer Rate von 50 Hz. Die Logdaten aller Fahrzeuge enthalten durchschnittlich 218.057 Einträge.

**Testfahrtdaten.** Die realen Fahrdaten des Industriepartners entstammen tatsächlichen Testfahrten im Rahmen von kundenorientierten Fahrversuchen durch Mitarbeiter:innen und firmeneigene Fahrzeuge. Die Daten wurden in 10-Minuten-Intervalle eingeteilt. Anschließend wurden Intervalle identifiziert, in denen die relevanten Signale des TA-Systems aktiviert sind und Veränderungen im Verhalten zeigen. Das

3 https://github.com/lcodeca/LuSTScenario

Resultat sind 53 Fahrten von zehnminütiger Dauer. Die extrahierten Daten bestehen aus 30 Signalen aus den 53 Fahrten, die mit einer Rate von 50 Hz aufgenommen sind. In den Logdaten sind 30.006 Einträge gespeichert.

#### 6.2.3.2 *Ablauf der Datenanalyse*

Der in Kapitel 6.2.3 vorgestellte Aufbau der Evaluation definiert eine Analyse mit vier Schwerpunkten. Im Folgenden werden die Eigenschaften dieser Analyseansätze im Einzelnen beschrieben.

**Anwendbarkeit von Annotationen.** Die Evaluation der Annotationen erfolgt durch die Auswertung der Verteilung der Annotationen hinsichtlich ihrer Level und der Anforderungen, aus denen sie stammen. Hierbei wird untersucht, wie sich die Annotationen in jedem Level auf die verschiedenen Anforderungstypen in der Anforderungsspezifikation verteilen. Zusätzlich werden die Elemente der Annotationen auf jedem Level und ihr Vorkommen in untersuchten Anforderungsspezifikation untersucht. Abschließend erfolgt eine Diskussion über den Aufwand an Arbeitszeit, der für die Annotation der Anforderungen aufgewendet wurde.

**Ähnlichkeit von Teststufen.** Die Bewertung der Teststufenähnlichkeit erfolgt über den Vergleich der Fahrten in der Simulation mit den Fahrten der realen Fahrdaten auf der Basis systemabhängiger und unabhängiger Faktoren. Die *Fahrzeuggeschwindigkeit* und die *Fahrzeugbeschleunigung* sind systemunabhängige Faktoren und *Warnstufe* ein systemabhängiger Faktor für die Analyse.

Die Verteilungen dieser Signalwerte wird beobachtet und analysiert, um Erkenntnisse über die Simulationsqualität bezüglich der Realitätsnähe zu gewinnen. Daher wird diese quantitative Analyse verwendet, um eine qualitative Aussage über die Ähnlichkeit abzuleiten.

**Passgenauigkeit von Teststufen.** Die beiden Teststufen werden weiter hinsichtlich ihrer Übereinstimmung mit der Anforderungsspezifikation und den Annotationen verglichen. Dazu wird ein zweifacher Vergleich durchgeführt: hinsichtlich der Annotationen und hinsichtlich der Anforderungen, von denen diese Annotationen stammen.

> *Annotationsabdeckung:* Ausgangspunkt für diese Auswertung ist die Zuordnung von Annotationen zu Signalen. Die Metrik der Annotationsabdeckung definiert das Verhältnis der Annotationen, die eine Abbildung aufweisen und daher potenziell innerhalb der Tests untersucht werden können und allen Annotationen die erstellt wurden. Diese Analyse wird für jedes Level der Annotationen durchgeführt.
>
> *Anforderungsabdeckung:* Die Metrik der Anforderungsabdeckung stellt das Verhältnis der Anforderungen dar, die innerhalb der Tests ganz, teilweise oder nicht analysiert werden können bezogen auf alle Anforderungen mit enthaltenen Annotationen.

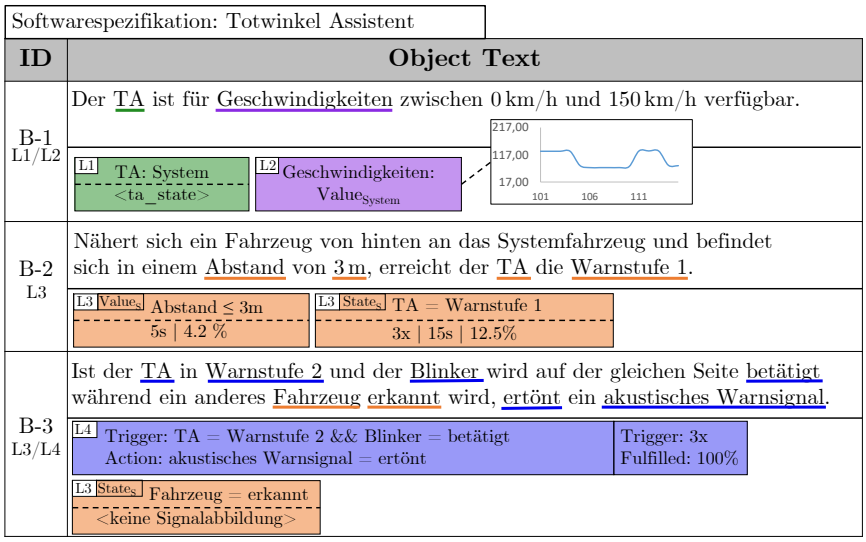



Abbildung 6.4: Darstellung der Evaluationsergebnisse auf exemplarischen Anforderungen

Das Beispiel in Abbildung 6.4 veranschaulicht die Metriken mit Beispielanforderungen. Von den sechs Annotationen werden fünf auf Signale abgebildet, die Annotation `Fahrzeug = erkannt` hat keine Abbildung. Daraus folgt eine Annotationsabdeckung von 83 %. Zwei der Anforderungen (B-1 & B-2) enthalten nur Annotationen mit einer Signalabbildung, während B-3 sowohl abgebildete als auch nicht abgebildete Annotationen enthält. Somit beträgt die Anforderungsabdeckung 67 % für vollständig zugeordnete Anforderungen, 33 % für teilweise zugeordnete Anforderungen und 0 % für vollständig nicht zugeordnete Anforderungen.

**Vergleich von Testfällen.** Die vierte Analyse beinhaltet die Auswertung der Testfälle.

Analog zur Evaluation der Passgenauigkeit von Teststufen konzentriert sich auch diese Evaluation auf die Metrik der Annotations- und Anforderungsabdeckung. Die beiden Metriken werden jedoch unterschiedlich interpretiert. Für den Vergleich von Testfällen werden nur die abgebildeten Annotationen betrachtet und hinsichtlich der Erfüllung in den Logdaten analysiert.

> *Annotationsabdeckung:* Die Annotationsabdeckung gibt das Verhältnis der Annotationen an, die innerhalb der Tests tatsächlich erfüllt werden zu allen abgebildeten Annotationen.
>
> *Anforderungsabdeckung:* Für die Anforderungsabdeckung werden die erfüllten Annotationen mit den Anforderungen in Beziehung gebracht, um einen Einblick zu erhalten, wie gut die Testdurchläufe die Spezifikationen tatsächlich abbilden.

Werden diese Metriken auf das Beispiel in Abbildung 6.4 angewendet, ist zu sehen, dass alle abgebildeten Annotationen ausgewertet

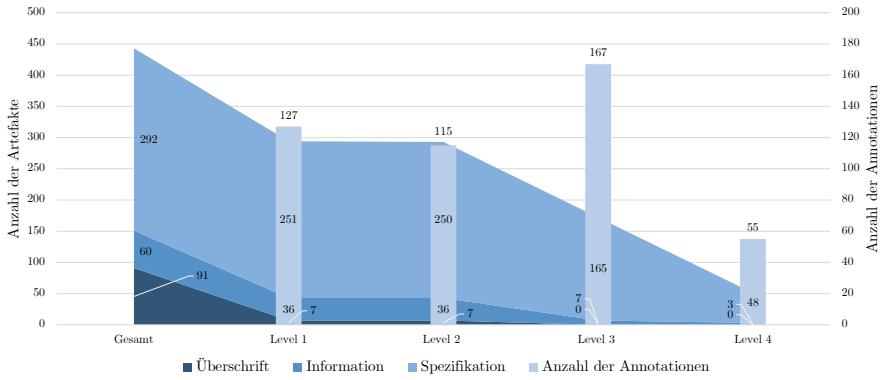



Abbildung 6.5: Statistische Auswertung von Annotationen in den Anforderungen

werden. Daher betragen die Annotationsabdeckung und die Anforderungsabdeckung 83 % bzw. 67 %. Das Potenzial der Auswertung wird also ausgeschöpft, da 100 % der abgebildeten Annotationen erfüllt sind.

### 6.2.4 *Ergebnisse*

Dieser Abschnitt enthält die Analyse des Experiments gemäß den in den vorhergehenden Kapiteln vorgestellten Strategien. Die Struktur ist so aufgebaut, dass sie die in Kapitel 6.2.3 vorgestellten Forschungsfragen beantwortet.

#### 6.2.4.1 *Evaluation der Anwendbarkeit von Annotationen*

Der folgende Abschnitt stellt die Verteilung der Annotation in der Anforderungsspezifikation dar. Aus Gründen der Lesbarkeit wird in diesem Abschnitt der übergeordnete Begriff für die drei Arten von Texttypen (Anforderung, Information, Überschrift) in der Spezifikation, wie in Kapitel 2.1 erklärt wird, *Artefakte* sein. Die Artefakte werden, wie bereits beschrieben, manuell annotiert.

Die Abbildung 6.5 zeigt die Anzahl und Art der Artefakte, die Annotationen für jedes Level enthalten, sowie die Anzahl der eindeutigen Annotationen für jedes Level. Letzteres wird durch die Säulen dargestellt, die sich auf die zweite Achse beziehen.

Level 1 enthält 127 Annotationen mit 105 Annotationen die als *System* annotiert sind und 22 als *Environment*. Da die Level 1 Annotationen einheitlich in den Anforderungen verwendet werden, verteilen sie sich auf 294 Artefakte mit durchschnittlich 2,3 Level 1 Annotationen pro annotiertem Artefakt. Die Aufschlüsselung der Artefakte zeigt, dass 85,4 % Anforderungen, 12,2 % Informationen und etwa 2,4 % Überschriften sind. Daher enthalten etwa 86 % aller Anforderungen eine Annotation auf Level 1, 60 % aller Informationen und etwa 7.7 % aller Überschriften.

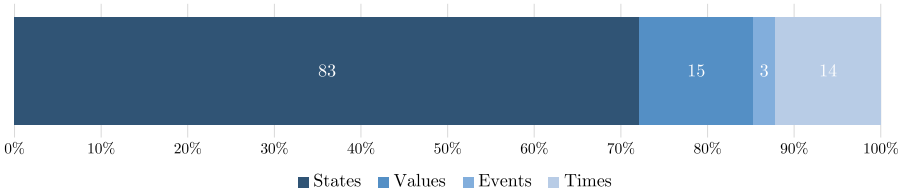



Abbildung 6.6: Verteilung der Level 2 Annotationen auf die einzelnen Elemente

Ausgehend von diesen Annotationen werden 115 auf Level 2 verfeinert, was zu 293 annotierten Artefakten führt. Die Mehrzahl der Annotationen auf Level 2 sind *States*, was in der Verteilung der Level 2 Annotationen in Abbildung 6.6 dargestellt wird.

Es gibt lediglich eine Anforderung, die nur Level 1 Annotationen enthält. Daher ist die Anzahl der Annotationen aus Level 2 pro Artefakt ähnlich der für Level 1 und beträgt 2,2.

Im Experiment werden 73 der Annotationen von Level 2 auf Level 3 übertragen, was 63,5 % aller Level 2 Annotationen ausmacht. Durch die Möglichkeit, mehrere Zustandswerte einem Zustandsnamen pro Level 2 Annotation zu erstellen, ergeben sich daraus 167 Annotationen. Daher werden jeder Annotation auf Level 2, die auf Level 3 verwendet wird, durchschnittlich 2,3 Bedingungen zugewiesen. Die Gesamtzahl der Annotationen nimmt zu, verteilt sich jedoch auf deutlich weniger Artefakte mit nur 172, die Annotationen auf Level 3 enthalten. Etwa 4,1 % dieser Artefakte sind Informationen, der restliche Anteil sind Anforderungen. Im Durchschnitt gibt es 2,1 Level 3 Annotationen pro annotiertem Artefakt.

Von allen Level 3 Annotationen werden 61,1 % verwendet, um kausale Beziehungen auf Level 4 herzustellen, was zu 55 unterschiedlichen Annotationen führt. Daher kommt jede Annotation aus Level 3 in etwa 1,5 Annotationen des Level 4 vor. Im Durchschnitt enthält jede Annotation aus Level 4 etwa 1,6 {L3}-Pre-Condition oder {L3}-Trigger und etwa 1,2 {L3}-Action. Mit 63,6 % aller Annotationen des Level 4 ist die Mehrzahl der kausalen Beziehungen als Trigger annotiert. Darüber hinaus sind 85,3 % aller Level 4-Annotationen mit mehreren verknüpften Bedingungen oder Actions mit dem logischen Operator *AND* verbunden, während die verbleibende Minderheit über ein *OR* verbunden ist. Es enthalten 51 Artefakte Level 4 Annotationen, wobei nur 3 davon Informationen und die verbleibenden Anforderungen sind.

Die sehr wenigen vorkommenden Annotationen innerhalb der Überschriften sind auf generische Annotationen oder spezifische Systemnamen zurückzuführen, die in einem kleinen Teil der Überschriften vorkommen. Das Verhältnis von Informationen und Überschriften ist in Bezug auf die Gesamtverteilung und die Verteilung der Annotationen ähnlich. Da die Informationen als zusätzlicher Inhalt zur

weiteren Spezifizierung der tatsächlichen Anforderungen dienen, enthalten sie einige Annotationen. Wie in der Analyse beobachtet wurde, insbesondere in Level 3 und Level 4, wo der Informationsgehalt der Annotationen erheblich zunimmt, stammt die überwiegende Mehrheit der Annotationen aus Anforderungsartefakten.

Der Annotationsprozess ist ein mehrschrittiger Prozess. Die Zeit wird für den Annotationsprozess und den Überprüfungsprozess getrennt gemessen. Die gesamte Anforderungsspezifikation wird von zwei Domänenexperten in 5 Stunden und 47 Minuten annotiert. Die Dauer des gesamten Annotations-Review-Prozesses beträgt 83 Minuten.

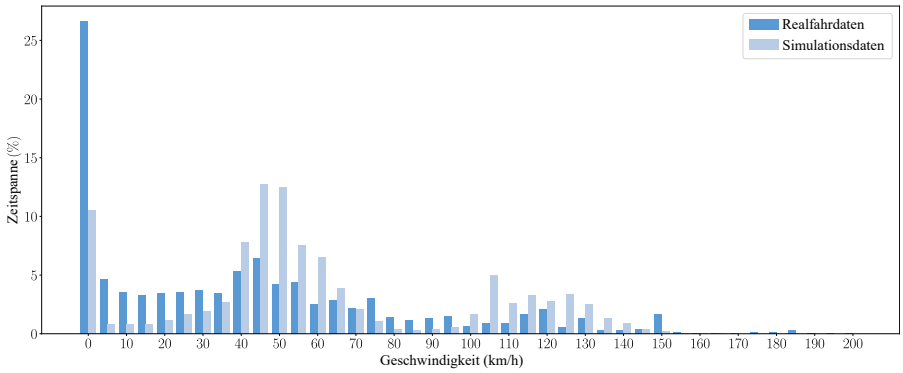
**Interpretation der Ergebnisse:** Die Analyse der Anwendbarkeit der Multilevel Markup Language auf eine umfassende Anforderungsspezifikation untersucht und beantwortet die erste Forschungsfrage. Die vorgestellten Ergebnisse bestätigen die Annahme, dass die Sprache gut skaliert. Es wird gezeigt, dass es möglich ist, einen wesentlichen Teil der funktionalen Implikation der Spezifikation mithilfe der strukturierten Annotationen darzustellen. Darüber hinaus werden alle möglichen Arten von Annotationen verwendet, die die Relevanz aller Bestandteile der Multilevel Markup Language darstellen. Darüber hinaus kann die Verteilung der Annotationen und der zugehörigen Levels als relatives Maß, beziehungsweise als Vergleichsparameter für die Komplexität zwischen zwei Spezifikationen interpretiert werden.

#### 6.2.4.2 *Evaluation der Ähnlichkeit von Teststufen*

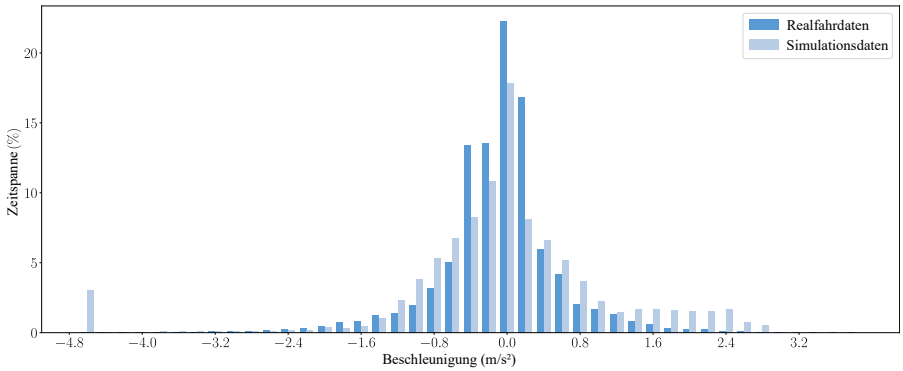
In diesem Abschnitt wird analysiert, wie ähnlich zwei Testphasen für diese spezielle Funktion sind. Dazu dienen die Logdaten der jeweiligen Teststufe. Insbesondere wird in der hier vorgestellten Auswertung analysiert, wie gut die Simulationsszenarien die realen Fahrdaten repräsentieren. Die Analyse erfolgt hinsichtlich systemunabhängiger Faktoren, der Geschwindigkeit und der Beschleunigung der Fahrzeuge und systemabhängiger Faktoren, der Warnstufe des TA-Systems.

Die Faktoren, mit denen die Teststufen verglichen werden, sind im allgemeinen Testprozess nicht festgelegt und werden mithilfe von Domänenwissen und Systemkenntnissen definiert. Die Auswahl der möglichen Faktoren entsteht durch die extrahierten Annotationen des Markierungsprozesses in der Anforderungsspezifikation.

Die Warnstufe wird als systemabhängige Faktor gewählt, da sich dieser Systemzustand innerhalb des Annotationsprozesses als sehr relevant herausgestellt hat. Die Auswahl der Vergleichsfaktoren kann daher von den Nutzer:innen auf Basis von Vorkenntnissen getroffen werden und wird durch die Annotation der Systemspezifikationen erleichtert.




(a)




(b)

Abbildung 6.7: Histogramm der (a) Geschwindigkeit und der (b) Beschleunigung innerhalb der Simulation und der Realfahrten.

**Geschwindigkeitsvergleich.** Abbildung 6.7 zeigt die Histogramme der Geschwindigkeiten und Beschleunigungen für die realen Daten und der Simulation als prozentuale Zeitanteile für diskretisierte Werte.

In Abbildung 6.7a sind die Geschwindigkeiten in Intervalle von 5 km/h diskretisiert. Die realen Fahrzeuge verbringen mehr Zeit im Stillstand, da mehr als 25 % der Zeit die Fahrzeuggeschwindigkeit zwischen 0 und 5 km/h lag. Der entsprechende Anteil für die simulierten Fahrzeuge beträgt nur etwa 10 %.

Es ist zu beobachten, dass die Fahrzeuge bei den Testfahrten im Vergleich zur Simulation in der Regel mit geringeren Geschwindigkeiten fahren. Die Simulation zeigt annähernd normalverteilte Geschwindigkeiten um die 50 km/h und 130 km/h Marke, mit einer Ausnahme bei 110 km/h. Dies lässt auf ein deterministisches Fahrerverhalten in der Simulation schließen, da dies die zulässigen Geschwindigkeiten innerhalb der Stadt, auf der Autobahn und auf der Autobahn bei Regen sind.

Um die Verteilungen der Geschwindigkeiten weitergehend zu interpretieren, zeigt Abbildung 6.8a einen Box-Whisker-Plot der Daten. Der dargestellte Kasten zeigt das erste und dritte Quartil, wobei die enthaltene Linie den Median darstellt. Die vertikalen Linien zeigen

die 5 und 95 Perzentile. Das beschriebene Verhalten wird durch diese Darstellung zusätzlich unterstützt, da die Quartile der Simulationsdaten höher verschoben sind als die entsprechenden Quartile der realen Daten.

Die 95 %-Markierungen sind ähnlich mit einem Unterschied von unter 3 km/h. Die gezeigten Mediane liegen bei 36,9 km/h und 56,6 km/h für reale und Simulationsdaten. Dies ist eine signifikante Abweichung. Die Mittelwerte und Standardabweichungen der Geschwindigkeiten sind für reale und Simulationsdaten 43,0 km/h $\pm$ 42,03 km/h beziehungsweise 62,33 km/h $\pm$ 38,35 km/h. Die Standardabweichungen zeigen weiterhin, wie vielfältig und breit gestreut die Daten sind.

Der Vergleich der Geschwindigkeiten zeigt einen deutlichen Unterschied zwischen den realen Daten und den Simulationsdaten. Jeder Datensatz zeigt spezifische Besonderheiten auf, die sich durch die jeweiligen Eigenschaften der Logdaten (beispielsweise deterministisches Verhalten in der Simulation) erklären.

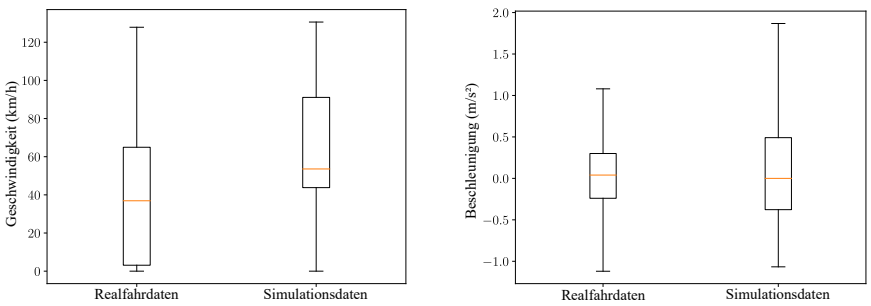
**Beschleunigungsvergleich.** Abbildung 6.7b zeigt das Histogramm der Fahrzeugbeschleunigungen für die realen Daten und der Simulation. Die Werte sind in Intervalle von 0,2 $m/s^2$ diskretisiert. Hier zeigen die simulierten Fahrzeuge ein nahezu normalverteiltes Verhalten um Null. Eine Ausnahme bilden die konstanten Prozentsätze für die Werte zwischen 1,4 $m/s^2$ und 2,4 $m/s^2$, die auch aufgrund des im Simulator definierten deterministischen Verhaltens existieren. Zusätzlich gibt es einen Anstieg der Vorkommen bei -4,6 $m/s^2$. Dies ist mit hoher Wahrscheinlichkeit der definierte Wert für abruptes Abbremsen.

Die Beschleunigungswerte für die realen Daten weisen eine ähnliche Verteilung mit einer annähernd normalen Verteilung auf. Die Unterschiede zeigen sich deutlich im Box-Whisker-Plot in Abbildung 6.8b. Die Quartile der Simulationsdaten decken einen breiteren Wertebereich ab, während die Mediane mit 0,04 $m/s^2$ und 0,0 $m/s^2$ für die Real- und Simulationsdaten ähnlich sind. Darüber hinaus ist die 95 % Marke der Simulationsdaten höher als bei den realen Daten, was mit den Beobachtungen aus dem Histogramm übereinstimmt.

Die Werte für Mittelwert und Standardabweichung unterstützen diese Beobachtung mit 0,021 $\pm$ 0,67 $m/s^2$ und 0,035 $\pm$ 1,24 $m/s^2$ für reale beziehungsweise Simulationsdaten. Die Standardabweichung der Simulationsdaten ist deutlich größer, was sich bei Betrachtung der Quartile in der Box-Whisker-Grafik reproduzieren lässt.

**Warnstufenvergleich.** Die bisherige Analyse erfolgt anhand systemunabhängiger Faktoren. Zusätzlich können die Szenarien anhand systemabhängiger Faktoren verglichen werden. In diesem Fall erfolgt der exemplarische Vergleich anhand der Warnstufe des TA-Systems.

Eine allgemeine Analyse für jedes Signal eines Systems, das verglichen werden kann, ist die Zeitspanne, in der das Signal einen bestimmten Wert erreicht hat. Für die Warnstufe gibt es drei mögliche Werte, 0 für keine Warnung, 1 für eine erhöhte Warnstufe und 2 für




(a) (b)

Abbildung 6.8: Box-Whisker-Plot der (a) Geschwindigkeit und der (b) Beschleunigung innerhalb der Simulation und der Realfahrdaten.

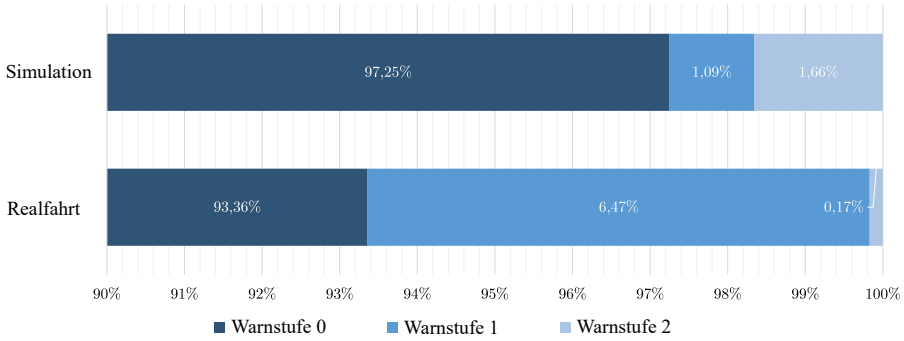



Abbildung 6.9: Vergleich der unterschiedlichen Warnstufen bezogen auf ihre Aktivierungszeit.

die höchste Warnstufe - wie in Kapitel 6.2.1 erläutert. Der Anteil der Gesamtzeit, für den diese Signalwerte angenommen werden, wird für reale und Simulationsdaten in Abbildung 6.9 visualisiert. Es ist zu beachten, dass die Skala der Achse bei 90 % beginnt, um die kleinen Prozentsätze für die Warnstufen 1 und 2 hervorzuheben.

Sowohl für die Simulation als auch für die realen Fahrdaten befindet sich das System in der überwiegenden Mehrheit der Zeit in Warnstufe 0, was eine unkritische Situation ohne Objekt im toten Winkel des Fahrzeugs bedeutet. Die Warnstufen 1 und 2 sind zwischen den beiden Datenquellen unterschiedlich verteilt. In den realen Fahrdaten wird die Warnstufe 1 für unter 6,5 % erreicht, während diese Stufe in den Simulationsdaten in 1,09 % der Zeit vorhanden ist. Für Warnstufe 2 sind die Verhältnisse umgekehrt. In der Simulation ist diese Warnstufe in 1,66 % der Zeit vorhanden, während in den realen Fahrdaten nur 0,17 % der Zeit Warnstufe 2 erreicht wird.

Zusammenfassend zeigt sich, dass die Warnstufe 1 in der Simulation nicht gleichermaßen abgedeckt ist wie in den realen Fahrdaten. Bei Warnstufe 2 zeigt die Simulation mehr Aktivierung im Vergleich zu den realen Daten. Diese Tatsache muss bei der Analyse der Annotatio-

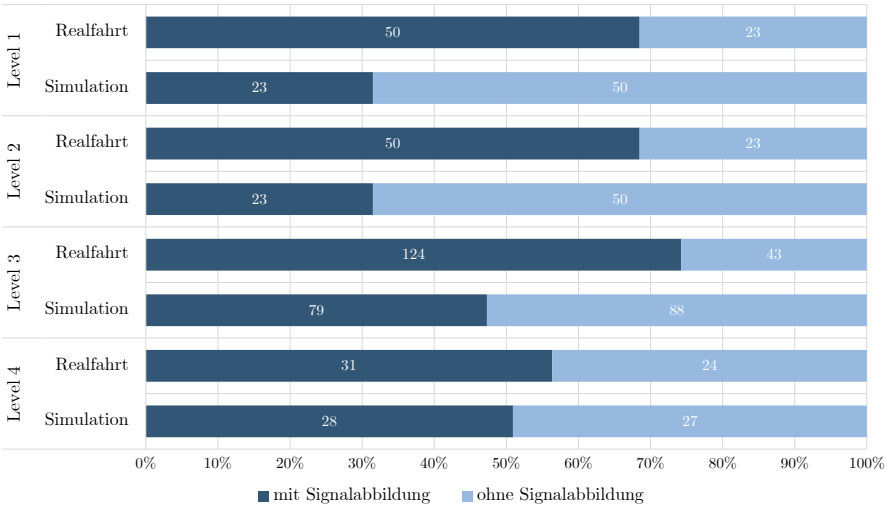



Abbildung 6.10: Verhältnisse der abgebildeten Annotationen je Level.

nen und beim Vergleich der beiden Teststufen berücksichtigt werden.

**Interpretation der Ergebnisse:** Die vorgestellten Ergebnisse bilden die Basis zur Beantwortung der zweiten Forschungsfrage, die das Potenzial des Ansatzes zur Beurteilung der Ähnlichkeit der verschiedenen Teststufen angibt. Die Ergebnisse zeigen, dass es wenige Ähnlichkeiten zwischen der Verteilung der analysierten Signale der realen Daten und der Simulationsdaten gibt. Daraus lässt sich kein Nachteil einer der Teststufen ableiten. Ziel der Untersuchung ist ein Vergleich der ausgewählten Teststufen, das Ergebnis muss im Kontext des Systems und der Ziele der Nutzer:innen bewertet werden.

#### 6.2.4.3 *Evaluation der Passgenauigkeit von Teststufen*

Um die beiden Teststufen hinsichtlich der Einhaltung der Spezifikation zu vergleichen, werden die Annotationen als Vergleichsmittel verwendet. Daher werden die Metriken der Annotation und der Anforderungsabdeckung, wie in Kapitel 6.2.3.2 definiert, angewendet. Dies ist Grundlage um aus dem Abbildungsprozess Aussagen über das Potenzial der verschiedenen Teststufen abzuleiten.

Die Verhältnisse von Annotationen, die auf Signale innerhalb der Real- und Simulationsdaten abgebildet werden, sind in Abbildung 6.10 bezogen auf ihr Level dargestellt. Wie in Abschnitt 6.2.3.1 beschrieben, erfolgt der Abbildungsprozess auf der Grundlage der Annotationen des Level 3. Innerhalb der realen Fahrdaten werden fast 75 % der Annotationen des Level 3 auf Signale mit entsprechenden Werten abgebildet. Für die Simulation konnten etwa 47 % der Annotationen abgebildet werden.

Die in Abbildung 6.10 dargestellten Verhältnisse von Level 1 und 2 sind die direkten Ergebnisse der Abbildung des Level 3, da die Anno-

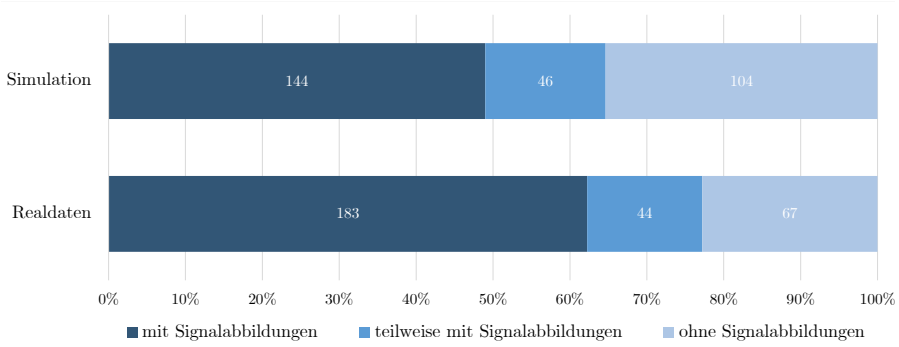



Abbildung 6.11: Verhältnisse der Anforderungen mit abgebildeten Annotationen

tationen auf Level 1 und 2 implizit in den Annotationen auf Level 3 enthalten sind. Daher ist die Gesamtzahl der abgebildeten Annotationen auf Level 1 und 2 gleich, mit 50 für die realen Daten und 23 für die Simulationsdaten. Darüber hinaus ist die Gesamtzahl der Annotationen auf Level 1 und 2 gleich, da diese Analyse in Bezug auf die 73 Annotationen auf Level 1 und 2 durchgeführt wird, die tatsächlich auf Level 3 verfeinert werden. Folglich ist das Verhältnis der abgebildeten Annotationen auf Level 1 und 2 gleich.

Die Ergebnisse auf Level 4 hängen ebenfalls direkt von der Abbildung auf Ebene 3 ab, da jede Annotation auf Ebene 4 eine Aggregation von Annotationen auf Level 3 ist.

Im Allgemeinen zeigt die Analyse, dass das Verhältnis der testbaren Annotationen innerhalb der Simulation für jedes Level selten die 50 %-Marke überschreitet. Die Auswertung der Realfahrdaten zeigt hingegen mehr Potential.

Die zweite Metrik ist die Abdeckung der Anforderungen, die in Abbildung 6.11 dargestellt ist. Die Abbildung zeigt die Verhältnisse der Anforderungen, bei denen entweder alle enthaltenen Annotationen abgebildet werden (vollständig abgebildet), nur eine Teilmenge der Anmerkungen abgebildet werden (teilweise abgebildet) oder keine Annotation abgebildet wird (vollständig nicht abgebildet). Wie bei der vorherigen Analyse zeigen die Ergebnisse ein größeres Potenzial für die Teststufe mit Realfahrzeugen, da aufgrund einer vorhandenen Signalzuordnung potenziell mehr Anforderungen analysiert werden. Dies wird dadurch gekennzeichnet, dass 62 % der Anforderungen das Potential zeigen, das innerhalb der realen Daten vollständig verifiziert werden kann und nur 49 % innerhalb der Simulation. Die Anteile der teilweise abgebildeten Anforderungen sind zwischen den Testphasen sehr ähnlich, mit einem Unterschied von nur zwei Annotationen.

**Interpretation der Ergebnisse:** Die vorgestellten Ergebnisse der Analyse zeigen das Potenzial dieses Evaluationsschrittes, die Möglichkeit verschiedene Teststufen zu beurteilen - und damit die dritte Forschungsfrage zu beantworten. Explizit zeigt diese Auswertung,

dass die Testphase, in der die Anforderungen mit den realen Testfahrten verifiziert werden, mehr Potential aufweist. In Bezug auf Annotation und Anforderungsabdeckung übertrifft die Teststufe der Realfahrdaten die Simulationsteststufe.

In weiteren Untersuchungen müssen die fehlenden Signalabbildungen der Simulation analysiert werden, um die Aussagekraft der Simulation zu erhöhen.

In der Simulation konnte beispielsweise der Zustand der Fahrzeugtüren nicht simuliert und ausgewertet werden. Daraus folgt, dass Annotationen diesbezüglich nicht ausgewertet wurden.

Im Hinblick auf die realen Testfahrten bedeuten Annotationen, die nicht auf bestimmte Signale abgebildet werden konnten, dass einige Aspekte der Anforderungsspezifikation nicht explizit als Signale innerhalb des Fahrzeugs dargestellt werden. Für das TA-System des Industriepartners betraf dies Annotationen, beispielsweise von Baumaßnahmen (Betonpfeiler oder Masten), die das Fahrzeug erkennen soll. Abstrakte Meta-Informationen wie diese werden nicht als Teil der fahrzeuginternen Signale aufgezeichnet und können daher nicht ausgewertet werden.

#### 6.2.4.4 *Evaluation des Vergleichs von Testfällen*

Die vierte Evaluation untersucht das zuvor beschriebene Potential und bewertet wie gut die Annotationen die Anforderungen erfüllen. Die Evaluation betrachtet die tatsächliche Durchführung des Tests und die qualitative Analyse, ob die Annotationen für die verschiedenen Tests erfüllt sind. In dieser Auswertung wird untersucht, wie gut die Tests mit der Spezifikation übereinstimmen und wie sich die verschiedenen Teststufen hinsichtlich ihrer Annotation und Anforderungsabdeckung unterscheiden.

Die Annotations- und Anforderungsabdeckung ist in Kapitel 6.2.3.2 definiert. Die Abdeckung bezieht sich auf die Annotationen die durch die jeweiligen Logdatenauswertung erfüllt werden. Für die Anforderungsabdeckung werden diese Annotationen bezogen auf die Spezifikation ausgewertet.

Zur Anwendung der Metriken wird die Auswertung der Logdaten in Bezug auf die Annotationen für jedes Fahrzeug in der Simulation und jede Fahrt der Fahrdaten durchgeführt. Die im Folgenden berichteten Ergebnisse sind für zwei unterschiedliche Untersuchungen nützlich. Erstens als Mittel zur Bewertung des korrekten Systemverhaltens. Zweitens zur Bewertung wie Testfälle einer Testphase die Merkmale des Systems abdecken. Welche Analyse im Vordergrund steht ist abhängig von den Nutzer:innen und kann vom Kontext des Testansatzes variieren.

Zur Untersuchung der Annotationsabdeckung zeigt die Abbildung 6.12 das Verhältnis von erfüllten und nicht erfüllten Annotationen pro

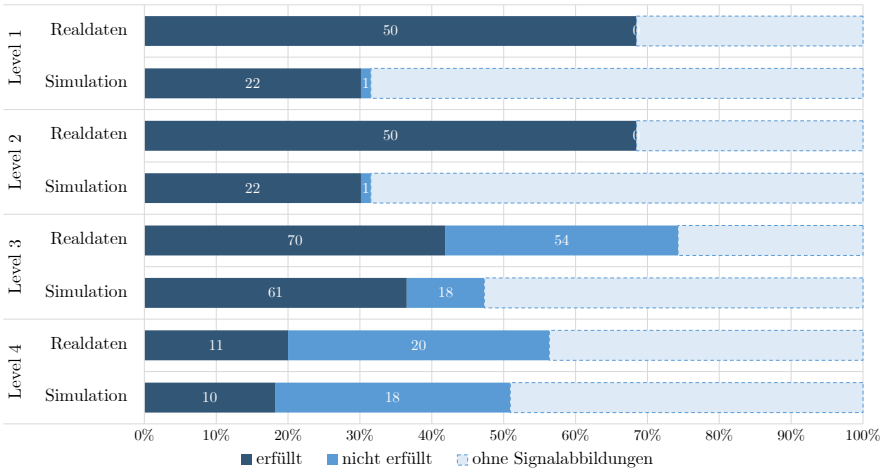



Abbildung 6.12: Verhältnisse der erfüllten Annotationen je Level

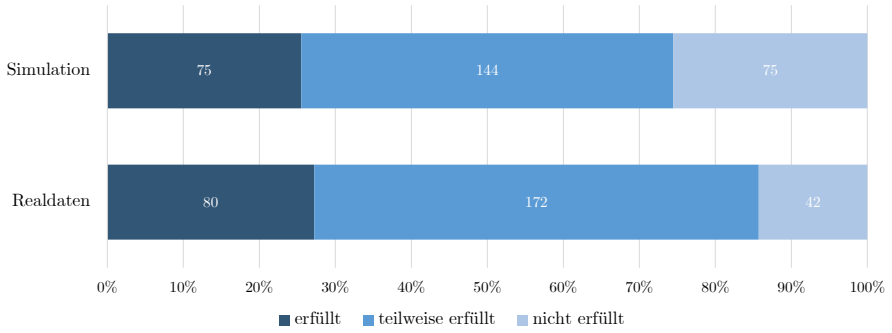



Abbildung 6.13: Verhältnisse der erfüllten Anforderungen

Level. Zusätzlich wird das Potenzial durch die Einbeziehung der Anzahl der Annotationen, die nicht abgebildet werden, veranschaulicht.

Im Level 1 und 2 werden alle abgebildeten Annotationen von den Realfahrdaten erfüllt. Bezogen auf die Simulation ist bis auf eine Annotation eine vollständige Abdeckung erreicht.

Im Gegensatz dazu steht die Analyse der Annotationen in Level 3. Während die Anzahl der erfüllten Annotationen innerhalb der realen Fahrdaten insgesamt höher ist als der Simulation, decken die Testfälle innerhalb der Fahrzeugteststufe nur 56 % ihres theoretischen Potentials ab. In der Simulation decken die erfüllten Annotationen 77 % der Annotationen ab, die tatsächlich getestet wurden.

Auf Level 4 zeigen die beiden Teststufen sehr ähnliche Erfüllungsquoten, wobei 35 % und 36 % der abgebildetet Annotationen erfüllt sind, was 11 und 10 tatsächliche Annotationen für reale und Simulationsdaten bedeutet.

Durch die Referenzierung der Ergebnisse auf die Anforderungsspezifikation wird eine Aussage zur Erfüllung der natürlichsprachlichen Anforderungen ermöglicht. Es ist zu beachten, dass eine Anforderung

sowohl erfüllte als auch nicht erfüllte Annotationen enthalten kann. Abbildung 6.13 gibt das Verhältnis von Anforderungen in Bezug auf den Grad der Erfüllung an. Es ist zu beachten, dass als Ausgangspunkt nur die Anforderungen im Fokus stehen, die tatsächlich Annotationen enthalten.

Bezogen auf die durchgeführte Simulation enthalten 75 (25,5 %) der 294 Anforderungen ausschließlich erfüllte Annotationen. Die Auswertung der realen Fahrten zeigt, dass 80 (27,2 %) der betrachteten Anforderungen nur erfüllte Annotationen enthalten. Im Gegensatz dazu sind 75 (25,5 %) der Anforderungen innerhalb der Simulation bezüglich der inhärenten Annotation und 42 (14,3 %) bezüglich der realen Fahrdaten vollständig unerfüllt. Daher decken die Testfahrten mehr von der funktionalen Implikation des TA-Systems ab, da der Anteil der teilweise abgedeckten Anforderungen größer ist als in der Simulation.

Die Annotationen werden im Hinblick auf ihre Erfüllung in Bezug auf ihr Level weiter analysiert. Abbildung 6.12 zeigt den durchschnittlichen Prozentsatz der erfüllten Annotationen pro Anforderung für jedes Level. Das Verhältnis der erfüllten Annotationen je Anforderungen ist in der Auswertung der Realfahrdaten in jedem Level höher.

**Interpretation der Ergebnisse:** Die Auswertung des Vergleichs von Testfällen zeigt, wie umfangreich Tests innerhalb der verschiedenen Teststufen die Spezifikation verifizieren. Die Erkenntnisse liefern einen Beitrag zur Beantwortung der vierten Forschungsfrage. Die Ergebnisse zeigen, dass in dieser spezifischen Anwendung die Testfahrten mehr Eigenheiten des TA-Systems abdecken. Diese Ergebnisse sind zu erwarten, da die vorangegangene Potenzialanalyse bereits einen Vorteil der Fahrzeugtestphase aufgezeigt hat.

Diese Analyse ist das letztendliche Ziel der Anwendung der Multilevel Markup Language, da dies die Ergebnisse sind, über die die Ingenieur:innen sofort nachdenken und die er zur Funktions- oder Testfalloptimierung verwendet.

### 6.2.5 *Validität*

Es gibt mehrere Risiken der Validität dieser experimentellen Bewertung, die im folgenden beschrieben und diskutiert werden.

**Interne Risiken der Validität.** Das erste interne Risiko der Validität ist das TA-System der Simulation. Die Verwendung der tatsächlichen Implementierung des Systems vom Partner aus der Automobilindustrie hätte realitätsnähere Ergebnisse produziert. Da der produktive Code einer Geheimhaltungsklausel unterliegt, konnte dieser für das Experiment nicht beschafft werden. Um das Risiko zu minimieren, wird das TA-System in enger Übereinstimmung mit den Spezifikationen des Originalsystems entwickelt.

Das zweite Riskio der internen Validität ist in der Abbildung von Annotationen auf die Signale. Für die Abbildung der Signale und Werte des Systems, wie sie in den Testphasen vorliegen, auf die Annotationen ist Domänenwissen erforderlich. Für die Validierung der Abbildung standen jedoch keine Ingenieur:innen des Automobilpartners zur Verfügung. Um dieser mangelnden Beteiligung der Entwickler:innen entgegenzuwirken, wurde der Abbildungsprozess von beratenden Experten aus unserer Forschungseinrichtung bewertet.

Schließlich sind die Annotationen selbst ein potenzielles Risiko der Validität. Da diese jedoch für die Nutzer:innen immer subjektiv sind, werden sie nicht als ein tatsächliches Risiko der Validität angesehen. Die Person, die die Spezifikation annotiert, sollte über Hintergrund- und Expertenwissen über das System verfügen. So gelten die Annotationen als valide. Um das potenzielle Risiko in unserem Experiment zu verringern, werden die Annotationen in diese Experiment von zwei Experten überprüft.

**Risiko der Konstruktvalidität.** Zusätzlich gibt es ein weiteres Risiko des Gesamtexperiments, das durch die Auswahl des Szenarios entsteht. Im Idealfall sollten die Szenarien der verschiedenen Testphasen bis zu einem gewissen Grad standardmäßig aufeinander abgestimmt sein. Dieser Mangel an Voreinstellung manifestiert sich in der Wahl des Simulationsszenarios, das aus Gründen der Vergleichbarkeit das gleiche sein sollte wie in den Testfahrten. In der Praxis ist diese Ausrichtung jedoch oft nicht gegeben. Daher wurde ein sehr realistisches Simulationsszenario gewählt, um den Verkehr realitätsnah abzubilden und damit realistisches Systemverhalten zu untersuchen.

### 6.2.6 *Diskussion*

Im Folgenden diskutieren wir die vier verschiedenen Analyseansätze und die Ergebnisse.

**Diskussion zur Anwendbarkeit von Annotationen.** Die vom Test unabhängige statische Analyse der Annotationen zeigt die allgemeine Anwendbarkeit der Multilevel Markup Language auf Anforderungsspezifikationen in großem Umfang. Ein signifikanter Anteil der funktionalen Implikationen der Spezifikation kann mithilfe der Annotationen abgedeckt werden. Alle Elemente der vier Levels werden hierbei angewendet. Dabei sind Level 4 Annotationen nur in einer kleineren Teilmenge von Anforderungen vorhanden. Nutzer:innen des Partners aus der Automobilindustrie, die mit dem TA-System vertraut sind, hätten mit hoher Wahrscheinlichkeit mehr kausale Zusammenhänge hergestellt, dennoch zeigen die Annotationen des Experiments das proportionale Verhältnis. Zusätzlich ist zu beachten, dass dieser Prozess sehr subjektiv ist und unterschiedliche Nutzer:innen höchstwahrscheinlich unterschiedliche Annotationen erstellen. Die Sprache kann in umfangreichen Spezifikationen eingesetzt werden und stellt

eine Basis für weitere Analysen dar. Der Aufwand des Annotationsprozesses liegt unter sechs Stunden und ist vermutlich geringer als der Aufwand zur Erstellung eines formalen Modells aus einer Spezifikation von fast 300 Anforderungen.

**Diskussion zur Ähnlichkeit der Teststufen.** Die Bewertung der Ähnlichkeit der Teststufen dient als Mittel, um festzustellen, wie gut die Simulation das reale Fahrerverhalten und den Verkehrsfluss abbildet. In der vorgestellten Anwendung unterschied sich die Simulation erheblich von den realen Daten. Insbesondere die Geschwindigkeiten der Fahrzeuge sind anders verteilt als in den realen Fahrdaten. Dies beeinträchtigt jedoch nicht notwendigerweise die Aussagekraft der Simulation. Die Tatsache, dass die Höchstgeschwindigkeit in der Simulation etwa 30 km/h niedriger ist als in den realen Fahrversuchen, kann einen Einfluss auf die Beurteilung des Systems haben. Keine Aussage über das Verhalten des Systems bei diesen hohen Geschwindigkeiten kann getroffen werden, wenn das System nur mit der Simulation getestet wird. Weiterhin kann der Vergleich systemabhängiger Faktoren wichtige Erkenntnisse über die Aussagekraft und Bedeutung der Teststufen liefern. Wenn beispielsweise eine Simulation im Vergleich zu einer weiteren Teststufe eine wesentlich geringere Aktivierung eines Signals zeigt, ist die Aussagekraft dieser Simulation bezüglich dieses Signals zu hinterfragen.

Diese Analyse ist ein leistungsfähiges Mittel zur Beurteilung der Validität eines Teststufenkontextes, wie der vorgestellte Vergleich zeigt. Der Schwerpunkt dieser Analyse liegt nicht auf dem eigentlichen, umfangreichen Vergleich der verwendeten Simulation, sondern zielt darauf ab, das Potential des Ansatzes aufzuzeigen. Die vorgestellte Analyse dient als relevanter Kontext für die nachfolgenden Analysen.

**Diskussion der Passgenauigkeit der Teststufen.** Aus den Ergebnissen des Vergleichs der Teststufen lassen sich zwei Schlussfolgerungen ziehen. Zum einen lässt sich erklären, warum der Abbildungsprozess der realen Fahrten nicht 100 % erreicht, obwohl die Spezifikationen, die Kommunikationsmatrix und die Implementierung des Systems aus einem Unternehmen stammen. Da in den Logdaten nur interne Signale enthalten sind, können Annotationen zur Umgebung nicht ausgewertet werden. Beispielsweise konnten Annotationen wie `Baustelle` oder `Höhe des Radweges` $\geq$ `20cm` nicht zugeordnet werden. Die Analyse solcher Annotationen würde eine umfangreichere Datenbank erfordern, in der auch Aussagen von aggregierten Signalen vorhanden sind. Dies ist ein Nachteil der Testphase, das ohne die vorgestellte Analyse für Nutzer:innen vielleicht nicht so offensichtlich ist.

Zweitens machen die verschiedenen Level der Textphrasen die Testphasen vergleichbar. Die Simulation enthält weniger Signale, da der Simulator Abstraktionen von Fahrzeugen verwendet. In unserem Experiment werden die Fahrzeuge durch den Simulator SUMO bereitgestellt. Annotationen beispielsweise mit dem Text `Türen`, `Ausstiegswarnung`

oder `Fahrrad = erkannt` können nicht abgebildet werden, da der Simulator SUMO diese Informationen nicht liefert. Ein zweiter Grund ist der Entwicklungsstand des Systems. Das TA-System in der Simulation ist eine prototypische Implementierung und bietet nicht die volle Funktionalität des ursprünglichen TA-Systems.

Zusammenfassend zeigt sich, dass die Bewertung der Teststufenübereinstimmung eine wichtige Aussage über das Systemverifikationspotenzial einer Teststufe liefert.

**Diskussion des Vergleichs von Testfällen.** Die Ergebnisse des Testfallvergleichs korrelieren mit den Ergebnissen der Teststufenübereinstimmung. Da die Testphase mit den realen Fahrdaten ein höheres Verifikationspotential bezüglich der Annotation zeigte, manifestiert sich dies im Testfallvergleich. Die Tests in dieser Teststufe decken eine größere Anzahl der Anforderungen und der darin enthaltenen Annotationen ab.

Für das Experiment stehen nicht die absoluten Zahlen im Fokus der Untersuchung, sondern die Auswertungsmöglichkeiten, die es erlauben, zwei sehr komplexe Teststufen und die sie konstituierenden Testfälle zu vergleichen. Neben dem Vergleich wird auch die Auswahl einer Teststufe erleichtert. Zusätzlich kann eine weiterführende Optimierung der Testfälle stattfinden. Die Untersuchung der Aussagekraft von Testfällen im Hinblick auf die Anforderungsabdeckung ist ein weiteres Instrument zur systematischen Bewertung der funktionalen Verifikationsmöglichkeiten der angewandten Tests.

**Allgemeine Anwendbarkeit des Ansatzes.** Der Ansatz kann verwendet werden, um Informationen über das Testverfahren in jeder Testphase zu sammeln.

Angesichts der Annotationen zu einer Anforderungsspezifikation sind die vorgestellten Auswertungen und Analyseausgaben vollständig automatisiert.

Neben umfassenden Auswertungen auf der Grundlage natürlichsprachlicher Anforderungen sind auch umfassende Auswertungen der Testfälle und Testphasen möglich. Dies entspricht den Verifikationsansätzen in der Industrie, wo mehrere Teststufen für einen erfolgreichen Softwareentwicklungsprozess notwendig sind. Die Auswertung des Teststufenvergleichs ermöglicht es den Entwickler:innen und Tester:innen, Entwicklungsstände und Testoptionen besser zu vergleichen.

Mit den abgeleiteten Aussagen können Nutzer:innen nicht nur funktionale Mängel des Systems beheben, sondern auch Defizite innerhalb des Tests selbst.

**Limitationen.** Unser Ansatz ist durch einige externe Aspekte eingeschränkt. Die dargestellten Ergebnisse sind stark von der Qualität der Anforderungen abhängig. Dazu gehören mehrdeutige Anforderungen, die von den Nutzer:innen missverstanden werden und zu Annotationen führen, die sich nicht abbilden lassen, oder zu Ergebnissen, die

falsch interpretiert werden. Unser Ansatz zeigt nur die Ergebnisse der annotierten Textphrasen an. Wenn wichtige Informationen nicht als Anforderungen dokumentiert sind, kann der gezeigte Ansatz wichtige Aspekte nicht berücksichtigen. Eine weitere Einschränkung des Ansatzes ist der subjektive Annotationsprozess. Die entwickelte Sprache ist bewusst eine manuelle Methode für die Nutzer:innen, um beobachtbare Objekte auszuwählen. Für valide Ergebnisse ist während des Annotationsprozesses Domänenwissen erforderlich. Da keine Überprüfung der Annotationsgültigkeit vorgesehen ist, werden schlechte oder falsche Annotationen zu schlechten Ergebnissen führen.

Darüber hinaus werden komplexe Anforderungen möglicherweise nicht markiert, weil die Sprache zu wenige Elemente für geeignete Annotationen enthält. Wie in 2.1 beschrieben, ist die Sprache zum Teil an die bestehenden, etablierten Syntaxen angelehnt. In der Studie konnten alle gewünschten Informationen extrahiert werden. Es ist jedoch möglich, dass die Sprache nicht alle Aspekte der Spezifikation abdeckt.

# 7

# TEIL 4: DARSTELLUNGSPROZESS

Der Darstellungsprozess ist der vierte Schritt des Testansatzes und umfasst die Darstellung der Annotationen der Nutzer:innen, die tabellarische Darstellung von extrahierten Annotationen und die Ergebnisdarstellung im Anschluss an den Auswertungsprozess in den natürlichsprachlichen Anforderungen. Kapitel 7.1 beschreibt die tabellarische und die anforderungsbezogene Darstellung von Annotationen und Ergebnissen und zeigt die Vor- und Nachteile beider Darstellungsformen.

Im Gesamtansatz sind die Nutzer:innen mit Ausnahme des Auswertungsprozess in alle Teilprozesse aktiv involviert. Um die Bedeutung des Gesamtansatzes, die praktische Relevanz der Testmethode und den Nutzen der grafischen Darstellung von Annotationen in Anforderungsdokumenten zu untersuchen, wurden Repräsentanten aus der Industrie in einem Studiensetting befragt. Kapitel 7.2 beschreibt die Durchführung der Studie und bewertet den Gesamttestansatz entsprechend der Resonanz.

## 7.1 VISUALISIERUNG VON ANNOTATIONEN

Die Annotationen werden im Testprozess auf zwei unterschiedlichen Arten dargestellt. Zum einen in tabellarischer Form und zum anderen als eingefärbte Annotation einer ausgewählten Anforderungen. Unabhängig davon, ob die Annotation während des Annotationsprozesses oder während des Darstellungsprozesses angezeigt wird, sind beide Darstellungsformen synchron und beziehen die gleichen Informationen ein. Beide Darstellungsformen zeigen jedoch unterschiedliche Stärken in der Veranschaulichung einzelner Aspekte.

### 7.1.1 *Tabellarische Darstellung von Annotationen*

Die tabellarische Darstellung enthält grundsätzlich für jedes Level eine Tabelle, in der die Annotationen zeilenweise dargestellt sind. Für jede Annotation steht die ID der Anforderung zur Verfügung, in der die Annotation durch die Nutzer:innen erstellt wurde. Die Informationen in den Tabellen variieren und sind levelabhängig.

In der Tabelle der Level 1 Annotationen ist zusätzlich zur ID der Scope (System oder Environment) und der annotierte Text aufgeführt.

Die Tabelle der Level 2 Annotationen enthält die Spalteneinträge analog zu Level 1, erweitert um den Annotationstypen (State, Value, Event, Time).

In der Level 3 Tabelle sind unter anderem Annotationstyp und Annotationsscope enthalten. Darüber hinaus ist der annotierte Text (beispielsweise *„Tür=geöffnet“*) getrennt nach Name (*„Tür“*), Wert (*„geöffnet“*) und Operator („=“) sowie angehängte Time-Annotation spaltenweise aufgeführt.

Abbildung 7.1 zeigt einen exemplarischen Auszug einer Level 4 Tabelle und ihren Spalteneinträgen. Neben den zwei verwendeten Level 3 Annotationen sind die zugehörigen IDs aufgelistet. Zusätzlich beschreibt die Spalte *Condition Type* den verwendeten Typ ({L3}-Pre-Condition oder {L3}-Trigger) der Level 4 Annotationsbedingung.

Level 1 | Level 2 | Level 3 | Level 4

| Condition Type | Condition | Action | Condition ReqID | Action ReqID |
|---|---|---|---|---|
| Precondition | fahrzeuggeschwindigkeit > 0 km/h | acc = verfügbar | R2 | R2 |
| Precondition | fahrzeuggeschwindigkeit < 190 km/h | acc = verfügbar | R2 | R2 |
| Precondition | abstand <= 50 m | führenden fahrzeugs = anfahren | R6 | R6 |
| Precondition | vorausfahrenden fahrzeugs = abbiegevorgang | acc = deaktiviert | R7 | R7 |

Abbildung 7.1: Exemplarische Darstellung von Level 4 Annotationen

Die Ergebnisse nach einem und mehreren Testdurchläufen werden ebenfalls tabellarisch dargestellt. Hierzu wird eine weitere Spalte hinzugefügt, in der die Ergebnisse der jeweiligen Level angezeigt werden. Die grafische Ergebnispräsentation der Value Annotationen liegt separat als Diagramm vor. In der Tabelle wird lediglich die Information mitgeteilt, ob Ergebnisse für diese Auswertung vorhanden sind.

Abbildung 7.2 zeigt die Ergebnisdarstellung der Annotationen aus Abbildung 7.1.

Level 1 | Level 2 | Level 3 | Level 4

| Condition Type | Condition | Action | Condition ReqID | Action ReqID | Results |
|---|---|---|---|---|---|
| Precondition | fahrzeuggeschwindigkeit > 0 km/h | acc = verfügbar | R2 | R2 | Fulfilled: 100%, Unfulfilled: 0 |
| Precondition | fahrzeuggeschwindigkeit < 190 km/h | acc = verfügbar | R2 | R2 | Fulfilled: 100%, Unfulfilled: 0 |
| Precondition | abstand <= 50 m | führenden fahrzeugs = anfahren | R6 | R6 | Fulfilled: 66%, Unfulfilled: 33% |
| Precondition | vorausfahrenden fahrzeugs = abbiegevorgang | acc = deaktiviert | R7 | R7 | Fulfilled: 100%, Unfulfilled: 0 |

Abbildung 7.2: Exemplarische Darstellung von Level 4 Annotationen

Die tabellarische Darstellung der Annotationen sowohl im Annotations- als auch im Darstellungsprozess zeigt alle enthaltenen Annotationen nach Level sortiert an. Diese Darstellung ermöglicht den Nutzer:innen das Gesamtverhalten des spezifizierten Systems besser nachvollziehen zu können und soll die Extraktion von Informationen für weitere Schritte im Entwicklungsprozess erleichtern. Während der Ergebnisdarstellung liefert die tabellarische Darstellung der Testergebnisse einen umfassenden Überblick über das Verhalten des Systems im Testdurchlauf.

### 7.1.2 *Anforderungsbezogene Darstellung von Annotationen*

Die zweite Möglichkeit der Annotationsdarstellung ist eine anforderungsbasierte farbliche Darstellung, die zeitgleich mit den ausgewählten Anforderungen angezeigt wird. In dieser Darstellungsform sehen die Nutzer:innen alle Annotationen einer Anforderung. Liegen Ergebnisse im Anschluss an den Auswertungsprozess vor, werden die Ergebnisse unterhalb der Annotation angezeigt. Eine Ausnahme bilden die Value Annotationen. Die grafischen Ergebnisse sind separat als Diagramm verfügbar. Die restlichen Annotationen und die dazugehörigen Ergebnisse sind in den Farben Grün, Gelb oder Rot, wie in Kapitel 6.1.1 erläutert, dargestellt. Abbildung 7.3 zeigt eine exemplarische Anforderungen *R6* mit den dazugehörigen Annotationen und Ergebnissen im darunter liegenden Feld. Die Dauer (*Duration*) wird in der Anzahl der Simulationsschritte angegeben. Die tatsächlich benötige Zeit hängt demnach von der Frequenz der Systemausführung ab.

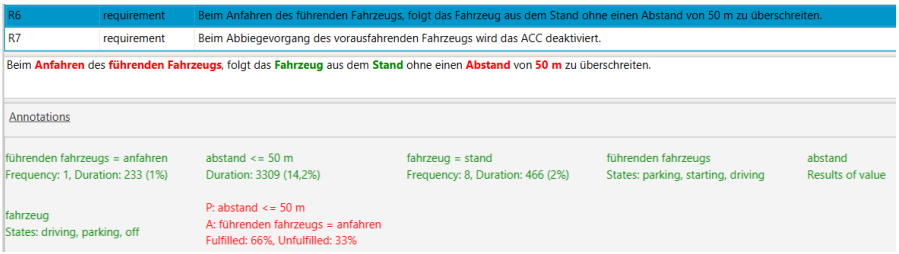



Abbildung 7.3: Exemplarische Darstellung der anforderungsbasierten Darstellung von Annotationen

Die farbliche Auswertung bezieht sich auf zwei Aspekte der Darstellung. Zum einen sind die Annotationen mit den Ergebnissen eingefärbt (Rot, Gelb oder Grün), zum anderen sind die dazugehörigen annotierten Textbestandteile analog dazu eingefärbt. Bei mehrfach annotierten Texten werden die Farben in absteigender Kritikalität (Rot $\rightarrow$ Gelb $\rightarrow$ Grün) dargestellt. Die Nutzer:innen können so die Ergebnisse des Testfalls übersichtlich den einzelnen Bestandteilen der Anforderungen zuordnen und erhalten eine farblich kodierte Aussage zur Erfüllung der Anforderung.

## 7.2 STUDIE ZUR NUTZBARKEIT VON ANNOTATIONEN IN NATÜRLICHSPRACHLICHEN ANFORDERUNGEN

Der Gesamttestansatz basiert auf der Annotation von Anforderungen mithilfe der Multilevel Markup Language. Im Anschluss an den Annotationsprozess und aus einer Systemausführung resultieren Testergebnisse, welche die Nutzer:innen im Softwareentwicklungsprozess unterstützen und die Beurteilung der Softwarequalität erleichtern. Hierbei sind die Nutzer:innen im Gesamttestprozess direkt oder indi-

rekt in drei der vier Prozessschritte involviert. Im Annotationsprozess steht die manuelle Annotierung von Anforderungen im Vordergrund. Im darauffolgenden Abbildungsprozess findet die manuelle Signalabbildung der Annotationen auf Signale der Testdurchführung statt. Der Auswertungsprozess ist vollständig automatisiert und stellt die Ergebnisse bereit, die den Nutzer:innen im Darstellungsprozess angezeigt werden. Zur Evaluierung der Schnittstelle zwischen Testansatz und Nutzer:innen wurden Probanden aus der Industrie in einem standardisierten Studiendesign befragt. Die Befragung zielte auf die Einschätzung der praktischen Anwendbarkeit des Ansatzes im Umfeld von Requirement- und Testingenieur:innen ab. Insbesondere wurde dabei die Bewertung Darstellungsform (tabellarisch und anforderungsbasiert) erfragt.

### 7.2.1 *Kontext*

In der Studie werden die Berührungspunkte des Ansatzes mit den Nutzer:innen mit Fokus auf der Annotations- und Ergebnisdarstellung evaluiert.

Den Probanden werden acht Anforderungen zur Verfügung gestellt, anhand derer sie die Funktionsweise des Adaptive Cruise Control (ACC) Fahrerassistenzsystems nachvollziehen [99]. Die in den acht Anforderungen beschriebene Funktion ist die folgende:

Das System behält eine vom Fahrer definierte Geschwindigkeit bei. Bei erkanntem vorausfahrenden Fahrzeug reguliert das System eigenständig die Geschwindigkeit und den Abstand zum erkannten Fahrzeug. Hierbei beschleunigt das ACC-System das Fahrzeug bis zum Erreichen der definierten Geschwindigkeit und bremst das Fahrzeug gegebenenfalls bis zum Stillstand auf 0 km/h ab. Abbildung 7.4 zeigt das Verhalten des Systems im grünen Fahrzeug bei der Erkennung des vorausfahrenden blauen Fahrzeugs.

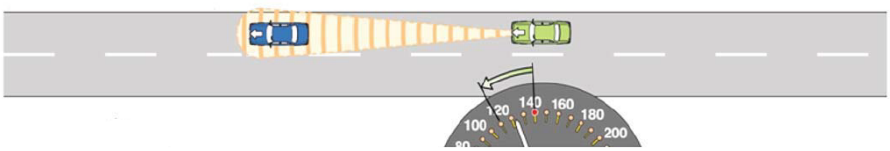



Abbildung 7.4: Erkennung eines vorausfahrenden Fahrzeugs durch das ACC-System.

Die vom Fahrer angestrebte Zielgeschwindigkeit von 140 km/h ist auf dem Tachometer mit einem roten Punkt markiert. Das ACC-System bremst jedoch auf eine Geschwindigkeit von 110 km/h ab, um einen angemessenen Sicherheitsabstand einzuhalten.

Die acht Anforderungen, die als Ausganspunkt der Studie dienen, sind natürlichsprachliche Anforderungen eines ACC-Systems. Die verwendeten Anforderungen sind an reale Anforderungen aus der

Industrie angelehnt und sind in der Formulierung und der Komplexität realitätsnah. Sie stellen einen Auszug einer Gesamtspezifikation dar und enthalten keine Überschriften oder Anforderungen vom Typ Informationen. Sie sind in unstrukturierter, uneingeschränkter natürlicher Sprache geschrieben ohne Verwendung von einheitlichen Sprachschablonen oder Formulierungsmustern.

Zur Evaluation des Gesamttestzyklus werden Ergebnisse einer fiktiven Systemausführung verwendet.

### 7.2.2 *Zielsetzung*

Ziel dieser Studie ist die Evaluierung der praktischen Nutzbarkeit und der Anwendbarkeit der unterschiedlichen Darstellungsformen im industriellen Umfeld. Hierzu wird eine fünfteilige interaktive Umfrage durchgeführt, in der die Probanden als Nutzer:innen den Annotations- und Darstellungsprozess experimentell erproben und anschließend Evaluationsfragen beantworten. Die Antworten lassen Rückschlüsse auf einen potenziellen Nutzen im zukünftigen Softwareentwicklungsprozess von neuen komplexen autonomen Systemen im industriellen Umfeld zu.

Zur Evaluierung der Darstellungsformen und praktischen Relevanz wurden vier Forschungsfragen definiert:

- **Forschungsfrage 1**: Sind die Elemente der Multilevel Markup Language zur Extraktion von relevanten zu testenden Informationen aus natürlichsprachlichen Anforderungen geeignet?
- **Forschungsfrage 2**: Können Annotationen genutzt werden um die Auswahl von Testfällen zu unterstützen?
- **Forschungsfrage 3**: Kann die Auswertung der Annotationen nach einem Testdurchlauf das Verständnis für das System erhöhen?
- **Forschungsfrage 4**: Verbessert die Darstellung der Ergebnisse in den Anforderungen das Verständnis der Testergebnisse bezogen auf die Anforderungen?

Die vier Forschungsfragen adressieren die Aspekte des Testansatzes, in die die Nutzer:innen aktiv involviert sind. Die erste Forschungsfrage bezieht sich auf den Annotationsprozess, deren Ziel die Auswertung von Testdurchläufen ist. Hierbei wird eruiert, wie sehr sich die Elemente der Multilevel Markup Language im praktischen Einsatz eignen.

Die zweite Forschungsfrage untersucht den Nutzen der erstellten Annotationen für die Testfallgenerierung.

Mit der dritten Forschungsfrage wird untersucht, wie sehr Annotationen den Auswertungsprozess, insbesondere für komplexe Systeme,

unterstützen. Für große komplexe Systeme stellt die Testfallauswertung von Gesamtsystemen bezogen auf Anforderungen eine Herausforderungen dar.

Die vierte Forschungsfrage adressiert den möglichen Nutzen der farblichen Annotation von Anforderungen anhand der Testergebnisse im Kontext der Systemspezifikation.

### 7.2.3 *Methodik*

Zur Evaluierung der Forschungsfragen wird eine interaktive Interviewstudie durchgeführt. Diese umfasst fünf Teile. Die ersten vier sind den einzelnen Forschungsfragen gewidmet und gliedern sich in einen Aufgaben- und einen Evaluationsteil. Im fünften Teil wird die Einschätzung der praktischen Relevanz des Gesamtansatzes erfragt.

Jeder Interviewtermin folgte einem fest vorgegebenen Schema, dauerte durchschnittlich 62 Minuten und wurde ausschließlich virtuell durchgeführt. Zur visuellen Unterstützung des Interviews sahen die Probanden alle Fragen und Informationen parallel auf einem geteilten Bildschirm. Nach einer kurzen Begrüßung und einer Vorstellung des Interviewers zeigt dieser den Probanden die Agenda. Sie beinhaltet eine fachliche Einleitung, die fünf Teile der Evaluation und einen kurzen Abschluss. In der Einleitung erklärte der Interviewer den Kontext des Testansatzes und erläuterte die Herausforderungen im Abgleich von natürlichsprachlichen Anforderungen und Systemtests. Anschließend wurden der Testansatz überblicksartig dargestellt und die vier Prozessschritte aufgezählt. Anhand eines Beispiels wurde insbesondere auf die Multilevel Markup Language eingegangen. Um allgemeine Verständnisfragen zu klären und um sicherzustellen, dass alle Fragen aus der Evaluation verstanden werden, stellt der Interviewer die folgenden drei Fragen:

- Wissen Sie, was mit einer natürlichsprachlichen Anforderung gemeint ist?
- Wissen Sie, was mit einer Annotation gemeint ist?
- Wissen Sie, was mit einem Testfall gemeint ist?

Sobald alle Fragen seitens der Probanden geklärt und beantwortet sind, startet die Evaluation.

Für alle Interviews standen acht natürlichsprachliche Anforderungen zur Verfügung, anhand derer der Proband den Testansatz anwendete. Tabelle 7.1 zeigt die acht Anforderungen aus einer Softwarespezifikation des ACC-Systems, welches in Kapitel 7.2.1 erläutert ist.

Die Anforderungen sind an reale Anforderungen eines Industriepartners angelehnt. Aufgrund der Geheimhaltungsklauseln können keine Originalanforderungen für die Evaluation eingesetzt werden.

Tabelle 7.1: Auszug einer Systemspezifiktaion des ACC-Systems

| ID | Anforderungstext |
|---|---|
| R1 | Das ACC reagiert ausschließlich auf vorausfahrende Fahrzeuge. |
| R2 | Das ACC ist bei einer Fahrzeuggeschwindigkeit zwischen 0 km/h und 190 km/h verfügbar. |
| R3 | Das ACC wählt zum Abbremsen eine Beschleunigung von $-3\,m/s^2$ um eine komfortable Distanzregulierung zu ermöglichen. |
| R4 | Das ACC beschleunigt mit $4\,m/s^2$ um eine komfortable Fahrzeugverfolgung zu erreichen. |
| R5 | Das ACC bremst das Systemfahrzeug bis zum Stillstand. |
| R6 | Beim Anfahren des führenden Fahrzeugs folgt das Fahrzeug aus dem Stand ohne einen Abstand von $50\,m$ zu überschreiten. |
| R7 | Beim Abbiegevorgang des vorausfahrenden Fahrzeugs wird das ACC deaktiviert. |
| R8 | Bei Deaktivierung wird die Beschleunigung des Systemfahrzeugs auf $0\,m/s^2$ gesetzt. |

Um die Konzentration auf die Anforderungen und Fragestellungen zu optimieren, liest der Proband alle acht Anforderungen laut vor.

Für die Beantwortung der Fragen steht eine numerische Rating-Skala von eins bis fünf zur Verfügung. Eins wird mit *sehr schlecht*, zwei mit *schlecht*, drei mit *mittelmäßig*, vier mit *gut* und fünf mit *sehr gut* assoziiert.

#### 7.2.3.1 *Teil 1: Evaluierung der Annotationsmethode*

Der erste Teil der Evaluation fokussiert sich auf die Annotationsmöglichkeiten. Hierbei steht das Annotieren von Texten in den natürlichsprachlichen Anforderungen aus Tabelle 7.1 im Vordergrund. Damit die Bewertung der Probanden nicht durch Designentscheidungen des eingesetzten Tools beeinflusst werden, startete die Evaluation mit einem Beispiel des Interviewers. Dabei erklärte der Interviewer wie Annotationen erstellt werden. Alle vier Level wurden vorgestellt und Texte in den Anforderungen exemplarisch annotiert. Anschließend wurde die Maussteuerung für den Probanden freigegeben. Der Interviewer stellte dem Probanden folgende vier Aufgaben:

AUFGABE 1: Erstellen Sie eine beliebige Annotation auf Level 1.

AUFGABE 2: Erstellen Sie eine beliebige Annotation auf Level 2.

AUFGABE 3: Erstellen Sie eine beliebige Annotation auf Level 3.

AUFGABE 4: Erstellen Sie eine beliebige Annotation auf Level 4.

Während der Aufgabe, hat der Proband jederzeit die Möglichkeit Unklarheiten durch Fragen an den Interviewer zu klären. Ziel ist es, den Annotationsprozess dem Probanden verständlich zu zeigen und eine Beurteilung der Annotationsmethode durch die experimentelle Nutzung zu erleichtern. Hierbei steht die richtige Annotation der Texte nicht im Vordergrund und wird nicht weitergehend ausgewertet.

Im Anschluss an den Aufgabenteil wurde die Mausfreigabe beendet. Der Proband sieht anschließend die Evaluationsfragen und bekommt sie vom Interviewer mündlich gestellt.

FRAGE 1: Wie sehr können Informationen mithilfe des Tools extrahiert werden?

FRAGE 2: Wie bewerten Sie die Methode um Informationen in den originalen Anforderungen zu markieren?

FRAGE 3: Wie werden die testbaren Informationen bisher extrahiert?

FRAGE 4: Wie sehr eignet sich die Methode um Informationen aus großen Lastenheften zu extrahieren im Vergleich zu Ihnen bekannten Methoden?

Für die Beantwortung der Fragen eins, zwei und vier steht die bereits vorgestellte Skala von eins bis fünf zur Verfügung. Für die Beantwortung der Frage 3 wird um eine kurze Antwort gebeten.

#### 7.2.3.2 *Teil 2: Evaluierung der Einsatzmöglichkeiten von Annotationen*

Im zweiten Teil steht die Nutzbarkeit von Annotationen und deren Einsatzmöglichkeiten im Vordergrund. Neben der Auswertung der Annotationen, können diese auch den Erstellungsprozess von Testfällen unterstützen. Im Vorfeld wurden alle Anforderungen des ACC-Systems mit dem höchstmöglichen Level annotiert. Die Annotationen sind, wie in Kapitel 7.1.1 dargestellt, tabellarisch verfügbar. Der zweite Teil der Evaluation untersucht den Nutzen der tabellarischen Darstellung für die Erstellung von Testfällen. Zu Beginn erhält der Proband die folgende Aufgabe unter Betrachtung der Anforderungen ohne Annotationen:

AUFGABE 1: Überlegen Sie sich drei Testfälle für das ACC und formulieren Sie diese kurz.

Anschließend werden die Anforderungen ausgeblendet und die tabellarische Darstellung der Annotationen gezeigt. Der Proband kann mithilfe der Maussteuerung die vier Tabellen der Annotationen anschauen und hört parallel Aufgabe 2:

AUFGABE 2: Überlegen Sie sich drei weitere Testfälle und formulieren Sie diese kurz (bei Betrachtung der Annotationen).

Hier steht der Vergleich der Testfallgenerierung im Vordergrund und die Frage wie sehr die Annotationen den Probanden unterstützen. Die tatsächlich genannten Testfälle stehen nicht im Fokus der Untersuchung und wurden nicht weiter ausgewertet.

Zur Evaluierung stellt der Interviewer zwei Fragen die jeweils mit der eingeführten Skala beantwortet werden:

FRAGE 1: Wie sehr helfen Annotationen bei der Erstellung von Testfällen?

FRAGE 2: Wie sehr helfen die Annotationen um das Verhalten des ACC besser zu verstehen?

#### 7.2.3.3 *Teil 3: Evaluierung der tabellarischer Darstellung von Testergebnissen*

Der dritte Teil evaluiert die Ergebnisdarstellung der Annotationen in tabellarischer Form. Diese ist exemplarisch in Abbildung 7.2 dargestellt. Um die Ergebnisse besser nachzuvollziehen und den gesamten Testansatz besser zu verstehen, gibt der Interviewer dem Probanden eine kurze Erläuterung zu Testdurchläufen und Logdatengenerierung. Zusätzlich wird auf die Abbildung der Annotationen auf die Signale kurz eingegangen. Die konkrete Signalabbildungstabelle wird den Probanden nicht gezeigt. Mit den Erläuterungen wird sichergestellt, dass alle Probanden das gleiche Verständnis des Kontexts haben und beurteilen die Ergebnisse, beziehungsweise die Ergebnisdarstellung mit gleichem minimalem Hintergrundwissen. Der Interviewer erklärt drei Arten wie Logdaten entstehen können. Erstens durch reale Testfahrten in denen alle Daten des Kommunikationsnetzwerks innerhalb eines Fahrzeugs aufgezeichnet werden. Zweitens ist es möglich in Testfahrten die in Simulationen stattfinden, Logdaten zu speichern. Drittens können Logdaten des Systems von vielen hunderten Fahrzeugen in Verkehrssimulationen aufgezeichnet werden. In allen drei exemplarischen Fällen werden die Logdaten ausgewertet und die Ergebnisse mit den Annotationen in Verbindung gesetzt. Der Proband sieht die Ergebnisse in tabellarischer Form und beantwortet folgende Evaluationsfragen:

FRAGE 1: Wie sehr helfen die gezeigten Auswertungen um das Verhalten des Systems im Testdurchlauf zu verstehen?

FRAGE 2: Wie sehr helfen die Auswertungen um zu beurteilen ob der Testfall erfolgreich war?

FRAGE 3: Wie sehr helfen die gezeigten Auswertungen um zu beurteilen ob der Testfall zu den Anforderungen passt?

#### 7.2.3.4 *Teil 4: Evaluierung der anforderungsbezogenen Darstellung von Testergebnissen*

Der vierte Teile der Evaluation fokussiert sich auf die Anwendbarkeit und Nützlichkeit der farblichen anforderungsbezogenen Darstellung der Ergebnisse. Abbildung 7.3 zeigt die exemplarische Darstellung der Anforderung *R6* aus Tabelle 7.1. Für die Betrachtung der Ergebnisse untersucht der Proband alle acht Anforderungen und die dazugehörigen angezeigten Annotationen. Parallel zur Betrachtung der Auswertungen muss der Proband folgende Aufgaben lösen:

AUFGABE 1: Finden Sie heraus, ob alle Annotationen getestet wurden, unabhängig vom Ergebnis.

AUFGABE 2: Finden Sie heraus, wie oft das Systemfahrzeug stand?

AUFGABE 3: Finden Sie heraus, welche Anforderung im Testfall nicht erfüllt wurde?

AUFGABE 4: Beurteilen Sie wie sehr der Testfall zu den Anforderungen passt.

Aufgabe 1 und Aufgabe 3 wurde von allen Probanden korrekt beantwortet. Aufgabe 2 beantworteten acht von neun Probanden (89 %) richtig. In Aufgabe 4 vergaben die Probanden durchschnittlich eine 4,1 von 5. Für die Evaluation stehen die korrekten Antworten jedoch nicht im Vordergrund der Evaluation. Sie richten lediglich den Fokus des Probanden auf die gezeigte Ergebnisdarstellung und erleichtert die Beantwortung der Evaluationsfragen:

FRAGE 1: Wie sehr konnten Sie beurteilen, ob der Testfall zu den Anforderungen passt?

FRAGE 2: Wie sehr helfen die farblichen Annotationen um Testfälle auszuwerten?

FRAGE 3: Wie sehr helfen die farblichen Anforderungen um Testfälle auszuwerten?

FRAGE 4: Wie sehr hilft die Ergebnisdarstellung insgesamt (Annotationen und Anforderungen) um die Erfüllung von Anforderungen zu beurteilen?

#### 7.2.3.5 *Teil 5: Evaluierung der praktische Relevanz*

Der fünfte Evaluationsteil enthält keine Aufgaben und dient der gesamtheitlichen Beurteilung des Testansatzes. In den Teilen eins bis vier hat der Proband die Schnittpunkte zwischen Testansatz und Nutzer kennengelernt und mithilfe der Aufgaben einen tiefgehenden Eindruck gewonnen. Im fünften Teil bewertet der Proband den Gesamttestansatz auf Grundlage der vorangegangenen Evaluationsteile

und den zusätzlichen Erläuterungen des Interviewers. Hierbei beantwortete der Proband unter anderem zwei Evaluationsfragen und hat anschließend die Möglichkeit Feedback in eigenen Worten zu geben.

FRAGE 1: Wie finden Sie den neuen Testansatz allgemein?

FRAGE 2: Finden Sie der gezeigte Testansatz kann das Testen zukünftiger komplexer automatisierter Fahrzeugfunktionen (Level 3 oder höher) verbessern?

Beide Fragen werden ebenfalls mit der Skala von eins bis fünf beantwortet, die für die vorangegangenen Evaluationsteile genutzt wurde. Nach der Durchführung aller Evaluationsteile bedankt sich der Interviewer und beendet die Evaluation.

### 7.2.4 *Ergebnisse*

Dieses Kapitel beschreibt die Ergebnisse des interaktiven Interviews angelehnt an die Struktur des methodischen Aufbaus. Für jeden Evaluationsteil werden die Ergebnisse dargestellt und anschließend interpretiert. Hierbei werden die Auswertungen in den Probandenkontext gesetzt und die Forschungsfragen aus Kapitel 7.2.2 beantwortet.

#### 7.2.4.1 *Auswertung der Teilnehmer*

An der interaktive Interviewstudie nahmen 9 Probanden teil. Da der Fokus der Evaluation auf dem Einsatz des entwickelten Ansatzes im praktischen Umfeld liegt, wurden ausschließlichen Probanden befragt die bereits berufstätig sind und die Ausbildung abgeschlossen haben. Die Probanden sind aus dem Bereich des Software Engineerings. Tabelle 7.2 zeigt eine Übersicht über die Probanden. Das Maximum liegt bei zehn Jahren Berufserfahrung, das Minimum liegt bei einem Jahr. Durchschnittlich sind die Probanden 4,4 Jahre beruflich tätig, der Median liegt bei 5 Berufsjahren.

Der entwickelte Testansatz ist branchenübergreifend einsetzbar. Speziell im Automotive Bereich ist die Logdatenanalyse und die Verwendung von natürlichsprachlichen Anforderungen jedoch stark im Fokus. Probanden aus der Automotive Branche sind besonders geeignet den Testansatz zu evaluieren. Die Probanden aus der Interviewstudie sind zu 56 % aus dem Automobilsektor.

Um die Auswertungen der Studie mit der Expertise der Probanden in den Kontext zu setzen, wurde im Vorfeld der Evaluation eine Selbsteinschätzung der Erfahrungen im Requirements- und Testmanagement durchgeführt. Hierbei schätzen die Probanden ihren Kenntnisstand zwischen 1 für *keine Erfahrungen* und 10 für *Ich bin Experte* selbst ein. Tabelle 7.3 zeigt die statistische Auswertung der Antworten.

Tabelle 7.2: Anzahl der Berufsjahre der befragten Probanden und ihre Position innerhalb der Firma mit dazugehöriger Branche.

| **Proband** | **Rolle** | **Domain** | **Berufserfahrung** |
|---|---|---|---|
| P1 | Requirements Engineer | Mobilität, Social Media | 1 |
| P2 | Solutions Architect | IT-Consulting | 1 |
| P3 | Testingenieur | Automotive | 1,5 |
| P4 | Software Consultant | IT-Consulting | 2 |
| P5 | Testingenieur Prototypen | Automotive | 5 |
| P6 | Software Entwickler | Automotive | 5 |
| P7 | Requirements Manager | Automotive | 6 |
| P8 | Software Entwickler, Testmanager | Automotive | 8 |
| P9 | IT-Consultant | Gebäudeautomation | 10 |

Tabelle 7.3: Selbsteinschätzung der Probanden im Requirements- und Test Engineering

| | **Mittelwert** | **Median** | **Min.** | **Max.** |
|---|---|---|---|---|
| Erfahrung im Requirements Engineering | 7,5 | 7 | 7 | 9 |
| Erfahrung im Test Engineering | 7,1 | 7 | 5 | 8 |

Das ausgeprägte Verständnis im Anforderungs- und Testmanagement der Probanden ermöglicht valide Aussagen zum entwickelten Testansatz.

#### 7.2.4.2 *Ergebnisse zur Evaluation der Annotationsmethode*

Durch die Ergebnisse dieses Kapitels lassen sich Rückschlüsse auf die Nutzbarkeit des Annotierungsprozess des Testansatzes ziehen. Abbildung 7.5 zeigt die durchschnittliche Bewertung der Fragen eins, zwei und vier, die im ersten Evaluationsteil gestellt wurden.

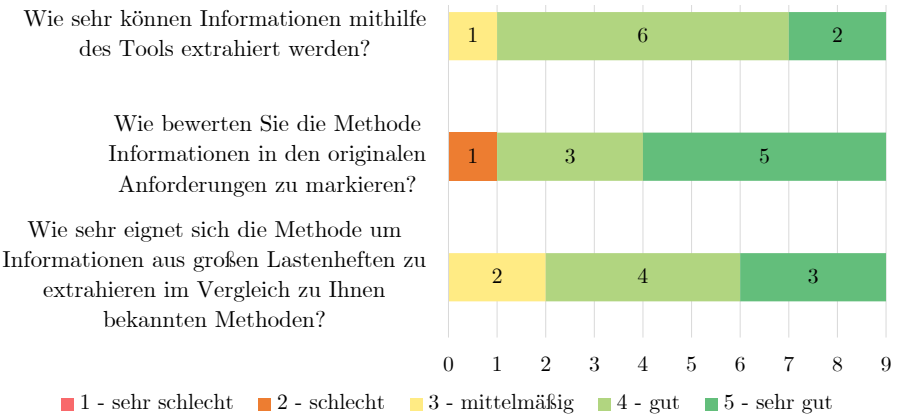



Abbildung 7.5: Anzahl der vergebenen Bewertungen je Frage des ersten Evaluationsteils

Im Gesamtdurchschnitt liegt der erste Evaluationsteil bei einer Bewertung von 4,2 von 5.

Die erste Frage bezieht sich auf die Informationsextraktion, die mithilfe der Methode durchgeführt wird. Hier wurde die Methode von allen Probanden zwischen 3 und 5 bewertet, der Median liegt bei 4 und der Durchschnitt bei 4,1.

Im Vergleich dazu, wurde die zweite Frage besser bewertet. Der Median aller Antworten liegt bei 5. Die Streuung der Bewertung ist deutlich größer und bewegt sich zwischen 2 und 5, wobei 5 der Maximalwert ist. Die Frage erhielt in diesem Evaluationsteil die höchste Durchschnittsbewertung von 4,3.

Die dritte Frage wird von den Probanden mit einer kurzen mündlichen Aussage beantwortet. Die Frage bezieht sich auf bisherige Extraktionsmethoden aus natürlichsprachliche Anforderungen und wurde von sieben der neun (77,8 %) Probanden mit *manuellem Lesen* als vorherrschende Extraktionsmethode beantwortet. Nach dem Kenntnisstand eines Probanden wird *parallel ein formales Model* zur Anforderungsspezfikation erstellt und daraus weiterführende Schritte abgeleitet. Ein weiterer Proband nannte die Extraktion von Informationen unter Verwendung der Gherkin-Syntax[1]. Diese basiert jedoch nicht auf natürlichsprachlichen Softwarespezifikationen.

Die vierte Frage hat engen Bezug zur dritten Frage, da hier die Annotationsmethode im Vergleich zur Antwort der dritte Frage bewertet wird. Das heißt, der Proband beurteilt die Annotationsmethode für zukünftige große komplexe Lastenhefte im Vergleich zu ihm bekannten Methoden. Die Bewertungen liegen zwischen 3 und 5. Der Median und der Durchschnitt liegen dicht beieinander, bei 4 und 4,1.

**Interpretation der Ergebnisse:** Die Analysen dieses Evaluationsteils bewerten das Annotieren von Textpassagen in Anforderungen und

1 https://cucumber.io/

beantworten die erste Forschungsfrage. Aus den Ergebnissen lässt sich schließen, dass die Annotationsmethode als eine geeignete Methodik eingeschätzt wird, um Informationen hervorzuheben und zu extrahieren. Speziell im Vergleich zu bisherigen vorherrschenden Methoden (manuelles Lesen) zeigt der Ansatz eine gute Anwendbarkeit. Bezogen auf die Erfahrungen der Probanden sind die Antworten sehr valide, da der durchschnittliche Erfahrungswert im Requirements Engineering der Probanden bei 7,2 liegt. Der Proband mit der höchsten Erfahrung im Requirements Engineering (Selbsteinschätzung 9 von 10) vergab im ersten Evaluationsteil eine Durchschnittsbewertung von 4,3, was über dem Gesamtdurchschnitt liegt. Das zeigt die positive Wahrnehmung der Annotationsmethode auch bei sehr erfahrenden Probanden.

#### 7.2.4.3 *Ergebnisse der Evaluation der Einsatzmöglichkeiten von Annotationen*

Im Anschluss an den Annotationsprozess stehen Annotationen in zwei Darstellungsformen, wie in Kapitel 7.1 beschrieben, zur Verfügung. Speziell Annotationen in der tabellarischen Darstellung können weitere Entwicklungsschritte unterstützen. Abbildung 7.6 zeigt die Durchschnittsbewertung der gestellten Fragen im zweiten Evaluationsteil.

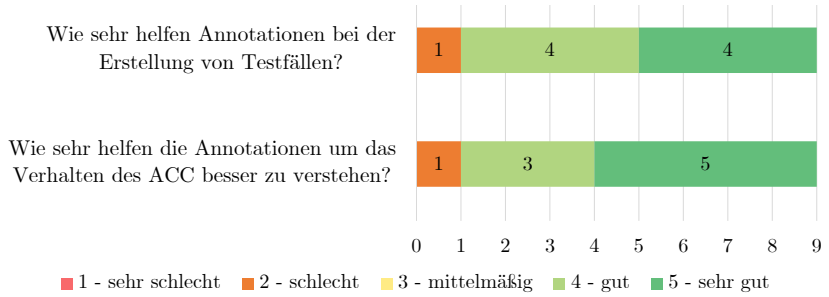



Abbildung 7.6: Anzahl der vergebenen Bewertungen je Frage des zweiten Evaluationsteils

Die erste Frage evaluiert die Nützlichkeit der Annotationen bei der Erstellung von Testfällen. Die Bewertungen decken einen Bereich von Minimum 2 bis Maximum 5 ab. Der Median liegt bei 4.

Die zweite Frage untersucht den Einfluss der Annotationen auf das Verständnis des spezifizierten Systems. Der Durchschnittswert der Antworten liegt bei 4,3 und damit unwesentlich über der Nützlichkeit für die Testfallgenerierung (4,2). Die Bewertungen liegen ebenfalls zwischen 2 und 5, der Median liegt bei 5.

Der Gesamtdurchschnitt des zweiten Evaluationsteils erreicht eine Bewertung von 4,3 bei einem möglichen Maximalwert von 5.

**Interpretation der Ergebnisse:** Die Ergebnisse des zweiten Evaluationsteils geben Rückschlüsse auf den möglichen Nutzen von er-

stellten Annotationen. Das ermöglicht die Beantwortung der zweiten Forschungsfrage. Die Einzelergebnisse und der Gesamtdurchschnitt von 4,3 zeigen, dass Annotationen das Erstellen von Testfällen und das Verständnis des Systems unterstützen. Dieser Evaluationsteil erhielt die höchste Gesamtbewertung der Studie, was zeigt, dass Annotationen nicht nur für die Testfallauswertung hilfreich, sondern auch für andere Entwicklungsprozesse einsetzbar sind. Zwei der neun Probanden haben in diesem Evaluationsteil ausschließlich eine 5 als Bewertung abgegeben. Die schlechteste Bewertung lag bei einem Durchschnitt von 3. Dieser Proband schätze seine Erfahrungen im Test Engineering mit 5 ein, was im Gesamtvergleich der kleinste Wert ist. Daraus lässt sich vermuten, dass der geringe Kontakt zum Testmanagement die Beurteilung der Nützlichkeit von Annotationen für den Testprozess erschwert.

#### 7.2.4.4 *Ergebnisse der Evaluation der tabellarischen Darstellung von Testergebnissen*

Im dritten Evaluationsteil steht die tabellarische Darstellung der Ergebnisse im Vordergrund. Der Proband sieht die Ergebnisse der Annotationen in tabellarischer Form und beurteilt den Mehrwert für das Systemverständnis, die Testfallauswertung und die Testfallauswahl für das System. Abbildung 7.7 zeigt für jede gestellte Frage die durchschnittliche Bewertung.

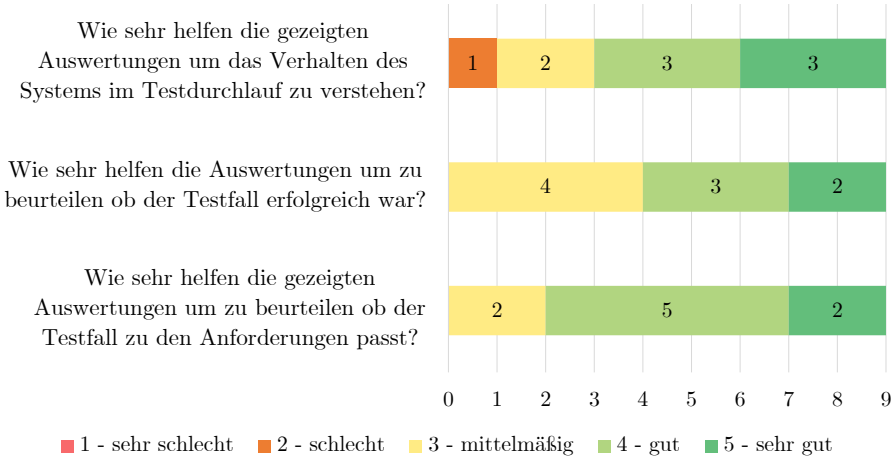



Abbildung 7.7: Anzahl der vergebenen Bewertungen je Frage des dritten Evaluationsteils

Bei der Beurteilung wie gut Testergebnisse in Annotation das Verständnis des Systems in Testdurchläufen unterstützen (Frage 1), vergaben die Probanden im Durchschnitt eine 3,9. Der Median liegt ebenfalls bei 4 wohingegen die Bewertungen zwischen einem Minimum von 2 bis zu einem Maximum von 5 reichen.

Etwas schlechter fällt die Bewertung der zweiten Frage aus. Obwohl alle Antworten mit einer Bewertung zwischen 3 und 5 vergeben wer-

den, der Median ebenfalls bei 4 liegt, erreicht der Durchschnittswert einen schlechteren Wert von 3,8.

Die bestbewertete Frage ist Frage 3 und bezieht sich auf die Beurteilung wie gut ein Testfall zu den Anforderungen passt. Der Durchschnittswert von 4, bei einem Median von ebenfalls 4 zeigt, dass die tabellarische Anzeige eine gute Unterstützung für die Auswahl von Testfällen ist. Hierbei liegen die gegebenen Antworten zwischen einem Minimum von 3 und einem Maximum von 5.

Der dritte Evaluationsteil erreicht einen Gesamtdurchschnitt von 3,9.

**Interpretation der Ergebnisse:** Die dritte Forschungsfrage kann mit den Auswertungen des dritten Evaluationsteils beantwortet werden. Die Ergebnisse zeigen, dass die Ergebnisdarstellung in tabellarischer Form zum einen das Systemverständnis in Testdurchläufen erhöht und zum anderen die Beurteilung von Testfällen erleichtert. Für die Aussage ob ein Testfall erfolgreich war, eignet sich die tabellarische Form, im Vergleich zur anforderungsbezogenen Darstellung, weniger. Der Wert von 3,8 ist im Vergleich zu allen anderen Antworten der zweitschlechteste. Die höchste Bewertung (durchschnittlich 5) vergab der Proband mit der meisten Berufserfahrung (10 Jahre; mehrere Firmen). Daraus lässt sich ableiten, dass Annotationen in tabellarischer Form auch von Nutzer:innen mit einem großen Erfahrungsschatz als nützlich und hilfreich gesehen werden.

#### 7.2.4.5 *Ergebnisse der Evaluation der anforderungsbezogenen Darstellung von Testergebnissen*

Im vierten Evaluationsteil steht die anforderungsbezogene Darstellung der Testergebnisse im Vordergrund. Die Ergebnisse geben zum einen Aufschluss darüber, wie die Darstellung die Auswahl von Testfällen unterstützen kann. Zum anderen zeigt die Auswertung wie gut die farbliche Ergebnisdarstellung, wie in Kapitel 7.1.2 beschrieben, die Auswertung von Testfällen ermöglicht. Abbildung 7.8 zeigt die durchschnittliche Bewertung der gestellten Fragen.

Dieser Evaluationsteil enthält vier Fragen und erreicht im Gesamtdurchschnitt eine Bewertung von 4,2. Die erste Frage eruiert die Unterstützung für die Beurteilung, ob der Testfall zu den Anforderungen passt. Die Probanden vergaben eine Bewertung von durchschnittlich 3,7, bei einem Maximum von 5 und einem Minimum von 1. Die gesamte Bewertungsskala wurde dabei ausgeschöpft. Der Median liegt bei 4.

Eine der Kerneigenschaften der Methode ist die Darstellung der Ergebnisse in den natürlichsprachlichen Anforderungen. Frage 2 bezieht sich auf die Darstellung der Ergebnisse mithilfe der farbigen Annotationen, wohingegen Frage drei sich auf die farbigen Anforderungen bezieht. In Frage zwei wurde mit einem Minimum von 3 und einem Maximum von 5 bei einem Median von 5, eine durchschnittliche

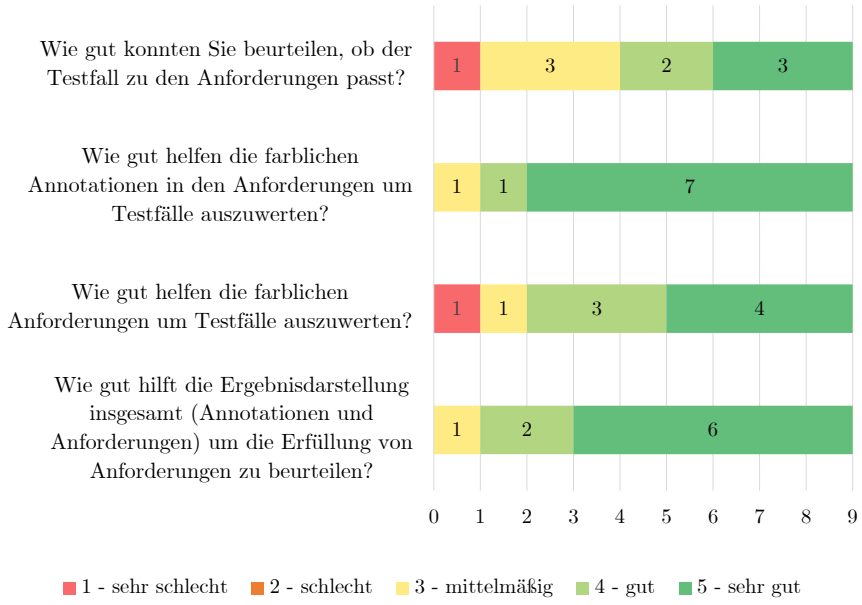



Abbildung 7.8: Anzahl der vergebenen Bewertungen je Frage des vierten Evaluationsteils

Bewertung von 4,7 erreicht. Im Gegensatz dazu erreichte Frage drei eine Durchschnittsbewertung von 4 bei einem Median von 4, einem Minimum von 1 und einem Maximum von 5.

Die vierte Frage stellt die Ergebnisdarstellung insgesamt in den Vordergrund. Hier vergaben die Probanden im Durchschnitt eine 4,6 von 5. Das Minimum und das Maximum liegen bei 3 und 5. Der Median lag bei den 9 Teilnehmern bei 5.

**Interpretation der Ergebnisse:** Die eingefärbten Anforderungen und die farbigen Darstellung der Annotationen sind Hauptmerkmale des entwickelten Testansatzes und werden durch den vierten Evaluationsteil analysiert. Die Ergebnisse adressieren die vierte Forschungsfrage. Die Auswertungen haben ergeben, dass die verwendeten Darstellungen der Testergebnisse einen sehr großen positiven Einfluss auf die Beurteilung der Testdurchläufe haben. In diesem Teil vergaben drei der neun Probanden in allen Fragen die Höchstbewertung von 5.

Speziell die Antworten der zweiten Frage (4,7) unterstreichen die Vermutung, dass Ergebnisse in natürlichsprachlichen Anforderungen einen erheblichen Mehrwert für die Testfallauswertung haben. Auf diese Frage antworteten 78 % (7 Probanden) mit *sehr gut*. Damit ist diese Frage die bestbewertete Frage der Gesamtevaluation.

Gleichzeitig, zeigen die Ergebnisse der ersten Frage den kleinsten Durchschnittswert (3,7) der Studie. Hier lag der Fokus der Fragestellung jedoch nicht auf der Testfallauswertung, sondern die Beurteilung wie gut ein Testfall zu den Anforderungen passt. Das Ergebnis macht deutlich, dass die farbigen Annotationen nicht jeden Bereich des Test

Engineering gleich gut unterstützt. Das zeigt zusätzlich die Bewertung eines Probanden, der für die Fragen eins und drei jeweils ein *sehr schlecht* vergeben hat.

#### 7.2.4.6 *Ergebnisse der Evaluation der praktischen Relevanz*

Im letzten Teil der Evaluation liegt der Fokus auf dem Gesamteindruck des Testansatzes und die praktische Relevanz im Testmanagement für zukünftige komplexe Fahrzeugfunktionen. Abbildung 7.9 zeigt die durchschnittliche Bewertung der Fragen des letzten Evaluationsteils.

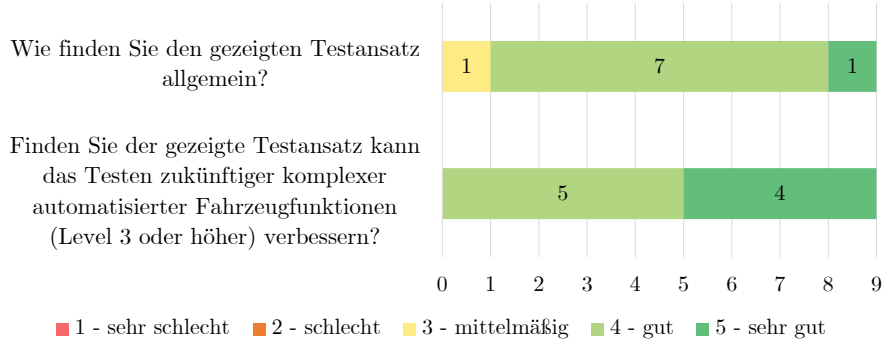



Abbildung 7.9: Anzahl der vergebenen Bewertungen je Frage des fünften Evaluationsteils

Der allgemeine Testansatz (Frage 1) wird durchschnittlich mit 4 bewertet. Die Antworten bewegen sich zwischen einem Minimum von 3 und einem Maximum von 5. Der Median liegt bei 4.

Im Hinblick auf den zukünftigen praktischen Nutzen des Testansatzes sehen die Probanden, bei einer durchschnittlichen Bewertung von 4,4, eine gute Unterstützung im Testprozess. Der Median liegt bei 4 und das Minimum und Maximum bei jeweils 4 und 5

**Interpretation der Ergebnisse:** Eine Gesamtanalyse des Testansatzes ist durch den fünften Evaluationsteil möglich. Hier stand die ganzheitliche Bewertung des Testansatzes im Vordergrund. Die erste Frage eruiert den Gesamtansatz. Die durchschnittliche Bewertung von 4 zeigt, dass der Testansatz eine gute Methode ist, die das Testen von komplexen Systemen verbessert. Lediglich ein Proband vergab die Maximalbewertung von 5, da in jedem Interview Verbesserungsvorschläge oder kritische Punkte angesprochen wurden. Hierbei standen zwei Aspekte in der Mehrzahl der Interviews im Vordergrund. Erstens, ist der Einsatz der Testmethode mit manuellem Aufwand im Annotationsprozess verbunden. Daher sind die Ergebnisse immer unter Berücksichtigung der Qualität der Annotationen zu betrachten. Zweitens, sind die Aussagen zu erfüllten Anforderungen nur im Zusammenhang mit der Vollständigkeit der Annotationen zu treffen. Die

Ergebnisse der Probanden aus dem Automotive Bereich zeigen keine signifikante Abweichung zu allen anderen Probanden.

### 7.2.5 *Validität*

Die Ergebnisse der Studie unterliegen möglichen Einschränkungen ihrer Validität. Wir identifizierten vier mögliche Einflüsse die in den folgenden Absätzen erläutert und diskutiert werden.

**Auswahl der Studienteilnehmer**: Die erste mögliche Einschränkung der Validität bezieht sich auf die Auswahl der Probanden. Die Studie wurde mit neun Probanden aus dem Software Engineering Umfeld mit Bezug zum Anforderungs- und Testmanagement durchgeführt. Da der Urheber des vorgestellten Testansatzes gleichzeitig das Interview führte, besteht die Möglichkeit, dass Probanden zurückhaltender mit niedriger Punktbewertung waren. Zudem kannte der Interviewer zwei Drittel der Probanden persönlich, was die Bewertung zusätzliche beeinflussen könnte. Um diesen Einfluss der Validität möglichst gering zu halten, fand eine Selbsteinschätzung der Erfahrung in den Bereichen Requirements- und Testengineering statt. Die Auswertungen in Abschnitt 7.2.4.1 zeigen die Expertise in den jeweiligen Bereichen die für die Beurteilung und Einschätzung des Ansatzes im Vordergrund steht.

**Auswahl der Daten**: Die zweite Einschränkung der Validität umfasst die Auswahl der Anforderungen im Interview. Die Auswahl der gezeigten Anforderungen unterlag verschiedenen Kriterien. Beispielsweise musste sichergestellt werden, dass Annotationen mit allen Elementen der Mulitlevel Markup Language prinzipiell möglich war. Außerdem wurden Anforderungen aus einer Fahrzeugfunktion gewählt, die weitverbreitet in vielen Fahrzeugen verbaut ist. Damit wurde erreicht, dass alle Probanden ihren Evaluationsfokus auf den Testansatz legten und nicht mangelndes Grundverständnis der Fahrzeugfunktion in die Bewertung eingeht. Es ist jedoch nicht auszuschließen, dass diese Kriterien und die resultierenden Anforderungen auch die Bewertung des Testansatzes beeinflussten. So ist denkbar, dass gerade wenn alle Annotationen möglich sind auch der Testansatz insgesamt positiver bewertet wird. Um diesen Aspekt der Validität einzuschränken, wurden reale Anforderungen eines Industriepartners gewählt, die angepasst wurden, um die Geheimhaltung von Softwaredetails nicht zu gefährden. Hiermit wurde gezeigt, wie sich der Testansatz unter annähernd realen Bedingungen einsetzen lässt.

**Dauer des Interviews**: Der dritte Einfluss auf die Validität ist durch den Zeitrahmen des Interviews gegeben. Um Teilnehmer aus dem industriellen Umfeld zu gewinnen, musste vorab eine Zeitrahmen definiert werden, der sich in den Alltag der Teilnehmer integrieren lässt. Das Interview wurde auf eine Stunde angesetzt. In diesem Zeitfenster musste planmäßig, wie in Kapitel 7.2.3 beschrieben, der Testansatz

erklärt werden, die Teilnehmer mussten den Testansatz selbst nutzen und anschließen Evaluationsfragen beantworten. Durch den Zeitrahmen war keine umfassende Beschreibung des gesamten Testansatzes möglich, sondern fokussierte sich auf die manuellen Aspekte, damit die Teilnehmer die gestellten Aufgaben umfassend durchführen konnten. Daraus folgte, dass die automatisierten Teile, insbesondere der Auswertungsprozess, gar nicht oder nur sehr kurz skizziert werden konnte. Dieser Aspekt könnte die Bewertung tendenziell negativ beeinflusst haben, da die Vorteile der automatisierten Schritte den Teilnehmern unbekannt blieben. Durch die offene Gesprächsführung während des Interviews wurde versucht diesen Einfluss zu minimieren, da die Teilnehmer jederzeit im Dialog mit dem Interviewer standen und alle aufkommenden Fragen sofort stellen konnten. Ziel war es, dass alle Teilnehmer den den Ansatz ohne offene Fragen bewerten konnten.

**Auswahl der Bewertungsskala**: Ein weiterer, vierter, Einfluss auf die Validität ist die Auswahl der Bewertungsskala. Für die Beantwortung der Evaluationsfragen stand eine numerische Rating-Skala von eins bis fünf zur Verfügung. Es gibt eine Vielzahl weiterer möglichen Bewertungsskalen mit unterschiedlichem Fokus der Evaluation und Einfluss auf den Teilnehmer [7]. Es kann nicht ausgeschlossen werden, dass andere Bewertungsskalen zu einem anderen Gesamtergebniss führen. Dieser Einfluss ist für eine spezifisches Interview schwer zu untersuchen. Dazu müssten weitere vergleichbare Interviews mit unterschiedlichen Skalen durchgeführt werden, um diesen Einfluss weiter zu untersuchen.

### 7.2.6 *Diskussion*

Zusammenfassend zeigt die Studie eine gute Anwendbarkeit des Testansatzes. Tabelle 7.4 zeigt die Bewertungen der Evaluationsteile im Überblick.

Tabelle 7.4: Durchschnittliche Bewertung jedes Evaluationsteils

| **Evaluationsteil** | Teil 1 | Teil 2 | Teil 3 | Teil 4 | Teil 5 |
|---|---|---|---|---|---|
| **Mittelwert** | 4,2 | 4,3 | 3,9 | 4,2 | 4,2 |

In vier der fünf Evaluationsteile wird eine durchschnittliche Bewertung von über 4,0 erreicht, was mit *gut* assoziiert wird. Teil 3 schnitt mit 3,9 ab und setzte die tabellarische Darstellung der Ergebnisse in den Vordergrund. In Zusammenhang mit dem zweiten Teil der Evaluation, welches die höchste Bewertung erhielt, verdeutlicht die Studie die Einsatzbereiche der unterschiedlichen Darstellungsformen. Die tabellarische Darstellung der Annotationen eignet sich besonders

gut für die Erstellung von Testfällen und erhöht das Verständnis des Systemverhaltens (Teil 2 der Evaluation). Im Gegensatz dazu, ist das Verhalten des Systems in Testdurchläufen durch die tabellarische Ergebnisdarstellung weniger gut ablesbar (Teil 3 der Evaluation). Teil eins und Teil vier evaluieren die unterschiedlichen Aspekte der anforderungsbasierten Darstellung der Annotationen. In dieser Darstellung sind die Annotationen gemeinsam mit der ausgewählten Anforderung sichtbar. Beide Teile wurden mit 4,2 bewertet, was etwa dem Gesamtdurchschnitt von 4,1 entspricht. Besonders für die Testauswertung sind eingefärbte Annotationen sehr gut geeignet, was einen Kernaspekte des Testansatzes und der Multilevel Markup Language darstellt.

Alle Probanden hatten am Ende der Evaluation die Möglichkeit Feedback in eigenen Worten wiederzugeben. Hier traten die Probanden mit dem Interviewer in den Austausch. In allen Interviews wurden hier positive und negative Aspekte des Ansatzes beleuchtet.

In den Dialogen mit den Probanden stehen vier Vorteile im Vordergrund.

Erstens sehen viele Probanden im gezeigten Ansatz eine systematische Herangehensweise für die Verarbeitung natürlichsprachlicher Anforderungen, was besonders in großen Requierements Engineering Abteilungen wichtig ist. Hier sind derzeit keine einheitlichen Vorgehensweisen etabliert.

Zweitens ein Teil der Probanden äußerte, dass weniger Aspekte des Systems vergessen werden. Alle Parameter, äußere Einflüsse und kleinere Sonderfälle werden gleichermaßen mit den wichtigen Kerneigenschaften des Systems dargestellt. Das ermöglicht umfassendere Tests.

Drittens wurden besonders die eingefärbten Annotationen hervorgehoben, die eine Testfallauswertung mit den Anforderungen ermöglicht. In derzeitig eingesetzten Testverfahren stehen die Einzelergebnisse eines Tests im Vordergrund. Oft ist den Testingenieur:innen hier nicht klar, ob jedoch die Anforderungsabdeckung ausreichend ist, um valide Aussagen zur korrekten Systemfunktionalität zu treffen. Der entwickelte Testansatz kann hier einen erheblichen Mehrwert leisten.

Viertens betonten die Probanden die nicht im Automotive Kontext tätig sind, dass eine Übertragbarkeit in andere Branchen ebenfalls relevant sei. Besonders die Extraktion von Informationen aus natürlichsprachlichen Anforderungen mithilfe der Annotationen ist branchenübergreifend nützlich. Hier sind Testverfahren weniger signalbasiert. Daher steht der Auswertungsprozess und der daraus resultierende Darstellungsprozess weniger im Vordergrund. Abhängig von den Logdaten, beispielsweise bei Anwendersoftware oder Cloud-Services, können diese Aspekte des Ansatzes ebenfalls relevant sein, da diese gleichermaßen zustandsbasiert sind.

Im Austausch mit dem Interviewer standen drei mögliche Nachteile des Ansatzes im Vordergrund.

Erstens wird von der Mehrheit der Teilnehmer der manuelle Aufwand und die praktische Einarbeitung von Mitarbeitern kritisch gesehen. Viele Requirements- und Testingenieur:innen sind darauf angewiesen Tools zu nutzen, die selbsterklärend und mit wenig manuellem Aufwand verbunden sind. Neue Verfahren oder Tools können nur eingesetzt werden, wenn entsprechend Zeit und Aufwand für die Nutzung von der Firmenleitung berücksichtigt wird. Eine mögliche Automatisierung und Unterstützung für den Annotationsprozess wird in Kapitel 4.2 beschrieben und zeigt sehr hohes Potenzial die Akzeptanz der Methode zu erhöhen.

Zweitens betonen manche Probanden, dass die Qualität des gesamten Testansatzes vom Annotationsprozess und der Vollständigkeit der Annotationen abhängt. Zwar liefern die Annotationen Hinweise auf die Anforderungsabdeckung, jedoch gibt es keine Sicherstellung, dass Annotationen vollständig erstellt wurden. Speziell in großen Firmen mit mehreren Mitarbeitern im Anforderungs- und Testmanagement sind Metriken, die den Annotationsprozess bewerten, hilfreich. Kapitel 8 beschreibt zukünftige Herangehensweisen für mögliche Metriken.

Drittens sehen einige Probanden den Ansatz bei wechselnden oder gar fehlenden Anforderung kritisch. Der zunehmende Anteil agiler Softwareprojekte erfordert ständig wechselnde Anforderungen und in der Entwicklung von Prototypen sind Anforderungen häufig noch nicht dokumentiert. Das Ändern von Anforderungen erfordert ebenfalls eine Anpassung der Annotationen, gegebenenfalls auch der Abbildungstabelle im Abbildungsprozess. Diese Änderungen müssen parallel zur Softwareentwicklung stattfinden. Je nach Einbettung des Testansatzes in den Entwicklungsprozess kann das mit unterschiedlichem Aufwand verbunden sein. Bisherige Testspezifikationen in agilen Projekten unterliegen jedoch den gleichen Herausforderungen und werden im Entwicklungsprozess spät finalisiert. Die Studie in Kapitel 6.2 zeigt, welches Potenzial der Testansatz neben der alleinigen Annotationsauswertung bietet. Der erhöhte Aufwand für die Anpassung von Annotationen geht mit einer Vielzahl neuen Erkenntnissen einher, die den Entwicklungsprozess letztendlich verbessert.

**Limitationen:** Die Aussagekraft der Studie und der resultierenden Ergebnisse ist durch Teilaspekte des Studienaufbaus limitiert.

Im Vordergrund der Studie seht die praktische Relevanz für heutige und zukünftige Softwareentwicklungen. Der gezeigte Ansatz hat das Ziel, heutige Entwicklungsprozesse zu verbessern. In gewinnorientierten Unternehmen geht eine Verbesserung oft mit Kostenersparnissen oder Gewinnmaximierung einher. Die Studie untersucht nicht die wirtschaftlichen Auswirkungen und setzt ebenfalls nicht den Nutzen ins Verhältnis zu möglichen (personellen) Kosten. Daher lassen sich die Ergebnisse nur im Kontext der Entwicklungsprozesse beurteilen und eruieren den Nutzen im Vergleich zu heutigen Methoden. Ob sich

Entwicklungsprozesse dadurch tatsächliche wirtschaftlich verbessern, kann nicht gezeigt werden.

Eine weitere Limitation umfasst die Aussagekraft für andere Branchen, außerhalb der Automobilbranche. Der Fokus dieser Arbeit liegt auf dem Einsatz in der Automobilbranche, ist aber nicht auf diese beschränkt. Alle Teile des Testansatzes lassen sich über die Automobilbranche hinaus einsetzen. Die Ergebnisse der Interviewstudie sind auf der einen Seite zwar allgemeingültig, auf der anderen Seite für den Automotive Kontext stärker zu gewichten. Das hat zwei Hauptgründe. Erstens, sind die gezeigten Anforderungen an eine Softwarespezifikationen einer Fahrzeugfunktionen angelehnt. Das führt dazu, dass die Studienteilnehmer ihre Antworten nur bedingt auf allgemeine Anforderungen beziehen können. Zweitens, sind circa 56 % der Teilnehmer in der Automobilbranche tätig und geben ihre Antworten mit ihrem beruflichen Kontext. Die praktische Relevanz und die Vorteile des Testansatzes lassen sich, aus den Interviewergebnissen, nur bedingt auf andere Branchen übertragen.

# 8 ZUSAMMENFASSUNG UND AUSBLICK

Dieses Kapitel fasst alle Beiträge und Ergebnisse zusammen und gibt einen Ausblick auf zukünftige Arbeiten.

## 8.1 ZUSAMMENFASSUNG DER ERGEBNISSE

Die Verknüpfung von natürlichsprachlichen Anforderungen und Systemausführungen ist eine zunehmende Herausforderung in der Entwicklung komplexer Softwaresysteme. So erhöht sich auch der Aufwand im Anforderungs- und Testmanagement. Bisherige Methoden stoßen insbesondere bei natürlichsprachlichen Anforderungen an Grenzen und können nicht vollständig automatisiert werden. Diese Arbeit zeigt die Entwicklung eines Ansatzes, der auf einer Markierungssprache basiert. Dieser extrahiert Informationen aus Anforderungen und stellt sie Systemausführungen zur Verfügung. Im Anschluss werden die Systemaufzeichnungen automatisiert ausgewertet und die Ergebnisse in den natürlichsprachlichen Anforderungen dargestellt. Das ermöglicht eine Auswertung des Systemverhaltens innerhalb des Anforderungskontexts. Zusätzlich wurde eine Methode zur Unterstützung des Markierungsprozesses entwickelt. Basierend auf den Methoden des maschinellen Lernens kann der Algorithmus für beliebige Anforderungsdokumente eingesetzt werden. Wie bereits im Einführungskapitel erläutert, leistet diese Arbeit vier große Beiträge.

- Beitrag A: Entwicklung einer Multilevel Markup Language zur Markierung von Textpassagen in natürlichsprachlichen Anforderungsdokumenten
- Beitrag B: Vorgehensmodell zur Entwicklung eines automatisierten Algorithmus für die Erkennung von Systemzuständen
- Beitrag C: Auf einer Multilevel Markup Language basierende vierteilige Analysemethode zur Auswertung von Systemverhalten
- Beitrag D: Analyse zur Nutzbarkeit von Annotationen in natürlichsprachlichen Systemspezifikationen

Im Folgenden werden die wesentlichen Teilergebnisse dieser Beiträge zusammengefasst ausgeführt.

### A | ENTWICKLUNG EINER MULTILEVEL MARKUP LANGUAGE ZUR MARKIERUNG VON TEXTPASSAGEN IN NATÜRLICHSPRACHLICHEN ANFORDERUNGSDOKUMENTEN

In diesem Beitrag steht die Entwicklung einer Multilevel Markup Language im Fokus. Diese Markierungssprache besteht aus vier Levels, die jeweils verschiedene Elemente enthalten und einen Detailgrad der Auswertungsmöglichkeit repräsentieren. Mit den Elementen des Level 1 (Scope-Level) können Textpassagen in systemzugehörig oder nicht systemzugehörig eingeteilt werden. Die Auswertungen zeigen, ob den markierten Textpassagen eine Signalabbildung zugeordnet wird. In Level 2 (Type-Level) findet eine Zuordnung zu einem der vier Typen *Value*, *State*, *Event* und *Time* statt. In Level 3 (Condition-Level) besteht die Möglichkeit aus den Markierungen des Level 2 Bedingungen zu erstellen, die auf ihre Erfüllung hin überprüft werden. Das Level 4 (Causality-Level) ermöglicht einen kausalen Zusammenhang zwischen zwei Level 3 Bedingungen. Hierbei können *Trigger*, *Pre-Conditions* und *Actions* definiert und verknüpft werden.

### B | VORGEHENSMODELL ZUR ENTWICKLUNG EINES AUTOMATISIERTEN ALGORITHMUS FÜR DIE ERKENNUNG VON SYSTEMZUSTÄNDEN

Der zweite Beitrag leistet eine mögliche Unterstützung innerhalb der Multilevel Markup Language. Die Sprache ist vorrangig eine manuelle Methode, um abhängig vom Testziel und Entwicklungsstand der Software Informationen zu extrahieren. Mithilfe maschinellen Lernens können Elemente dieser Sprache automatisiert erkannt und markiert werden. Hierbei ist die Qualität der Erkennung unter anderem vom Aufbau und der verwendeten Sprache der Anforderung abhängig. Der Algorithmus muss die spezifischen Besonderheiten der Anforderungen erlernen um sie anschließend zuverlässig zu erkennen. Hierfür werden große Datenmengen benötigt. Zwei Vorgehensweisen zur Entwicklung eines automatisierten Erkennungsansatzes basierend auf Methoden des maschinellen Lernens wurden anhand der Ergebnisse der Evaluation von 2.000 Anforderungen verglichen. Da Softwarefunktionen im Automobilbereich überwiegend zustandsbasiert arbeiten, lag der Fokus auf der automatisierten Erkennung von Systemzuständen.

### C | AUF EINER MULTILEVEL MARKUP LANGUAGE BASIERENDE VIERTEILIGE ANALYSEMETHODE ZUR AUSWERTUNG VON SYSTEMVERHALTEN

Der dritte Beitrag dieser Arbeit umfasst neben den Auswertungen der markierten Textpassagen einen vierteiligen Auswertungsansatz zur

Evaluation des Systemverhaltens, der neben der Auswertung von Teststufen, Testausführungen und Textmarkierungen auch einen Vergleich diesbezüglich ermöglicht. Die Auswertung von Systemausführungen ist ein wichtiger Bestandteil des gesamten Entwicklungsprozesses. Die vorgestellte Methode ermöglicht eine automatisierte, umfassende Auswertung von sehr großen und komplexen Testdurchläufen, beispielsweise von Simulationen mit über tausend Fahrzeugen oder realen Testfahrten.

#### D | ANALYSE ZUR NUTZBARKEIT VON ANNOTATIONEN IN NATÜRLICHSPRACHLICHEN SYSTEMSPEZIFIKATIONEN

Im vierten Beitrag stand die Nutzbarkeit und Anwendbarkeit der Annotationen im Annotations- und Darstellungsprozess im Untersuchungsfokus. Hierzu wurde eine Interviewstudie mit neun Probanden aus der Praxis durchgeführt. Die Teilnehmer wendeten die Kernelemente des Testansatzes an einem Beispiel an und evaluierten das Verfahren anschließend. Mithilfe der Antworten konnten die verschieden Aspekte des Testansatzes beurteilt und die Vor- und Nachteile kritisch analysiert werden.

## 8.2 AUSBLICK

Im Folgenden werden Anknüpfungspunkte und Ideen für weitere Forschungsarbeiten die sich aus dieser Arbeit ergeben könnten, vorgestellt.

#### ECHTZEITANALYSE VON ANNOTATIONEN

Der vorgestellte Testansatz stellt eine gute Grundlage zur Auswertung aufgezeichneten Systemverhaltens dar. Eine besonderer Herausforderung im Testmanagement ist die zeitaufwendige Planung und Durchführung von dynamischen Tests auf Gesamtsystemebene. Hierzu zählen beispielsweise Testfahrten zur Absicherung von Fahrzeugfunktionen. Aussagen über das System sind oft erst im Anschluss an den Testprozess verfügbar. Eine mögliche Ergänzung des vorgestellten Testansatzes um eine Echtzeitauswertung könnte dieses Problem adressieren. Es würde speziell bei Testfahrten bereits während der Systemausführung die Ergebnisse der Fahrt im Kontext der Anforderungen sichtbar machen und Aussagen über das Softwareverhalten ermöglichen. Wenn Systemausführungen beeinflussbar sind, können diese effizienter genutzt werden, da die Nutzer:innen sofort eine Rückmeldung erhalten welche Anforderungen noch nicht abgedeckt sind. Der Mehrwert durch die Echtzeitanalyse liegt in einer Effizienzsteigerung, führt zu Kostenersparnissen und reduziert den personellen Aufwand.

### AUSBAU DER AUTOMATISIERTEN ERKENNUNG VON ELEMENTEN DER MULTILEVEL MARKUP LANGUAGE

Die automatisierte Erkennung von Systemzuständen unterstützt das Annotieren der Anforderungen. Mithilfe der zwei Trainingsmethoden können zukünftig weitere Elemente der Multilevel Markup Language automatisiert erkannt werden. Der Trainingsprozess ist generisch für beliebige Anforderungsdokumente und Elemente der Sprache. Wir zeigen in der Studie den Einfluss des Aufwands der Nutzer:innen auf die Qualität der Erkennung. Die Erkennung mehrerer Elemente, insbesondere auf Level 1 und Level 2, optimiert den Testprozess weiter, da ohne manuellen Aufwand Annotationen zur Verfügung stehen, die entweder direkt ausgewertet werden oder in die automatisierte Erstellung von Testfällen einfließen können.

### ÜBERTRAGUNG DES TESTANSATZES AUF WEITERE DOMÄNEN

Der Testansatz verbindet natürlichsprachliche Anforderungen mit Systemausführungen. Dabei kann der Ansatz in beliebigen Domäne eingesetzt werden. In dieser Arbeit liegt der Fokus jedoch auf dem Einsatz im Automotive Kontext. Alle Studien dieser Arbeit sind mit Fahrzeugsystemspezifikationen evaluiert. Dennoch lassen die Ergebnisse den Schluss zu, dass eine Übertragung auf weitere Domänen möglich ist. Für diese Annahme gibt es drei Anhaltspunkte. Erstens enthalten die Elemente der Multilevel Markup Language keine fahrzeugspezifischen Anteile und sind für alle Bereiche des Software Engineerings nutzbar. Zweitens werden zwei Vorgehensmethoden zur automatisierten Erkennung von Systemzuständen präsentiert, die einen Modellaufbau mit beliebigen Systemspezifikationen zulassen. Die dafür notwendigen Spezifikationen sind nicht auf Fahrzeugspezifikationen beschränkt. Drittens nahmen an der Interviewstudie zur Evaluierung des Darstellungsprozesses Teilnehmer aus unterschiedlichen Branchen teil, wobei die Beurteilung der Teilnehmer des Automobilsektors und anderer Branchen analog ausfiel. Der Ansatz kann so vermutlich auch Relevanz in anderen Branchen erzielen.

# LITERATURVERZEICHNIS


[1] Vincenzo Ambriola und Vincenzo Gervasi. „Processing natural language requirements“. In: *Proceedings 12th IEEE International Conference Automated Software Engineering*. 1997, S. 36–45. DOI: 10.1109/ASE.1997.632822.

[2] Andrea Arcuri. „RESTful API Automated Test Case Generation with EvoMaster“. In: *ACM Trans. Softw. Eng. Methodol.* 28.1 (2019). DOI: 10.1145/3293455.

[3] Bruno M. Arnaut, Denise B. Ferrari und Marcelo L. de Oliveira e Souza. „A requirements engineering and management process in concept phase of complex systems“. In: *2016 IEEE International Symposium on Systems Engineering (ISSE)*. 2016, S. 1–6. DOI: 10.1109/SysEng.2016.7753130.

[4] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand und Frank Zimmer. „Automated Extraction and Clustering of Requirements Glossary Terms“. In: *IEEE Transactions on Software Engineering* 43.10 (2017), S. 918–945. DOI: 10.1109/TSE.2016.2635134.

[5] Johannes Bach, Stefan Otten und Eric Sax. „Model based scenario specification for development and test of automated driving functions“. In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. 2016, S. 1149–1155. DOI: 10.1109/IVS.2016.7535534.

[6] Jerry Banks, John S. Carson, Barry L. Nelson und David M. Nicol. *Discrete-Event System Simulation*. Prentice Hall, 2000.

[7] Anna E. Bargagliotti und Lingfang Li. „Decision making using rating systems: When scale meets binary“. In: *Decision Sciences* 44.6 (2013), S. 1121–1137.

[8] Zeinab A. Barmi, Amir H. Ebrahimi und Robert Feldt. „Alignment of Requirements Specification and Testing: A Systematic Mapping Study“. In: *IEEE International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2011. DOI: 10.1109/ICSTW.2011.58.

[9] Fabian Beck, Hafiz A. Siddiqui, Alexandre Bergel und Daniel Weiskopf. „Method Execution Reports: Generating Text and Visualization to Describe Program Behavior“. In: *2017 IEEE Working Conference on Software Visualization (VISSOFT)*. 2017, S. 1–10. DOI: 10.1109/VISSOFT.2017.11.

[10] Daniel M. Berry. „Ambiguity in Natural Language Requirements Documents“. In: *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*. Hrsg. von Barbara Paech und Craig Martell. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, S. 1–7. ISBN: 978-3-540-89778-1.

[11] Daniel M. Berry, Ricardo Gacitua, Pete Sawyer und Sri F. Tjong. „The Case for Dumb Requirements Engineering Tools“. In: *Requirements Engineering: Foundation for Software Quality*. Hrsg. von Björn Regnell und Daniela Damian. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, S. 211–217. ISBN: 978-3-642-28714-5.

[12] Daniel M. Berry, Erik Kamsties und Michael M. Krieger. *From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity - A Handbook Version 1.0*. 2000.

[13] Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek und Robert Feldt. „Challenges and practices in aligning requirements with verification and validation: a case study of six companies“. In: *Empirical Software Engineering* 19.6 (2014), S. 1809–1855. DOI: 10.1007/s10664-013-9263-y.

[14] Hans Brandt-Pook und Rainer Kollmeier. *Softwareentwicklung kompakt und verständlich: Wie Softwaresysteme entstehen*. 2. Auflage. Wiesbaden: Springer Vieweg, 2015. ISBN: 3658108754. DOI: 10.1007/978-3-658-10876-2.

[15] Agustin Casamayor, Daniela Godoy und Marcelo Campo. „Identification of non-functional requirements in textual specifications: A semi-supervised learning approach“. In: *Information and Software Technology* 52.4 (2010), S. 436–445. ISSN: 0950-5849.

[16] Jason P.C. Chiu und Eric Nichols. „Named Entity Recognition with Bidirectional LSTM-CNNs“. In: *Transactions of the Association for Computational Linguistics* 4 (2016), S. 357–370. URL: https://transacl.org/ojs/index.php/tacl/article/view/792.

[17] Emilia Cioroaica, Florian Pudlitz, Ilias Gerostathopoulos und Thomas Kuhn. „Simulation methods and tools for collaborative embedded systems: with focus on the automotive smart ecosystems“. In: *SICS Software-Intensive Cyber-Physical Systems* 34.4 (2019), S. 213–223. ISSN: 2524-8510. DOI: 10.1007/s00450-019-00426-5.

[18] Jane Cleland-Huang, Carl K. Chang und Mark Christensen. „Event-based traceability for managing evolutionary change“. In: *IEEE Transactions on Software Engineering* 29.9 (2003), S. 796–810. DOI: 10.1109/TSE.2003.1232285.

[19] Lara Codeca, Raphael. Frank und Thomas Engel. „Luxembourg SUMO Traffic (LuST) Scenario: 24 hours of mobility for vehicular networking research“. In: *2015 IEEE Vehicular Networking Conference (VNC)*. 2015, S. 1–8. DOI: 10.1109/VNC.2015.7385539.

[20] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu und Pavel Kuksa. „Natural Language Processing (almost) from Scratch“. In: (2017).

[21] DIN. *Bahnanwendungen – Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme: DIN*. Berlin, Germany, 2012.

[22] Alex Dekhtyar und Vivian Fong. „RE data challenge: Requirements identification with Word2Vec and TensorFlow“. In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 2017, S. 484–489.

[23] Gouri Deshpande, Quim Motger, Cristina Palomares, Ikagarjot Kamra, Katarzyna Biesialska, Xavier Franch, Guenther Ruhe und Jason Ho. „Requirements Dependency Extraction by Integrating Active Learning with Ontology-Based Retrieval“. In: *2020 IEEE 28th International Requirements Engineering Conference (RE)*. 2020, S. 78–89. DOI: 10.1109/RE48521.2020.00020.

[24] Anurag Dwarakanath, Roshni R. Ramnani und Shubhashis Sengupta. „Automatic extraction of glossary terms from natural language requirements“. In: *2013 21st IEEE International Requirements Engineering Conference (RE)*. 2013, S. 314–319. DOI: 10.1109/RE.2013.6636736.

[25] Jonas Eckhardt, Andreas Vogelsang, Henning Femmer und Philipp Mager. „Challenging Incompleteness of Performance Requirements by Sentence Patterns“. In: *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 2016, S. 46–55. DOI: 10.1109/RE.2016.24.

[26] Agung Fatwanto. „Translating software requirements from natural language to formal specification“. In: *2012 IEEE International Conference on Computational Intelligence and Cybernetics (CyberneticsCom)*. 2012, S. 148–152. DOI: 10.1109/CyberneticsCom.2012.6381636.

[27] Henning Femmer und Andreas Vogelsang. „Requirements Quality Is Quality in Use“. In: *IEEE Software* 36.3 (2019), S. 83–91. DOI: 10.1109/MS.2018.110161823.

[28] Alessio Ferrari, Giuseppe Lipari, Stefania Gnesi und Giorgio O. Spagnolo. „Pragmatic ambiguity detection in natural language requirements“. In: *2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. 2014, S. 1–8. DOI: 10.1109/AIRE.2014.6894849.

[29] Alessio Ferrari, Felice dell'Orletta, Andrea Esuli, Vincenzo Gervasi und Stefania Gnesi. „Natural language requirements processing: A 4D vision". In: *IEEE Annals of the History of Computing* 34.06 (2017), S. 28–35.

[30] Alessio Ferrari, Felice dell'Orletta, Giorgio Oronzo Spagnolo und Stefania Gnesi. „Measuring and Improving the Completeness of Natural Language Requirements". In: *Requirements Engineering: Foundation for Software Quality*. Hrsg. von Camille Salinesi und Inge van de Weerd. Cham: Springer International Publishing, 2014, S. 23–38. ISBN: 978-3-319-05843-6.

[31] Peter Fröhlich und Johannes Link. „Automated Test Case Generation from Dynamic Models". In: *ECOOP 2000 — Object-Oriented Programming*. Hrsg. von Elisa Bertino. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, S. 472–491. ISBN: 978-3-540-45102-0.

[32] Clément Galko, Romain Rossi und Xavier Savatier. „Vehicle-Hardware-In-The-Loop system for ADAS prototyping and validation". In: *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*. 2014, S. 329–334. DOI: 10.1109/SAMOS.2014.6893229.

[33] Juan M. Carrillo de Gea, Joaquín Nicolás, José L. Fernández Alemán, Ambrosio Toval, Christof Ebert und Aurora Vizcaíno. „Requirements engineering tools: Capabilities, survey and assessment". In: *Information and Software Technology* 54.10 (2012), S. 1142–1157. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2012.04.005.

[34] Martin Glinz. „Wörterbuch der Requirements Engineering Terminologie." In: *Standard-Glossar für das Studium und die Prüfung zum Certified Professional for Requirements Engineering (CPRE).* (2020).

[35] Orlena C. Z. Gotel und Anthony C. W. Finkelstein. „An analysis of the requirements traceability problem". In: *Proceedings of IEEE International Conference on Requirements Engineering*. 1994, S. 94–101. DOI: 10.1109/ICRE.1994.292398.

[36] Alex Graves, Abdel-rahman Mohamed und Geoffrey Hinton. „Speech Recognition with Deep Recurrent Neural Networks". In: *IEEE International Conference on Acoustics, Speech and Signal Processing* (2013).

[37] Andreas Gregoriades, Maria Pampaka und Alistair Sutcliffe. „Simulation-based requirements discovery for smart driver assistive technologies". In: *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. 2014, S. 317–318. DOI: 10.1109/RE.2014.6912275.

[38] Carl A. Gunter, Elsa L. Gunter, Michael Jackson und Pamela Zave. „A reference model for requirements and specifications“. In: *IEEE Software* 17.3 (2000), S. 37–43. DOI: 10.1109/52.896248.

[39] Sonal Gupta und Christopher D. Manning. „Improved Pattern Learning for Bootstrapped Entity Extraction“. In: *Computational Natural Language Learning (CoNLL)*. 2014.

[40] Marti A. Hearst. „Automatic Acquisition of Hyponyms from Large Text Corpora“. In: *Proceedings of the 14th Conference on Computational Linguistics - Volume 2*. COLING '92. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, S. 539–545. DOI: 10.3115/992133.992154.

[41] Julia Hirschberg und Christopher D. Manning. „Advances in natural language processing“. In: *Science* 349.6245 (2015), S. 261–266. ISSN: 0036-8075. DOI: 10.1126/science.aaa8685.

[42] Sofija Hotomski, Eya B. Charrada und Martin Glinz. „Keeping Evolving Requirements and Acceptance Tests Aligned with Automatically Generated Guidance“. In: *Requirements Engineering: Foundation for Software Quality*. Hrsg. von Erik Kamsties, Jennifer Horkoff und Fabiano Dalpiaz. Cham: Springer International Publishing, 2018, S. 247–264. ISBN: 978-3-319-77243-1.

[43] Sofija Hotomski, Eya B. Charrada und Martin Glinz. „An Exploratory Study on Handling Requirements and Acceptance Test Documentation in Industry“. In: *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 2016, S. 116–125. DOI: 10.1109/RE.2016.37.

[44] Sofija Hotomski und Martin Glinz. „A Qualitative Study on using GuideGen to Keep Requirements and Acceptance Tests Aligned“. In: *2018 IEEE 26th International Requirements Engineering Conference (RE)*. 2018, S. 29–39. DOI: 10.1109/RE.2018.00-54.

[45] Sofija Hotomski und Martin Glinz. „GuideGen: An approach for keeping requirements and acceptance tests aligned via automatically generated guidance“. In: *Information and Software Technology* 110 (2019), S. 17–38. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2019.01.011.

[46] WuLing Huang, Kunfeng Wang, Yisheng Lv und FengHua Zhu. „Autonomous vehicles testing methods review“. In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, S. 163–168. DOI: 10.1109/ITSC.2016.7795548.

[47] Hubert F. Hofmann und Franz Lehner. „Requirements engineering as a success factor in software projects“. In: *IEEE Software* 18.4 (2001), S. 58–66. DOI: 10.1109/MS.2001.936219.

[48] IBM Corporation. *Rational DOORS: Letzter Zugriff: 17.02.2021*. URL: https://www.ibm.com/de-de/products/requirements-management.

[49] „IEEE Recommended Practice for Software Requirements Specifications“. In: *IEEE Std 830-1998* (1998), S. 1–40. DOI: 10.1109/IEEESTD.1998.88286.

[50] *ISO/DIS 26262 Road vehicles - Functional safety: ISO 26262*. Geneva, Switzerland, 2011.

[51] International Electrotechnical Commission. *ISO / IEC TR 24766: 2009, Information technology - Systems and software engineering - Guide for requirements engineering tool capabilites*. 2009.

[52] Andreas Jedlitschka und Dietmar Pfahl. „Reporting guidelines for controlled experiments in software engineering“. In: *2005 International Symposium on Empirical Software Engineering, 2005*. 2005, 10 p–. DOI: 10.1109/ISESE.2005.1541818.

[53] Mohamad Kassab, Colin Neill und Phillip Laplante. „State of practice in requirements engineering: Contemporary data“. In: *Innovations in Systems and Software Engineering* 10.4 (2014), S. 235–241. ISSN: 1614-5054. DOI: 10.1007/s11334-014-0232-4.

[54] Leonid Kof. „Requirements Analysis: Concept Extraction and Translation of Textual Specifications to Executable Models“. In: *Natural Language Processing and Information Systems*. Hrsg. von Helmut Horacek, Elisabeth Métais, Rafael Muñoz und Magdalena Wolska. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 79–90. ISBN: 978-3-642-12550-8.

[55] Bogdan Korel. „Dynamic method for software test data generation“. In: *Software Testing, Verification and Reliability* 2.4 (1992), S. 203–213.

[56] Sven J. Körner, Mathias Landhäußer und Walter F. Tichy. „Transferring research into the real world: How to improve RE with AI in the automotive industry“. In: *2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. 2014, S. 13–18. DOI: 10.1109/AIRE.2014.6894851.

[57] Zornitsa Kozareva. „Bootstrapping Named Entity Recognition with Automatically Generated Gazetteer Lists“. In: *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*. EACL '06. Stroudsburg, PA, USA: Association for Computational Linguistics, 2006, S. 15–21.

[58] Giuseppe Lami, Stefania Gnesi, Fabrizio Fabbrini, Mario Fusani und Gianluca Trentanni. „An automatic tool for the analysis of natural language requirements“. In: *Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Setiembre* (2004).

[59] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami und Chris Dyer. *Neural Architectures for Named Entity Recognition*. 2016.

[60] Yann LeCun, Yoshua Bengio und Geoffrey Hinton. „Deep learning“. In: *Nature* 521.7553 (2015), S. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539.

[61] Elizabeth D. Liddy. „Natural Language Processing“. In: *Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc.* (2001).

[62] Pablo A. Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner und Evamarie Wiessner. „Microscopic Traffic Simulation using SUMO“. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, S. 2575–2582. DOI: 10.1109/ITSC.2018.8569938.

[63] Garm Lucassen, Marcel Robeer, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf und Sjaak Brinkkemper. „Extracting conceptual models from user stories with Visual Narrator“. In: *Requirements Engineering* (2017), 22(3):339–358. DOI: 10.1007/s00766-017-0270-1.

[64] Alexander Maedche, Günter Neumann und Steffen Staab. „Bootstrapping an ontology-based information extraction system“. In: *Intelligent exploration of the web* (2003), S. 345–359.

[65] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard und David McClosky. „The Stanford CoreNLP natural language processing toolkit“. In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 2014, S. 55–60.

[66] Alistair Mavin, Philip Wilkinson, Adrian Harwood und Mark Novak. „Easy approach to requirements syntax (EARS)“. In: *2009 17th IEEE International Requirements Engineering Conference*. 2009, S. 317–322.

[67] Luisa Mich, Mariangela Franch und Pierluigi Novi Inverardi. „Market research for requirements analysis using linguistic tools“. In: *Requirements Engineering* 9.1 (2004), S. 40–56. DOI: 10.1007/s00766-003-0179-8.

[68] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado und Jeff Dean. „Distributed representations of words and phrases and their compositionality“. In: *Advances in neural information processing systems*. 2013, S. 3111–3119.

[69] Pascal Minnerup, Tobias Kessler und Alois Knoll. „Collecting Simulation Scenarios by Analyzing Physical Test Drives“. In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. 2015, S. 2915–2920. DOI: 10.1109/ITSC.2015.467.

[70] Mirko Nentwig und Marc Stamminger. „A method for the reproduction of vehicle test drives for the simulation based evaluation of image processing algorithms“. In: *13th International IEEE Conference on Intelligent Transportation Systems*. 2010, S. 1307–1312. DOI: 10.1109/ITSC.2010.5625005.

[71] Brian Nielsen und Arne Skou. „Automated Test Generation from Timed Automata“. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Hrsg. von Tiziana Margaria und Wang Yi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, S. 343–357. ISBN: 978-3-540-45319-2.

[72] Klaus Pohl. *Requirements engineering: Fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

[73] Robert Protzmann, Björn Schünemann und Ilja Radusch. „Simulation of convergent networks for intelligent transport systems with vsimrti“. In: *Networking Simulation for Intelligent Transportation Systems: High Mobile Wireless Nodes* (2017), S. 1–28.

[74] Florian Pudlitz, Florian Brokhausen und Andreas Vogelsang. „Extraction of System States from Natural Language Requirements“. In: *27th IEEE International Requirements Engineering Conference (RE)*. 2019. DOI: 10.14279/depositonce-8717.2.

[75] Florian Pudlitz, Florian Brokhausen und Andreas Vogelsang. „What Am I Testing and Where? Comparing Testing Procedures based on Lightweight Requirements Annotations“. In: *Empirical Software Engineering* (2020). DOI: 10.1007/s10664-020-09815-w.

[76] Florian Pudlitz, Andreas Vogelsang und Florian Brokhausen. „A Lightweight Multilevel Markup Language for Connecting Software Requirements and Simulations“. In: *Requirements Engineering: Foundation for Software Quality*. Hrsg. von Eric Knauss und Michael Goedicke. Cham: Springer International Publishing, 2019, S. 151–166. ISBN: 978-3-030-15538-4. DOI: 10.14279/depositonce-8300.

[77] RTCA. *Software Considerations in Airborne Systems and Equipment Certification: DO-178.C*. 2012.

[78] Balasubramaniam Ramesh und Matthias Jarke. „Toward reference models for requirements traceability“. In: *IEEE Transactions on Software Engineering* 27.1 (2001), S. 58–93. DOI: 10.1109/32.895989.

[79] Filippo Ricca, Marco Torchiano, Massimiliano Di Penta, Mariano Ceccato und Paolo Tonella. „Using acceptance tests as a support for clarifying requirements: A series of experiments“.

In: *Information and Software Technology* 51.2 (2009), S. 270–283. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2008.01.007.

[80] Ellen Riloff. „Automatically Generating Extraction Patterns from Untagged Text“. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*. AAAI'96. AAAI Press, 1996, S. 1044–1049. ISBN: 0-262-51091-X. URL: http://dl.acm.org/citation.cfm?id=1864519.1864542.

[81] Ellen Riloff, Janyce Wiebe und Theresa Wilson. „Learning Subjective Nouns Using Extraction Pattern Bootstrapping“. In: *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*. CONLL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, S. 25–32. DOI: 10.3115/1119176.1119180.

[82] Seb Rose, Matt Wynne und Aslak Hellesoy. *The cucumber for Java book: Behaviour-driven development for testers and developers*. Pragmatic Bookshelf, 2015.

[83] Valdivino Alexandre de Santiago Júnior und Nandamudi Lankalapalli Vijaykumar. „Generating model-based test cases from natural language requirements for space application software“. In: *Software Quality Journal* 20.1 (2012), S. 77–143. ISSN: 1573-1367. DOI: 10.1007/s11219-011-9155-6.

[84] Yotaro Seki, Shinpei Hayashi und Motoshi Saeki. „Detecting Bad Smells in Use Case Descriptions“. In: *2019 IEEE 27th International Requirements Engineering Conference (RE)*. 2019, S. 98–108. DOI: 10.1109/RE.2019.00021.

[85] Atif Shah, Mohamed A. Alasow, Faisal Sajjad und Jawad J. A. Baig. „An evaluation of software requirements tools“. In: *2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS)*. 2017, S. 278–283. DOI: 10.1109/INTELCIS.2017.8260075.

[86] Forrest Shull, Hrsg. *Guide to advanced empirical software engineering*. London: Springer, 2008. ISBN: 978-1-84800-043-8. DOI: 10.1007/978-1-84800-044-5.

[87] Ernst Sikora, Bastian Tenbergen und Klaus Pohl. „Industry needs and research directions in requirements engineering for embedded systems“. In: *Requirements Engineering* 17.1 (2012), S. 57–78.

[88] Jorge Teixeira, Luís Sarmento und Eugénio Oliveira. „A Bootstrapping Approach for Training a NER with Conditional Random Fields“. In: *Progress in Artificial Intelligence*. Hrsg. von Luis Antunes und H. Sofia Pinto. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 664–678. ISBN: 978-3-642-24769-9.

[89] Michael Thelen und Ellen Riloff. „A Bootstrapping Method for Learning Semantic Lexicons Using Extraction Pattern Contexts“. In: *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*. EMNLP '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, S. 214–221. DOI: 10.3115/1118693.1118721.

[90] Michael Unterkalmsteiner, Tony Gorschek, Robert Feldt und Eriks Klotins. „Assessing requirements engineering and software test alignment—Five case studies“. In: *Journal of Systems and Software* 109 (2015), S. 62–77. ISSN: 0164-1212. DOI: 10.1016/j.jss.2015.07.018.

[91] Eero J. Uusitalo, Marko Komssi, Marjo Kauppinen und Alan M. Davis. „Linking Requirements and Testing in Practice“. In: *16th IEEE International Requirements 2008*, S. 265–270. DOI: 10.1109/RE.2008.30.

[92] Verein Deutscher Ingenieure. „2206: Entwicklungsmethodik für mechatronische Systeme“. In: *Verein Deutscher Ingenieure* 2 (2004).

[93] Andreas Vlachos und Caroline Gasperin. „Bootstrapping and Evaluating Named Entity Recognition in the Biomedical Domain“. In: *Proceedings of the HLT-NAACL BioNLP Workshop on Linking Natural Language and Biology*. LNLBioNLP '06. Stroudsburg, PA, USA: Association for Computational Linguistics, 2006, S. 138–145.

[94] Andreas Vogelsang, Henning Femmer und Christian Winkler. „Systematic Elicitation of Mode Models for Multifunctional Systems“. In: *23rd IEEE International Requirements Engineering Conference (RE)*. 2015. DOI: 10.1109/RE.2015.7320447.

[95] Stefan Wagner, Daniel Méndez Fernández, Michael Felderer, Antonio Vetrò, Marcos Kalinowski, Roel Wieringa, Dietmar Pfahl, Tayana Conte, Marie-Therese Christiansson, Desmond Greer, Casper Lassenius, Tomi Männistö, Maleknaz Nayebi, Markku Oivo, Birgit Penzenstadler, Rafael Prikladnicki, Guenther Ruhe, André Schekelmann, Sagar Sen, Rodrigo Spínola, Ahmed Tuzcu, Jose Luis De La Vara und Dietmar Winkler. „Status Quo in Requirements Engineering: A Theory and a Global Family of Surveys“. In: *ACM Trans. Softw. Eng. Methodol.* 28.2 (2019). DOI: 10.1145/3306607.

[96] Benedikt Walter, Jan Martin, Jonathan Schmidt, Hanna Dettki und Stephan Rudolph. „Executable State Machines Derived from Structured Textual Requirements - Connecting Requirements and Formal System Design“. In: *7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. 2019, S. 195–202. DOI: 10.5220/0007236601950202.

[97] Michael W. Whalen, Ajitha Rajan, Mats P.E. Heimdahl und Steven P. Miller. „Coverage Metrics for Requirements-based Testing“. In: *International Symposium on Software Testing and Analysis (ISTA)*. 2006, S. 25–36. DOI: 10.1145/1146238.1146242.

[98] Jonas Winkler und Andreas Vogelsang. „Automatic classification of requirements based on convolutional neural networks“. In: *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. 2016, S. 39–45.

[99] Hermann Winner, Bernd Danner und Joachim Steinle. „Adaptive Cruise Control“. In: *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan Hakuli und Gabriele Wolf. Wiesbaden: Vieweg+Teubner, 2009, S. 478–521. ISBN: 978-3-8348-9977-4.

[100] Dan Wu, Wee S. Lee, Nan Ye und Hai L. Chieu. „Domain Adaptive Bootstrapping for Named Entity Recognition“. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3*. EMNLP '09. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, S. 1523–1532. ISBN: 978-1-932432-63-3.

[101] Dustin Wüest, Norbert Seyff und Martin Glinz. „FlexiSketch: A lightweight sketching and metamodeling approach for end-users“. In: *Software & Systems Modeling* 18.2 (2019), S. 1513–1541. DOI: 10.1007/s10270-017-0623-8.

[102] Hui Yang, Anne de Roeck, Vincenzo Gervasi, Alistair Willis und Bashar Nuseibeh. „Speculative requirements: Automatic detection of uncertainty in natural language requirements“. In: *2012 20th IEEE International Requirements Engineering Conference (RE)*. 2012, S. 11–20. DOI: 10.1109/RE.2012.6345795.

[103] Hui Yang, Alistair Willis, Anne de Roeck und Bashar Nuseibeh. „Automatic Detection of Nocuous Coordination Ambiguities in Natural Language Requirements“. In: *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. ASE '10. New York, NY, USA: Association for Computing Machinery, 2010, S. 53–62. ISBN: 9781450301169. DOI: 10.1145/1858996.1859007.

[104] Qian Yang, J. Jenny Li und David M. Weiss. „A Survey of Coverage-Based Testing Tools“. In: *The Computer Journal* 52.5 (2009), S. 589–597. DOI: 10.1093/comjnl/bxm021.

[105] Tom Young, Devamanyu Hazarika, Soujanya Poria und Erik Cambria. „Recent Trends in Deep Learning Based Natural Language Processing“. In: *IEEE Computational Intelligence Magazine* 13.3 (2018), S. 55–75. DOI: 10.1109/MCI.2018.2840738.

[106] Tao Yue, Shaukat Ali und Lionel Briand. „Automated Transition from Use Cases to UML State Machines to Support State-Based Testing“. In: *Modelling Foundations and Applications*. Hrsg. von Robert B. France, Jochen M. Kuester, Behzad Bordbar und Richard F. Paige. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 115–131.

[107] Tao Yue, Lionel C. Briand und Yvan Labiche. „A systematic review of transformation approaches between user requirements and analysis models“. In: *Requirements Engineering* 16.2 (2011), S. 75–99.

[108] Zahra Shakeri, Oliver Karras, Parisa Ghazi, Martin Glinz, Guenther Ruhe und Kurt Schneider. „What Works Better? A Study of Classifying Requirements“. In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 2017, S. 496–501. DOI: 10.1109/RE.2017.36.

[109] Pamela Zave und Michael Jackson. „Four Dark Corners of Requirements Engineering“. In: *ACM Trans. Softw. Eng. Methodol.* 6.1 (1997), S. 1–30. DOI: 10.1145/237432.237434.

[110] Sai Zhang, David Saff, Yingyi Bu und Michael D. Ernst. „Combined Static and Dynamic Automated Test Generation“. In: *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. ISSTA '11. New York, NY, USA: Association for Computing Machinery, 2011, S. 353–363. ISBN: 9781450305624. DOI: 10.1145/2001420.2001463.

[111] Hong Zhu, Patrick A. V. Hall und John H. R. May. „Software Unit Test Coverage and Adequacy“. In: *ACM Comput. Surv.* 29.4 (1997), S. 366–427. DOI: 10.1145/267580.267590.

# ABBILDUNGSVERZEICHNIS

# TABELLENVERZEICHNIS

# ABKÜRZUNGSVERZEICHNIS

| | |
|---|---|
| ACC | Adaptive Cruise Control |
| ASIL | Automotive Safety Integrity Level |
| BiLSTM | Bidirectional Long short-term memory |
| CNN | Convolutional Neural Network |
| CoNLL2003 | Conference on Natural Language Learning 2003 |
| CSV | Comma-separated values |
| EARS | Easy Approach to Requirements Syntax |
| HiL | Hardware-in-the-Loop |
| LuST | Luxembourg SUMO Traffic |
| NER | Named-Entity Recognition |
| NLP | Natural language processing |
| SiL | Software-in-the-Loop |
| SuD | System-under-Development |
| SUMO | Simulated Urban Mobility |
| SuT | System-under-Test |
| TA | Totwinkel-Assistent |
| VSimRTI | Vehicle-2-X Simulation Runtime Infrastructure |
| XML | Extensible Markup Language |

## EIDESSTATTLICHE VERSICHERUNG

Hiermit versichere ich an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel und Literatur angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind als solche kenntlich gemacht. Ich versichere an Eides statt, dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie - abgesehen von oben angegebenen Teilpublikationen und eingebundenen Artikeln und Manuskripten - noch nicht veröffentlicht worden ist sowie, dass ich eine Veröffentlichung der Dissertation vor Abschluss der Promotion nicht ohne Genehmigung des Promotionsausschusses vornehmen werde. Die Bestimmungen dieser Ordnung sind mir bekannt. Darüber hinaus erkläre ich hiermit, dass ich die Ordnung zur Sicherung guter wissenschaftlicher Praxis und zum Umgang mit wissenschaftlichem Fehlverhalten der Universität zu Köln gelesen und sie bei der Durchführung der Dissertation zugrundeliegenden Arbeiten und der schriftlich verfassten Dissertation beachtet habe und verpflichte mich hiermit, die dort genannten Vorgaben bei allen wissenschaftlichen Tätigkeiten zu beachten und umzusetzen. Ich versichere, dass die eingereichte elektronische Fassung der eingereichten Druckfassung vollständig entspricht.

*Berlin, Juli 2021*

Florian Pudlitz