Systematik zur integrativen Anforderungsanalyse und Testspezifikation für die Entwicklung von Systemen im Automobilbereich

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
der Fakultät für Elektrotechnik, Informatik und Mathematik
der Universität Paderborn

genehmigte
DISSERTATION

von
Carsten Wiecher
aus Wettringen

Tag des Kolloquiums: 08.03.2024
Referent: Prof. Dr.-Ing. Roman Dumitrescu
Korreferent: Prof. Dr.-Ing. Carsten Wolff

Vorwort

Diese Dissertation entstand während meiner Tätigkeit bei der Kostal Automobilelektrik GmbH & Co. KG und dem Institut für die Digitalisierung von Arbeits- und Lebenswelten der Fachhochschule Dortmund. Der Rahmen meiner Forschungsaktivitäten war das vom BMBF geförderte Verbundprojekt MoSyS (Menschorientierte Gestaltung komplexer System of Systems).

Mein besonderer Dank gilt Herrn Prof. Dr.-Ing. Roman Dumitrescu und Herrn Prof. Dr.-Ing. Carsten Wolff, die die Erstellung dieser Arbeit in einem herausragenden Forschungsumfeld ermöglicht haben. Die Möglichkeit, die Zwischenergebnisse in der Doktorandenrunde am Fraunhofer IEM und im Umfeld der FH Dortmund vorzustellen, haben diese Dissertation maßgeblich geprägt.

Herrn Prof. Dr. rer. nat. Joel Greenyer danke ich für die sehr gute und lehrreiche Zusammenarbeit, die zu mehreren Veröffentlichungen geführt hat. Die vielen Diskussionen, besonders in der Anfangsphase, haben den Grundstein für diese Dissertation gelegt. Ebenso danke ich Herrn Dr. rer. nat. Jannik Fischbach für die gute Zusammenarbeit und den interessanten und gewinnbringenden Austausch zu unseren Forschungsaktivitäten.

Meinen Kollegen und Vorgesetzten bei der Kostal Automobilelektrik GmbH & Co. KG danke ich für die kontinuierliche Unterstützung, das Feedback zu den Zwischenergebnissen und die Möglichkeit, innerhalb des Verbundprojekts anwendungsorientiert zu forschen. Besonders hervorheben möchte ich Andreas Tombült, Frank Piepenburg, Dr.-Ing. Christian Fölting und Alexander Bahr, die das Projekt von Beginn an unterstützt haben. Vielen Dank für das entgegengebrachte Vertrauen.

Ich danke allen Projektpartnern des MoSyS Projekts für den disziplinübergreifenden Austausch. Hervorheben möchte ich Matthias Günther, Constantin Mandel, Matthias Greinert, Bruno Albert, Amelie Tihlarik und Dr. phil. Silke Röbenack.

Meinen Eltern und Schwiegereltern danke ich für die Unterstützung. Der größte Dank gilt meiner Frau Ulrike für die Kraft in einer herausfordernden Zeit.

Coesfeld, im März 2023 Carsten Wiecher

Liste der veröffentlichten Teilergebnisse

[FFV+23] FISCHBACH, J.; FRATTINI, J.; VOGELSANG, A.; MÉNDEZ, D.; UNTERKALMSTEINER, M.; WEHRLE, A.; HENAO, P. R.; YOUSEFI, P.; JURICIC, T.; RADDUENZ, J.; WIECHER, C.: Automatic Creation of Acceptance Tests by Extracting Conditionals from Requirements: NLP Approach and Case Study. In: Journal of Systems and Software, 2023

[WFG+21] WIECHER, C.; FISCHBACH, J.; GREENYER, J.; VOGELSANG, A.; WOLFF, C.; DUMITRESCU, R.: Integrated and Iterative Requirements Analysis and Test Specification: A Case Study at Kostal. In: 24th International Conference on Model Driven Engineering Languages and Systems, MODELS, Fukuoka, Japan, October 10-15, 2021, IEEE, 2021, S. 112–122

[WG21] WIECHER, C.; GREENYER, J.: BeSoS: A Tool for Behavior-driven and Scenario-based Requirements Modeling for Systems of Systems. In: REFSQ 2021 Workshops, Essen, Germany, April 12, 20, CEUR-WS.org, 2021

[WGK19] WIECHER, C.; GREENYER, J.; KORTE, J.: Test-Driven Scenario Specification of Automotive Software Components. In: 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion, Munich, Germany, September 15-20, 2019, IEEE, 2019, S. 12–17

[WGW+21] WIECHER, C.; GREENYER, J.; WOLFF, C.; ANACKER, H.; DUMITRESCU, R.: Iterative and Scenario-Based Requirements Specification in a System of Systems Context. In: Requirements Engineering: Foundation for Software Quality - 27th International Working Conference, REFSQ, Essen, Germany, April 12-15, 2021, Springer, 2021, S. 165–181

[WJK+20] WIECHER, C.; JAPS, S.; KAISER, L.; GREENYER, J.; DUMITRESCU, R.; WOLFF, C.: Scenarios in the loop: integrated requirements analysis and automotive system validation. In: MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Canada, 18-23 October, 2020, Companion Proceedings, ACM, 2020, S. 35:1–35:10

[WMG+24] WIECHER, C.; MANDEL, C.; GÜNTHER, M.; FISCHBACH, J.; GREENYER, J.; GREINERT, M.; WOLFF, C.; DUMITRESCU, R.; MENDEZ, D.; ALBERS, A.: Model-based Analysis and Specification of Functional Requirements and Tests for Complex Automotive Systems, Systems Engineering, Wiley, 2024, DOI: 10.1002/sys.21748

Zusammenfassung

Systeme im Automobilbereich realisieren komplexe Funktionalitäten durch das Zusammenspiel in dynamisch vernetzten Systemverbünden. Die einzelnen Systeme stellen hierbei Teilfunktionen bereit, die in weitreichenden Entwicklungsnetzwerken entwickelt und zu einer durch den Anwender erlebbaren Funktionalität integriert werden. Für die erfolgreiche Entwicklung und Validierung der Systeme haben Anforderungs- und Testspezifikationen eine herausragende Bedeutung, da diese Spezifikationen die Basis für die disziplin- und organisationsübergreifende Entwicklungsarbeit sind. Es besteht jedoch die Herausforderung, Anforderungen und Tests so zu spezifizieren, dass diese niederschwellig editiert und kommuniziert werden können, gleichzeitig aber präzise die erwarteten Systemeigenschaften definieren. Diese Arbeit beschreibt einen umfassenden modellbasierten Ansatz. Ausgehend von einem zentralen Systemmodell wird die frühe und automatisierte Analyse von Anforderungen unterstützt, um schließlich vollständige und konsistente Anforderungs- und Testspezifikationen aus dem Systemmodell auszuleiten. Die geeignete Kombination von Techniken des Model-based Systems Engineering (MBSE), des Natural Language Processing (NLP) und der szenariobasierten Modellierung forciert die formale aber dennoch intuitive und somit anwendungsorientierte Spezifikation von Anforderungen und Tests.

Summary

Systems in the automotive industry achieve complex functionalities through the interaction in dynamically interconnected system networks. The individual systems offer partial functions developed within extensive development networks, which are then integrated to create a functionality that can be experienced by the user. For the successful development and validation of these systems, requirement and test specifications play a vital role, as these specifications form the basis for development work across disciplines and organisations. However, it is challenging to specify requirements and tests in a way that they can be easily edited and communicated, while also precisely defining the expected system properties. This thesis describes a comprehensive model-based approach. Starting from a central system model, early and automated analysis of requirements is supported, leading to complete and consistent requirement and test specifications derived from the system model. The appropriate combination of techniques from Model-Based Systems Engineering (MBSE), Natural Language Processing (NLP), and scenario-based modeling enables a formal yet intuitive and application-oriented specification of requirements and tests.

Für meine Kinder
Lorenz, Luise und Felix.

In Gedenken an meinen lieben Bruder Florian.

Inhaltsverzeichnis

Seite

Anhang

1 Einleitung

Diese Arbeit entstand im Rahmen des BMBF Projekts MoSyS - Menschorientierte Gestaltung komplexer System of Systems. Das Projekt orientiert sich an dem Leitbild des *Advanced Systems Engineering* (ASE) [DAR+21] und verfolgt das Ziel, über die Entwicklung von neuen Methoden, Hilfsmitteln und IT-Werkzeugen die zukünftige Ingenieursarbeit zu gestalten. An diesem Ziel richtet sich die vorliegende Arbeit aus und beschreibt eine *Systematik zur integrativen Anforderungsanalyse und Testspezifikation für die Entwicklung von Systemen im Automobilbereich.*

1.1 Problematik

Die Automobilindustrie befindet sich in einer umfassenden Transformation. Der Übergang zur Elektromobilität, automatisiertes Fahren und die immer stärkere Vernetzung zur Realisierung von nachhaltigen Mobilitätssystemen sind nur einige Beispiele für aktuelle Herausforderungen. Demgegenüber stehen komplexe Organisationen, die in weitreichenden Lieferantennetzwerken die für die Transformation notwendige Produkte entwickeln. Besonders in der Automobilindustrie sind dabei in der Produktentwicklung hohe Sicherheits- und Qualitätsstandards zu berücksichtigen, die unter einem gleichzeitig hohen Kostendruck eingehalten werden müssen. Historisch sind hieraus vielschichtige Lieferantennetzwerke entstanden, um Kosten zu senken, das Entwicklungsrisiko zu verteilen und die Komplexität des Systems Fahrzeug beherrschbar zu machen (vgl. [Vog20]).

Diese Rahmenbedingungen spiegeln sich auch in den Systemarchitekturen und Entwicklungsprozessen wider. So waren Fahrzeugarchitekturen in der Vergangenheit durch eine Vielzahl von Steuergeräten geprägt, die abgrenzbare Teilfunktionen realisierten, die wiederum über standardiserte Kommunikationsschnittstellen eine integrierte Gesamtfunktionalität auf Fahrzeugebene bereitstellten (vgl. [Vog20]). Ansätze des Systems Engineering (SE) ermöglichten dabei die fachdisziplinübergreifende und verteilte Entwicklung des Systems Fahrzeug, das für ein definiertes Umfeld bestimmt war (vgl. [GCW+13]). Heutige, zunehmend durch Software definierte Fahrzeuge, zeigen jedoch zentralisiertere Architekturen. Dabei entsteht bei der Entwicklung dieser multidisziplinären Systeme eine neue Komplexität, da das Fahrzeug mit wechselnden Systemen interagiert. Im Vergleich zu früheren Architekturen besteht der Unterschied darin, dass das Fahrzeug mit externen Systemen interagiert, deren Eigenschaften sich über den Produktlebenszyklus des Fahrzeugs ändern können. Daraus leitet sich die Notwendigkeit ab, dass Anforderungen an das zu entwickelnde System fortlaufend geprüft und bei Bedarf angepasst werden müssen.

Als Leitbild für die erfolgreiche Entwicklung zukünftiger Systeme dient in dieser Arbeit das ASE. Das ASE ist durch die drei Handlungsfelder *Advanced Systems*, *Systems Engineering* und *Advanced Engineering* charackterisiert. Das Handlungsfeld Advanced Systems beschreibt zukünftige Systeme als intelligente cyber-physische Systeme, die sich dyna-

misch mit anderen Systemen vernetzen und eine soziotechnische Interaktion unterstützen. Das Handlungsfeld Systems Engineering fokussiert die interdisziplinäre Zusammenarbeit in der Systementwicklung und ermöglicht die zielgerichtete und kollaborative Entwicklung zukünftiger Systeme, was durch einzelne Fachdisziplinen wie Maschinenbau, Elektrotechnik und Informatik nicht zu leisten ist. Erst das Zusammenwirken der einzelnen Disziplinen auf Basis eines gemeinsamen Systemverständnisses macht die Entwicklung zukünftiger Systeme möglich. Dabei ist es notwendig, das zu entwickelnde soziotechnische System und das jeweilige Entwicklungsprojekt integrativ zu betrachten. Das Handlungsfeld Advanced Engineering umfasst hierfür die Weiterentwicklung von etablierten Engineering-Ansätzen durch neue IT-Werkzeuge, Methoden, Prozesse und Arbeitsabläufe, um eine kreative und agile Arbeit zu fördern [DAR+21, S. 28].

Vor diesem Hintergrund hat die Analyse von Anforderungen im Kontext der Systementwicklung in der Automobilindustrie und verwandten Brachen eine besondere Bedeutung [DAR+21, S. 61]. Ebenso ist die Validierung der Anforderungen ein zentraler Bestandteil in der Systementwicklung, um das Einhalten der Entwicklungsziele sicherzustellen [ABK+16]. Sowohl für die Analyse von Anforderungen als auch für die Validierung stoßen etablierte Vorgehensweisen jedoch an ihre Grenzen, da sich durch den hohen Vernetzungsgrad der Advanced Systems Entwicklungsziele und Anforderungen fortlaufend ändern [NL18]. Die hohen Sicherheits- und Qualitätsstandards in der Automobilindustrie erfordern jedoch eine fundierte Analyse und Spezifikation von Anforderungen und Tests. Um Entwicklungsingenieure in den Analyse- und Spezifikationstätigkeiten für die Systementwicklung- und validierung zu unterstützen, sind neue Vorgehensmodelle notwendig, die ein iteratives, problemorientiertes und modellbasiertes Vorgehen ermöglichen. Dabei fehlen geeignete Werkzeuge und Hilfsmittel, die bei den Analysetätigkeiten unterstützen, um zielgerichtet und kundenoritentiert Anforderungs- und Testspezifikationen für die Entwicklung von zukünftigen Sytemen zu erstellen.

1.2 Motivierendes Beispiel

Zur Einordnung der folgenden Kapitel, sowie zur Veranschaulichung der erarbeiteten Lösung und zur Evaluierung der Forschungsergebnisse, wird in dieser Arbeit ein durchgängiges Beispiel verwendet. Bild 1-1 zeigt hierzu eine vereinfachte Übersicht eines *Smart Charging* Anwendungsfalls. Am Beispiel der Elektromobilität ist hier skizziert, wie über die zunehmende Vernetzung von Systemen eine integrierte und systemübergreifende Funktionalität bereitgestellt wird. In dieser Arbeit wird dabei die Entwicklung eines Onboard-Ladegerätes (OBC) für batterieelektrische Fahrzeuge aus Sicht eines Tier1-Zulieferers betrachtet. Dieser OBC konvertiert die AC-Spannung aus dem Versorgungsnetz in DC-Spannung zum Laden der Fahrzeugbatterie [SL20]. Herausfordernd ist hierbei, dass das batterieelektrische Fahrzeug mit wechselnden externen Systemen, wie unterschiedlichen Ladestationen und Bediengeräten interagiert, die wiederum in unterschiedlichen Organisationen in unabhängigen Entwicklungsprozessen entwickelt werden. Das führt zu einem volatilen Entwicklungsumfeld bei gleichzeitig umfangreichen Anforderungen

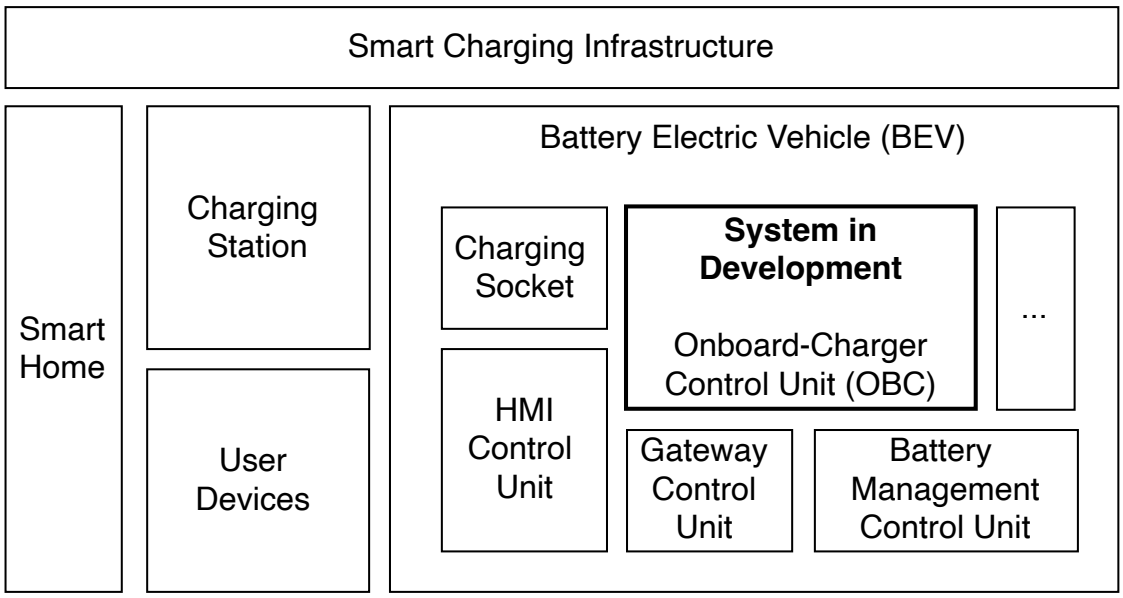


Bild 1-1: Ein Steuergerät als das zu entwickelnde System für die Realisierung der Ladefunktionalität im Kontext der Elektromobilität.

an das zu entwickelnde System. Um bspw. die Bedürfnisse auf einem globalen Markt zu adressieren, müssen unterschiedliche Versorgungsnetze in unterschiedlichen Regionen unterstützt werden. Ebenso existieren unterschiedliche Ladestandards, die in der Steuergeräteentwicklung zu berücksichtigen sind. Da die am Ladevorgang beteiligten Steuergeräte zudem hohe Sicherheitsstandards erfüllen müssen, resultiert hieraus eine insgesamt sehr hohe Entwicklungskomplexität [SL20].

Vor diesem technischen Hintergrund werden im weiteren Verlauf drei Funktionen zur Veranschaulichung und Evaluierung verwendet.

Funktion 1 - Regulierung der Ausgangsleistung (Derating): Bei einem aktivem Ladevorgang wird über Sensoren die Temperatur an unterschiedlichen Stellen innerhalb des OBCs erfasst. Werden innerhalb einer definierten Zeit Schwellenwerte überschritten, wird die Ausgangsleistung in Abhängigkeit der Temperatur reduziert, um eine Überhitzung des Steuergeräts zu vermeiden [SL20; WGK19].

Funktion 2 - Verriegelung des Ladesteckers (Plug Interlock): Die Verriegelung des Ladesteckers ist eine sicherheitsrelevante Funktion, die zur Vermeidung von Gefahrensituationen das Abziehen des Ladesteckers bei einem aktiven Ladevorgang verhindert. Hierzu findet ein Informationsaustausch zwischen dem OBC, der Ladedose innerhalb des Fahrzeugs und der externen Ladestation statt [SL20; WJK+20].

Funktion 3 - Zeitgesteuertes Laden (Timer Charging): Diese Funktion erweitert die grundlegende Ladefunktionalität des Fahrzeugs und erlaubt dem Anwender, den Ladevorgang zu konfigurieren. Über eine Benutzerschnittstelle kann u.a. der gewünschte Ladezustand (State of Charge (SOC)) zu einem definierten Zeitpunkt vorgegeben werden. Die Benutzerschnittstelle kann dabei ein Teil des Fahrzeugs sein, aber auch durch externe

Systeme wie Smartphones realisiert werden. Dabei ist zu berücksichtigen, dass bei einer Integration des Fahrzeugs in komplexe Elektromobilitäts-Architekturen (vgl. [KDB+19]), andere Systeme mit möglicherweise anderen Zielen Einfluss auf den Ladevorgang haben können [WMG+24].

1.3 Zielsetzung

Das Ziel dieser Arbeit ist eine *Systematik zur integrativen Anforderungsanalyse und Testspezifikation für die Entwicklung von Systemen im Automobilbereich.*

- Die Systematik soll Entwicklungsingenieure dabei unterstützen, Anforderungen an Systeme im Automobilbereich zu analysieren. Als Ausgangspunkt wird angenommen, dass ein initialer Satz an Stakeholder-Anforderungen vorliegt, der in Systemanforderungen zu überführen ist. Diese sind entsprechend der Vorgaben in der Automobilindustrie zu analysieren und in einer Spezifikation zu dokumentieren. Die resultierende Anforderungsspezifikation ist die Basis für den Systementwurf und die Implementierung.
- Neben der Anforderungsanalyse soll die Systematik eine konsistente Spezifikation von Testfällen unterstützen. Diese sind in einer Testfallspezifikation zu dokumentieren, die wiederum die Basis für den Entwurf und die Implementierung von Validierungssystemen, sowie die Automatisierung von Validierungsaktivitäten ist.
- Sowohl für die Anforderungsanalyse als auch für die Testfallspezifikation soll die Systematik zudem die systematische Externalisierung und Wiederverwendung von Lösungswissen unterstützen, um das vorhandene Wissen in nachfolgenden Entwicklungsiterationen für die Anforderungsanalyse und Testfallspezifikation anzuwenden.

Zur Adressierung der Entwicklungskomplexität im Kontext des ASE sollen die zuvor beschriebenen Teilziele durch modellbasierte Ansätze und geeignete Werkzeuge unterstützt werden, sodass ausgewählte Tätigkeiten zur Anforderungsanalyse und Testfallspezifikation automatisiert werden können und so ingesamt eine intuitive und arbeitsteilige Modellierung möglich wird.

2 Forschungsdesign und Aufbau der Arbeit

Dieses Kapitel strukturiert die einzelnen Forschungsaktivitäten und leitet den daraus resultierenden Aufbau dieser Arbeit ab. Die Forschungsaktivitäten orientieren sich an dem Design Science Research (DSR) Paradigma, das in Kapitel 2.1 eingeführt wird. Darauf aufbauend ist die Vorgehensweise in Kapitel 2.2 und schließlich der Aufbau der Arbeit in Kapitel 2.3 beschrieben.

2.1 Design Science Research (DSR)

Das DSR Paradigma unterstützt einen Problemlösungsprozess, in dem Wissen und Verständnis über einen Problembereich und die potentiellen Lösungen durch den Entwurf und die Anwendung von DSR Artefakten erweitert werden [HMP+04]. Dabei ist das Ziel von DSR die Erarbeitung von Gestaltungswissen, wie innovative Lösungen für wichtige Probleme effektiv erstellt werden können. Ein zentrales Element dieses Gestaltungswissens ist ein Bewertungsnachweis, der zeigt, inwieweit die gestaltete Lösung das identifizierte Problem lösen kann [VWH+20].

Für die Beschreibung des Problems, sowie der Erarbeitung und Bewertung der Lösung, schlagen HEVNER et al. ein DSR Framework vor [HMP+04], das in drei miteinander in Beziehung stehende Bereiche aufgeteilt ist:

1) Einen **Kontext**, der dadurch beschrieben wird, dass Menschen (Rollen, Möglichkeiten, Eigenschaften) innerhalb einer Organisation (Strategien, Strukturen, Kulturen, Prozesse) mit vorhandenen Technologien (Infrastrukturen, IT-Anwendungen, Kommunikationsplattformen etc.) ein Interesse an einer innovativen Lösung haben, um Aufgaben zu erfüllen, die sich aus vorhandenen Geschäftszielen ableiten.

2) **Forschungsaktivitäten** zur Entwicklung von Theorien und Artefakten und der Evaluierung, inwieweit das adressierte Problem damit gelöst werden kann.

3) Eine **Wissensbasis** mit Grundlagen (Theorien, Modellen, Methoden etc.), die für die Entwicklung der Lösung infrage kommen und entsprechende Methodiken (Datenanalysetechniken, Validierungskriterien etc.) zur Bewertung der Evaluationsergebnisse und Erweiterung der Wissensbasis.

Diese Aufteilung soll ermöglichen, dass ein Problem adressiert wird, welches für einen bestimmten Kontext eine hohe Relevanz hat und unter Berücksichtigung der vorhandenen Wissensbasis eine wissenschaftlich fundierte Erarbeitung einer innovativen Lösung ermöglicht [HMP+04]. Im Zentrum steht hierbei das entwickelte Artefakt. Daran anschließend beschreiben VOM BROCKE et al. [VWH+20], dass Wissen über ein Problem in einem bestimmten Kontext und das Wissen über mögliche Lösungen unabhängig voneinander existieren können. Demnach entsteht Gestaltungswissen erst dadurch, dass der Kontext (Problemraum) und potentielle Lösungen (Lösungsraum) über die Evaluierung von Arte-

fakten miteinander in Beziehung gesetzt werden. Dabei kann das Gestaltungswissen in unterschiedlichen Formen einen relevanten Beitrag zur Problemlösung liefern und entsprechend die Wissensbasis erweitern. GREGOR & HEVNER [GH13] schlagen hierfür eine Kategorisierung in drei unterschiedliche Artefakt-Level vor, die sich hinsichtlich Abstraktion und Reifegrad unterscheiden.

- **Level 1**: anwendbare Instantziierungen (Werkzeuge, implementierte Prozesse) (vgl. [HMP+04])
- **Level 2**: aufkommende/partielle Design-Theorie (Methoden, Modelle, Design Prinzipien, technologische Regeln) (vgl. [CSG15])
- **Level 3**: umfassende Design Theorie (vgl. [GJ07])

Nach dieser Kategorisierung kann eine Design-Theorie bottom-up entwickelt werden, beginnend mit einem neuen Artefakt (Level 1), wobei Abstraktion und Reifegrad mit jeder Iteration im Problemlösungsprozess zunehmen können. Hier ist ein wichtiger Aspekt, dass sich der Reifegrad innerhalb des Entwurfswissens sowohl auf das jeweilige Artefakt als auch auf das adressierte Problem bezieht [GH13]. So kann bspw. die Evaluierung eines Level 1 Artefakts zur Herleitung von Design-Prinzipien und technologischen Regeln (Level 2) genutzt werden, aber auch zu einer Neuausrichtung oder Eingrenzung der Problemstellung führen. Im Kontext dieser Arbeit liegt der Fokus auf der Erarbeitung von Level 1 und Level 2 Artefakten.

2.2 Vorgehensweise

Angelehnt an METH et al. [MMM15] und basierend auf dem Prozessmodel nach PEFFERS et al. [PTR+07] erfolgt die Erarbeitung von Gestaltungswissen in dieser Arbeit einem iterativen Problemlösungsprozess, in dem die einzelnen Iterationen entsprechend der Darstellung in Bild 2-1 in die Phasen *Problembewusstsein, Lösungsvorschlag, Artefakt-Design, Demonstration, Evaluation* und *Schlussfolgerung* gegliedert sind.

Iteration 1: Das Problembewusstsein basiert auf einer mehrjährigen teilnehmenden Beobachtung in der Steuergeräteentwicklung. Ausgehend von einer ersten Literaturanalyse wird das adressierte Problem analysiert. Auf Basis der ersten Problemanalyse wird ein Artefakt-Konzept und dessen Umsetzung beschrieben. Die Demonstration erfolgt in der ersten Iteration innerhalb eines Workshops und die Evaluierung über ein Experiment innerhalb eines Industrieunternehmens. Basis für die Evaluation ist die oben beschriebene *Funktion 1 - Regulierung der Ausgangsleistung (Derating)*. Die erste Iteration schließt mit einer Reflektionsphase ab. Die Zwischenergebnisse dieser Iteration sind in [WGK19] zusammengefasst.

Iteration 2: Auf Basis der Reflektionsphase aus der ersten Itration beginnt die zweite Iteration mit der Anpassung und Konkretisierung der identifizierten Handlungsfelder. Die Literaturanalyse wird fortgesetzt und führt zu einer Anpassung des Artefakt-Konzepts so-

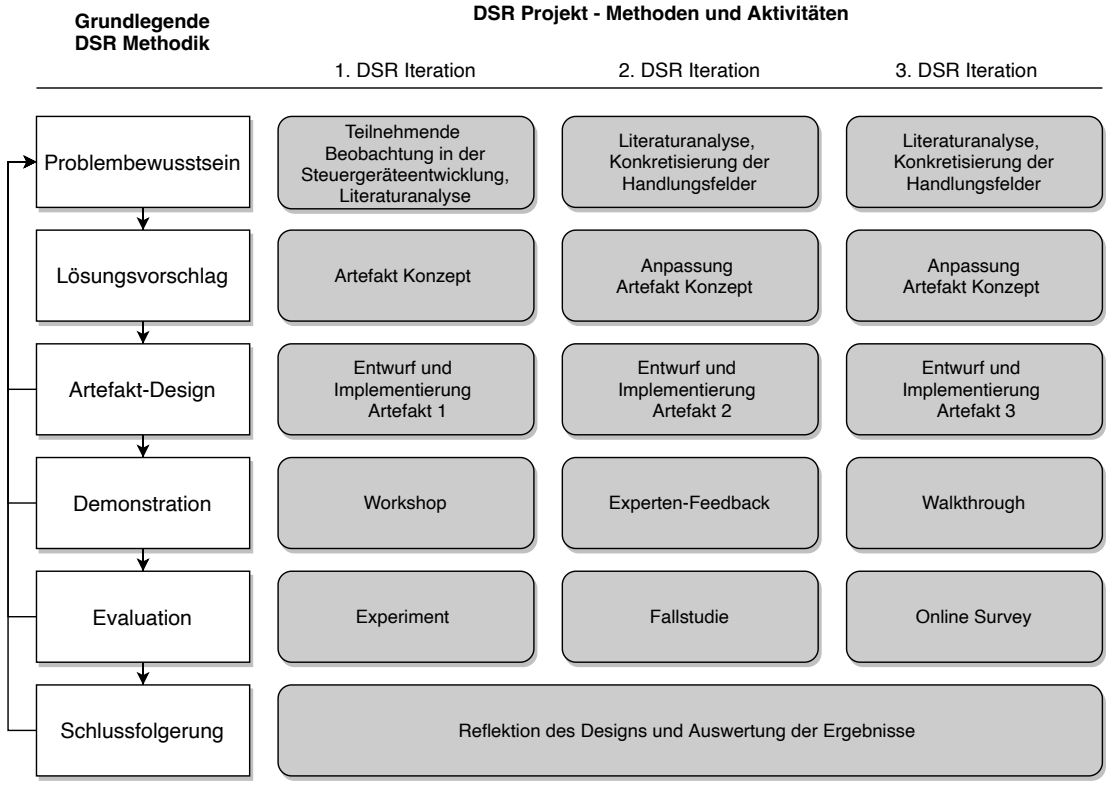


Bild 2-1: Die dieser Arbeit zugrunde liegende DSR Methodik mit den zugeordneten Methoden und Aktivitäten aufgeteilt in drei Iterationen, basierend auf [MMM15].

wie dessen Umsetzung. Die Demonstration erfolgt im Kontext des Industrieunternehmens durch die Anwendung des Artefakts anhand der *Funktion 2 - Verriegelung des Ladesteckers (Plug Interlock)*. Die Evaluierung ist mit Hilfe einer Fallstudie dokumentiert. Die wesentlichen Zwischenergebnisse dieser Iteration sind in [WJK+20; WG21; WGW+21; WFG+21; FFV+23] zusammengefasst.

Iteration 3: Auf die Reflektionsphase aus der zweiten Iteration folgt die erneute Anpassung und Konkretisierung der Handlungsfelder, sowie eine Fortsetzung der Literaturanalyse. Wie in den vorangegangenen Iterationen, erfolgt anschließend erneut die Implementierung eines angepassten Artefakts. Das finale Artefakt wird zur Demonstration innerhalb des Industrieunternehmens angewendet. Basis ist hier die *Funktion 3 - Zeitgesteuertes Laden (Timer Charging)*. Die Evaluierung basiert auf einer Anwendung des Artefakts in studentischen Projekten im Rahmen einer Lehrveranstaltung und ist mit Hilfe eines Surveys dokumentiert. Die wesentlichen Zwischenergebnisse aus der dritten Iteration sind in [WMG+24] zusammengefasst.

2.3 Aufbau der Arbeit

Basierend auf den Elementen des Gestaltungswissens für ein spezifisches DSR Projekt [VWH+20] zeigt Bild 2-2 den Aufbau dieser Arbeit, unterteilt in den Problem- und Lösungsraum, die über die Evaluierung miteinander in Beziehung stehen. Nach der Einleitung

in Kapitel 1 und den in diesem Kapitel 2 beschriebenen Hintergründen zur Forschungsmethodik, folgt die Problemanalyse in Kapitel 3, deren Ergebnis die Beschreibung des Problemraums ist. Dieser Problemraum ist unterteilt in eine Kontextbeschreibung, eine Beschreibung der identifizierten Handlungsfelder und in daraus abgeleiteten Gütekriterien. Diese Gütekriterien sind in einer Anforderungsliste beschrieben und bilden die Basis für

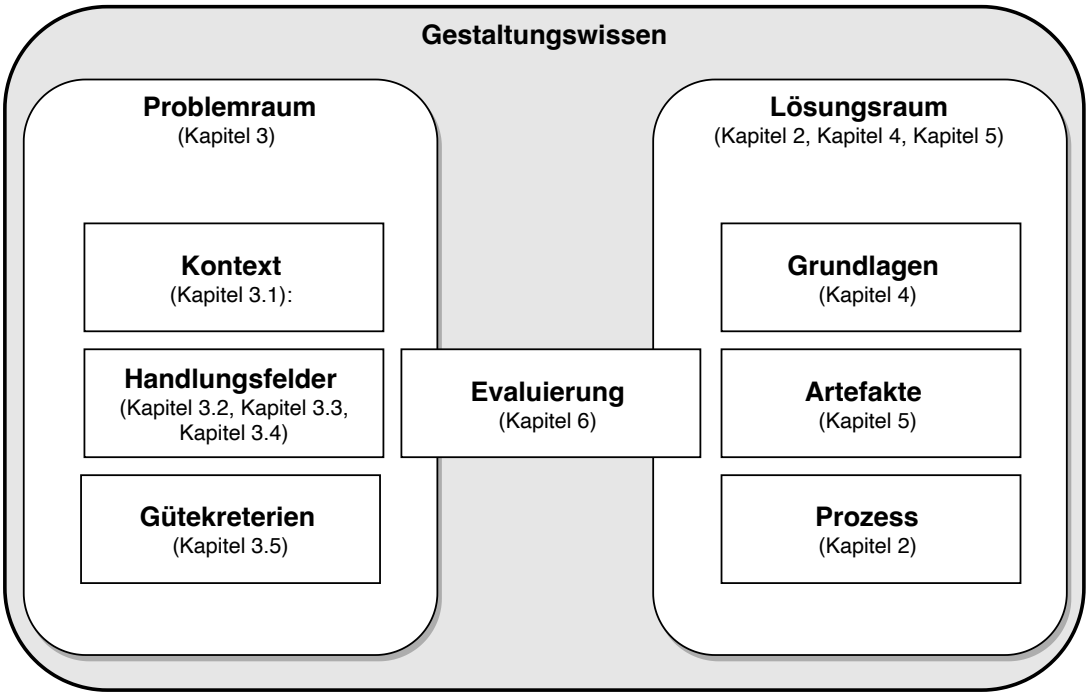


Bild 2-2: Gestaltungswissen für ein spezifischen DSR Projekt nach [VWH+20]

die in Kapitel 6 beschriebene Evaluierung. Ein zentraler Bestandteil dieser Arbeit sind die entwickelten Artefakte (Kapitel 5), die anhand der zuvor beschriebenen Vorgehensweise entwickelt wurden. Die Entwicklung der Artefakte basiert dabei auf den in Kapitel 4 beschriebenen Grundlagen. Trotz des stark iterativen Vorgehens, sind die Inhalte in den nachfolgenden Kapitel zusammenhängend dargestellt. Lediglich in der Evaluierung (Kapitel 6) sind die einzelnen DSR Iterationen explizit beschrieben, um die Ergebnisse und das Vorgehen nachvollziehbar zu machen.

3 Problemanalyse

Das Ergebnis der Problemanalyse sind Anforderungen an eine *Systematik zur integrativen Anforderungsanalyse und Testspezifikation für die Entwicklung von Systemen im Automobilbereich*. Dem DSR Ansatz folgend definieren diese Anforderungen das Zielbild für die Erarbeitung der einzelnen DSR Artefakte. Ebenso sind die Anforderungen die Basis für die Evaluierung der entwickelten Artefakte (vgl. [HMP+04; VWH+20]). Zur Herleitung der Anforderungen ist zunächst der Kontext der Arbeit in Kapitel 3.1 beschrieben. Darauf aufbauend ist die Problemanalyse in drei Handlungsfelder strukturiert. Das erste Handlungsfeld (Kapitel 3.2) untersucht die *Spezifikation und Analyse von Anforderungen*. Das zweite Handlungsfeld (Kapitel 3.3) befasst sich mit dem *Entwurf und der Spezifikation von Tests*. Sowohl die Spezifikation und Analyse von Anforderungen als auch der Entwurf und die Spezifikation von Tests sind dabei wesentliche Elemente des Systems Engineering (SE) und Grundlage für die Entwicklung von erfolgreichen Systemen (vgl. [DRF+15, S. 11]). Im Rahmen des SE umfasst das dritte Handlungsfeld die systematische *Wiederverwendung von Lösungswissen* (Kapitel 3.4). Die Untersuchung der drei Handlungsfelder führt zu der Problemabgrenzung in Kapitel 3.5 und schließlich zu der Herleitung der Gütekriterien für die entwickelten Artefakte in Kapitel 3.6.

3.1 Kontext der Arbeit

Das Ziel dieser Arbeit ist die Entwicklung einer Systematik, die Systemarchitekten und Test-Designer bei der Spezifikation und Analyse von Anforderungen bzw. bei dem Entwurf und der Spezifikation von Tests unterstützt. Der Fokus liegt dabei auf der Systementwicklung aus der Sicht eines Zulieferunternehmens, das Systemlösungen an Fahrzeughersteller (OEMs) bereitstellt. Dabei wird in dieser Arbeit innerhalb eines ganzheitlichen Produktentstehungsprozesses [FG13] der Zeitraum vom Start der Entwicklung (Entwicklungsauftrag durch den OEM) bis zum Ende der Entwicklung betrachtet. Das Ende der Entwicklung kann dabei durch den Start der Produktion des entwickelten Systems definiert sein, der Auftraggeber kann aber auch einen späteren Zeitpunkt festlegen, sodass die Produktion des Systems und die Entwicklung neuer Funktionalitäten parallel erfolgen kann.

Für eine detaillierte Einordnung der Arbeit werden im Folgenden die wesentlichen Begriffe definiert. Daran anschließend sind allgemeine Rahmenbedingungen für die Systementwicklung im Automobilbereich beschrieben. Die Kontextbeschreibung schließt mit einer Einordnung der Arbeit in das Forschungsfeld des ASE [DAR+21].

3.1.1 Begriffsdefinitionen

Nach DUMITRESCU wird der Begriff **Systematik** in dieser Arbeit als *ein universelles Rahmenwerk, das ein Vorgehensmodell sowie dedizierte Hilfsmittel zur erfolgreichen Umsetzung der Entwicklung technischer Systeme bereitstellt*, verstanden. Das **Vorgehens-**

modell strukturiert die in dieser Arbeit betrachteten Aspekte innerhalb der Entwicklung und unterstützt die projekt- und produktübergreifende Integration der Systematik in vorhandene Entwicklungsprozesse. **Hilfsmittel** sind hier u.a. Methoden, Modellierungssprachen, Lösungsmuster und Werkzeuge [Dum10, S. 5].

Die **Entwicklung** kann als organisatorische Einheit innerhalb eines Unternehmens oder als Prozess zur Produkterstellung verstanden werden [EM17, S. 11]. Im Kontext dieser Arbeit wird die Entwicklung als Gesamtheit der Tätigkeiten zum Entwurf sowie der Ausarbeitung einer Lösung und deren Umsetzung verstanden. Das Resultat der Entwicklung ist ein technisches System (vgl. [Dum10, S. 5]). Die Tätigkeiten für die Entwicklung von Systemen werden dabei durch unterschiedliche Rollen in unterschiedlichen Fachabteilungen durchgeführt, die wiederum Teil der Entwicklung als organisatorische Einheit sind.

Ein **System** kann als eine integrierte Menge von Systemelementen (Teilsystemen) gesehen werden, die ein bestimmtes Ziel erreicht [DRF+15, S. 5]. Durch die Systemgrenze wird das System von seinem Umfeld abgegrenzt. Wie in Bild 3-1 veranschaulicht, stehen die einzelnen Systemelemente miteinander in Beziehung, um eine Funktion zu realisieren, die Eingangsgrößen des Systems in Ausgangsgrößen umwandelt. Auf diese Weise interagiert das System mit seinem Umfeld [EM17, S. 28]. Demnach können Input-Transformation-

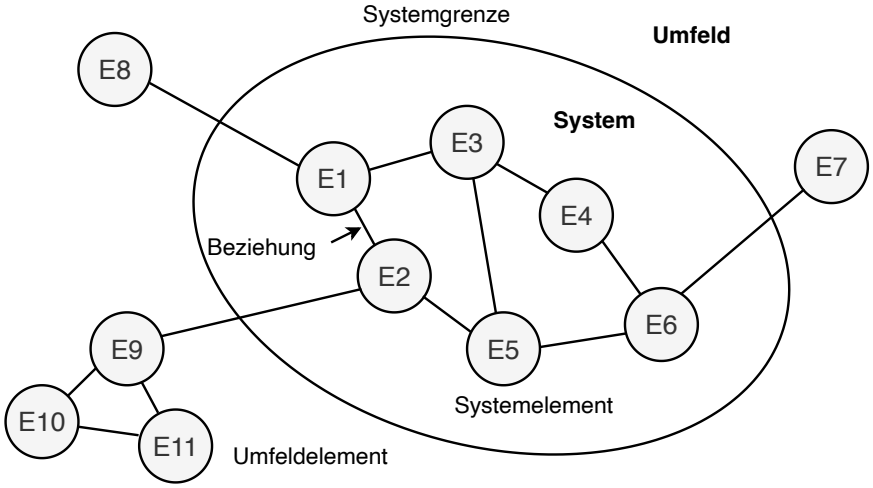


Bild 3-1: Schematische Darstellung eines Systems in seinem Umfeld (nach [Hab12])

Output-Prozesse durch transformierende Systeme beschrieben werden [Luh04, S. 47 ff.]. Zu berücksichtigen ist jedoch, dass bei einer Vielzahl von Interaktionen des Systems mit seinen Umfeldelementen eine Beschreibung des Systemverhaltens durch einfache und voneinander unabhängige Transformationsprozesse nicht mehr möglich ist [Luh04, S. 48] [HP85]. Reaktive Systeme hingegen adressieren die fortlaufende Interaktion mit der Systemumwelt [HP85]. Anstatt einer funktionsorientierten Sichtweise steht hier ein nachrichtenbasierter und kontinuierlicher Informationsaustausch zwischen dem System und seiner Umwelt im Vordergrund (ebd.).

Für die ***Systematik*** *zur integrativen Anforderungsanalyse und Testspezifikation zur* ***Ent-***

***wicklung** von **Systemen** im Automobilbereich* sind beide Sichtweisen relevant. Am zuvor eingeführten Beispiel (Kapitel 1.2) wird deutlich, dass es einerseits eine transformierende Grundfunktionalität geben kann (AC Spannung wird durch den OBC in DC Spannung umgewandelt), andererseits entsteht durch die Integration des Fahrzeugs in komplexe Ladeinfrastrukturen eine kontinuierliche Interaktion mit weiteren vernetzen Systemen, bspw. zur automatisierten Ladeplanung in einem Smart-Charging Kontext (s. [SL20]). Somit sind auf der Spezifikationsebene für Systeme im Automobilbereich beide Sichtweisen zu berücksichtigen [Bro06].

3.1.2 Systementwicklung im Automobilbereich

Wie zu Beginn (Kapitel 1.1) bereits angedeutet, ist die Entwicklung von Systemen im Automobilbereich von komplexen und historisch gewachsenen OEM - Zulieferer Beziehungen geprägt. Hinzu kommt, dass die Wiederverwendung von bereits entwickelten Systemen durch den hohen Kostendruck eine hohe Relevanz hat. Neben den bereits genannten Einflüssen auf die System- und Organisationsarchitektur (bspw. starke Modularisierung), führen die Rahmenbedingungen innerhalb der Automobilindustrie zu einem hohen Maß an Standardisierung [Vog20; Bro06].

Die Implementierung der Entwicklungsprozesse und die Bewertung ihrer Leistungsfähigkeit basiert auf dem ASPICE® Standard (Automotive Software Process Improvement and Capability Determination) [VDA], der Referenzprozesse und Reifegradmodelle beschreibt. Ziele, die durch die ASPICE konforme Prozessimplementierung erreicht werden sollen, sind u.a. eine bessere Planbarkeit der Entwicklungskosten, sowie eine einheitliche Qualitätsüberprüfung von Entwicklungsartefakten. Die Referenzprozesse sind dafür in drei Kategorien unterteilt, die eine Strukturierung in primäre, organisationale und unterstützende Lebenszyklus-Prozesse ermöglicht. Zudem erfolgt eine Einteilung in die Prozessgruppen Systems Engineering und Software Engineering [VDA, S. 12]. Die einzelnen Referenzprozesse innerhalb der beiden Prozessgruppen sind anhand des V-Modells (vgl. [Web09, S. 11]) strukturiert, wodurch die Beziehung zwischen den Anforderungen und dem Software/System Design auf der einen Seite, und Integrations- , Verifikations- und Validierungstätigkeiten auf der anderen Seite, veranschaulicht wird.

Im Kontext dieser Arbeit sind insbesondere der Referenzprozesse zur Systemanforderungsanalyse, sowie die Referenzprozesse zur Verifikation und Validierung (V&V) relevant. Im Rahmen der Systemanforderungsanalyse besteht die Aufgabe darin, *Stakeholder Anforderungen* in *Systemanforderungen* zu überführen, die wiederum die Basis für die nachgelagerten Systemdesign-Prozesse sind. Dabei sind die resultierenden Systemanforderungen hinsichtlich Korrektheit und Verifizierbarkeit zu überprüfen. Ein zentrales Ergebnis der Systemanforderungsanalyse ist die *Systemanforderungsspezifikation* [VDA, S. 39 f.]. Innerhalb der Systemintegration und der Systemqualifizierung und den damit einhergehenden V&V Tätigkeiten sind die Systemelemente zu einem Gesamtsystem zu integrieren, und es ist ein Nachweis zu erbringen, dass die integrierten Systemelemente mit der Systemar-

chitektur übereinstimmen. Weitergehend ist nachzuweisen, dass das integrierte System die Systemanforderungen erfüllt und zur Auslieferung bereit ist. Zentrale Ergebnisse der Referenzprozesse zur V&V sind *Testspezifikationen* [VDA, S. 43 ff.].

3.1.3 Advanced Systems Engineering - ASE

Das ASE ist das Leitbild für die *erfolgreiche Gestaltung von innovativen Marktleistungen und deren Entstehung* [DAR+21, S. 31]. Die Marktleistung entsteht durch das Zusammenwirken einer organisierten Menge von Mitarbeitern in einem Unternehmen und aufgabenspezifischen Technologien (bspw. IT-System und Werkzeuge) (vgl. [Rop09]). Hier kann zwischen dem System der Marktleistung als Ergebnis der Entwicklung (Advanced Systems), sowie dem System der Marktleistungsentstehung als organisatorische Einheit, unterschieden werden. Beide Systeme können dabei als soziotechnische Systeme betrachtet werden, die sich wechselseitig stark beeinflussen. Einerseits beeinflusst die Architektur der zu entwickelnden Marktleistung den Aufbau der Organisation, andererseits wird durch den Aufbau der Organisation die Architektur der Marktleistung beeinflusst [DAR+21, S. 102]. Beide Sichtweisen auf die soziotechnischen Systeme sind dabei gekennzeichnet durch eine starke und dynamische Vernetzung. Diese zunehmende Vernetzung führt zu immer komplexeren Schnittstellen, sowohl in den Prozess- und Organisationsstrukturen, als auch in den technischen Systemen [DAR+21, S. 58] [BBR21] [AL12].

Neben der hohen Schnittstellenkomplexität führt der hohe Vernetzungsgrad zu einem sehr volatilen Entwicklungsumfeld, da die einzelnen Systeme in unabhängigen Entwicklungsprozessen in unterschiedlichen Organisationen entwickelt werden, die Realisierung der Gesamtfunktionalität jedoch nur über die Integration der einzelnen Systeme möglich ist (s. Bild 1-1 mit Systemen zur Realisierung eines Smart-Charging Anwendungsfalls). Daraus folgt, dass sich Anforderungen kontinuierlich ändern können und für die erfolgreiche Entwicklung von Advanced Systems kontinuierlich zu prüfen und aufeinander abzustimmen sind [NL18]. Das bezieht sich zum einen auf das System der Marktleistung (- werden die Anforderung richtig realisiert und erfüllt das System die Erwartungen der Stakeholder?), und zum anderen auf das System der Marktleistungsentstehung (- sind die Anforderungen und Ziele für die einzelnen Systeme zwischen den an der Entwicklung beteiligten Organisationen abgestimmt und dokumentiert?).

Vor diesem Hintergrund haben sowohl die Anforderungsspezifikationen als auch die Testspezifikationen für die Entwicklung von zukünftigen Systemen im Automobilbereich einen hohen Stellenwert und beeinflussen maßgeblich die abteilungs- und unternehmensübergreifende Zusammenarbeit, da diese Spezifikationen als Schnittstellendokumente für die interdisziplinäre Zusammenarbeit verstanden werden können [WHK+20].

3.2 Spezifikation und Analyse von Anforderungen

Die Spezifikation und die Analyse von Anforderung sind Teil des *Requirements Engineering* (RE). Das *International Requirements Engineering Board* (IREB) definiert RE als:

> *„(...) systematic and disciplined approach to the specification and management of requirements with the goal of understanding the stakeholders' desires and needs and minimizing the risk of delivering a system that does not meet these desires and needs.*[1] *“*

Wie zuvor bereits angedeutet, sind Anforderungsspezifikationen ein Ergebnis der Analyse- und Spezifikationstätigkeiten ein zentrales Artefakt innerhalb des RE, das als Grundlage für die Systementwicklung dient. Neben dieser technischen Sicht sind die Anforderungsspezifikationen in der Automobilindustrie zusätzlich Vertragsgrundlage und Basis für die Zusammenarbeit zwischen OEM, Zulieferer und weiteren Entwicklungspartnern. Dabei ist für das Management der Anforderungen in der Automobilindustrie eine Aufteilung in *Lastenheft* und *Pflichtenheft* etabliert. Über diese Aufteilung soll festgehalten werden, *was* durch den Auftraggeber gefordert wird (Lastenheft) und *wie* der Auftragnehmer plant, diese Forderungen umzusetzen (Pflichtenheft) [WW02]. Diese klare Aufteilung ist in der Praxis jedoch nicht einzuhalten. Ein Problem ist, dass der OEM unterschiedliche Zulieferer damit beauftragt, Systeme mit gleicher Funktionalität zu entwickeln, die durch den OEM über Produktlinien hinweg integrierbar sein müssen. Hierzu ist es nötig, Teile der Lösung bereits im Lastenheft vorzugeben, um die Kompatibilität und Integrierbarkeit sicherzustellen. Andere Probleme entstehen nach WEBER & WEISBROD durch die Aufteilung von Anforderungen in unterschiedliche Disziplinen, wodurch auf Lastenheftebene separate aber miteinander in Beziehung stehende Probleme definiert werden müssen. Über die Spezifikation von Lösungselementen auf Lastenheftebene kann hier der notwendige Aufwand für die Koordinierung reduziert werden, was insbesondere notwendig ist, wenn die Lösungen durch unterschiedliche Unternehmen zugeliefert werden [WW02].

3.2.1 Spezifikation von Anforderungen in natürlicher Sprache

Neben einer Trennung in Lasten- und Pflichtenheft zur Definition und Abstimmung des Entwicklungsauftrags erfolgt im Rahmen des RE die Spezifikation und der Austausch von Anforderungen zwischen den Unternehmen überwiegend in natürlicher Sprache [LTK19]. Hierdurch soll eine allgemeinverständliche Basis für den Austausch von Informationen zwischen den Disziplinen geschaffen werden, wobei insbesondere auch nicht technische Disziplinen mit einbezogen werden können [LTK+18]. SIKORA et al. zeigen jedoch, dass die Spezifikation in natürlicher Sprache von Entwicklern in der Automobilindustrie und verwandten Branchen nicht als zufriedenstellend wahrgenommen wird. Gründe hierfür sind, dass der Umgang mit großen Anforderungsmengen als fehleranfällig gesehen wird und

[1] https://www.ireb.org/de/cpre/cpre-glossary/ CPRE Glossar - zuletzt abgerufen am 8.9.2022

dass die Analyse und Bewertung von Anforderungen in der Regel manuell erfolgt, was zu hohen Aufwänden führt [STP12]. Ergänzend beschreiben ALMEFELT et al., dass komplexe Anforderungsspezifikationen in natürlicher Sprache schwer überschaubar sind, was oft zu widersprüchlichen und nicht eindeutigen Spezifikationen führt. Ebenso geben die Autoren an, dass die Nachverfolgung der richtigen Umsetzung der Anforderungen herausfordernd ist [ABN+06], was insbesondere auch die V&V Aktivitäten mit einschließt [BRB+13].

MAVIN et al. [MWH+09] [MW19] fassen häufig auftretende Probleme in natürlich sprachlichen Anforderungen wie folgt zusammen:

- **Mehrdeutigkeit**: Ein Wort oder ein Satz hat zwei oder mehr verschiedene Bedeutungen [BK04].
- **Ungenauigkeit**: Mangel an Präzision, Struktur oder Details.
- **Komplexität**: Zusammengesetzte Anforderungen mit komplexen Teilklauseln oder mehreren zusammenhängenden Aussagen.
- **Auslassung**: Fehlende Anforderungen, insbesondere Anforderungen zur Behandlung unerwünschter Verhaltensweisen.
- **Duplikation**: Wiederholung von Anforderungen, die denselben Bedarf definieren.
- **Wortreichtum**: Verwendung einer unnötigen Anzahl von Wörtern.
- **Unangemessene Umsetzung**: Aussagen darüber, wie das System aufgebaut sein sollte, anstatt darüber, was es tun sollte.
- **Unprüfbarkeit**: Anforderungen, die sich bei der Implementierung des Systems nicht als wahr oder falsch erweisen.

Trotz der vielen bereits identifizierten Probleme ist die natürliche Sprache zur Spezifikation von Anforderungen in der Automobilindustrie nach wie vor etabliert [STP12] [LTK19]. Ein Grund für die Etablierung von Anforderungsspezifikationen in natürlicher Sprache ist die Möglichkeit, Informationen niederschwellig zu erfassen und auszutauschen. Einerseits ist somit eine intuitive und flexible Spezifikation der Anforderungen möglich, andererseits kann dies jedoch zu sehr heterogenen und qualitativ schlechten Spezifikationen führen. Dieser Konflikt zwischen intuitiver und flexibler Spezifikation auf der einen Seite und Qualitätsproblemen auf der anderen Seite wird u.a. in den von FEMMER et al. beschriebenen Symptomen für geringe Qualität von Anforderungsspezifikationen deutlich (s. Tabelle 3-1) [FMJ+14] (vgl. [ISOf]).

Nach MAVIN et al. [MWH+09] existiert bereits eine umfangreiche Wissensbasis, um die bekannten Herausforderungen in der Spezifikation von Anforderungen zu adressieren (vgl. [Hoo93]). Ergänzend sind Qualitätskriterien in Standards festgelegt (s. [ISOf]) und durch das IREB zusammengefasst. FEMMER & VOGELSANG zeigen jedoch, dass die bestehenden Qualitätsstandards nicht ausreichen. Sie betonen, dass die Spezifikation von qualitativ hochwertigen Anforderungen kein Selbstzweck ist, sondern innerhalb eines

Anzeichen	**Beschreibung**
Vergleichende Phrasen	beinhalten die erste Steigerung eines Adjektivs und dienen als konkreter Vergleich zu einem anderen Bezugspunkt, z. B. *besser als, höherwertiger*. Der Unterschied sollte messbar oder überprüfbar sein.
Superlative	sind die höchste Steigerung eines Adjektivs, wie z. B. *am besten, am meisten*, und drücken einen Vergleich mit der Haupteinheit aus. Für eine vollständige Spezifikation der Anforderung sollte ein Bezugspunkt angegeben werden, mit dem verglichen werden kann.
Subjektive Sprache	beinhaltet Formulierungen, die oft Eigenschaften eines Systems beschreiben, die schwer zu messen sind, wie z. B. *benutzerfreundlich, leicht zu bedienen, kostengünstig*. Die Semantik ist nicht objektiv, so dass die Bewertung der Benutzerfreundlichkeit stark vom jeweiligen Benutzer abhängen kann.
Vage Pronomen	die unklare Beziehungen eines Pronomens zum vorherigen Kontext beschreiben. Oft ist nicht klar, worauf sich Pronomen wie *es, dies, das* beziehen.
Mehrdeutige Adverbien und Adjektive	beschreiben das Phänomen, dass Eigenschaften in der Regel nicht detailliert beschrieben werden, wie z. B. *fast immer, signifikant, minimal*. An dieser Stelle fehlen Informationen.
Offene, nicht überprüfbare Begriffe	lassen Interpretationen zu, z. B. ist *Unterstützung bieten* nicht näher spezifiziert. Darüber hinaus können die Angaben zu Größen und Mengen unzureichend sein, so dass sie nicht messbar/-prüfbar sind, wie z. B. *aber nicht beschränkt auf, als Minimum*.
Schlupflöcher	wie z. B. *wenn möglich, soweit angemessen, soweit zutreffend*, können dazu führen, dass Teile der Spezifikation ignoriert werden. Es muss geklärt werden, was ein beschriebenes Verhalten ermöglicht oder verhindert. Darüber hinaus sollte definiert werden, wie ein System in Ausnahmesituationen arbeitet, z. B. *wenn nicht möglich*.
Negative Aussagen	umfassen Eigenschaften oder Funktionen, die ein System nicht implementieren sollte, was zu einer Unterspezifizierung führen kann.

Tabelle 3-1: Kriterien für die Spezifikation von Anforderungen in natürlicher Sprache [FV19] (vgl. [Juh21, S. 49])

Entwicklungsprojekts für unterschiedliche Aufgaben verwendet wird und hinsichtlich übergeordneter Ziele wie *Bedürfnisse der Stakeholder verstehen*, *Einigung erzielen*, *Schaffung desselben mentalen Modells für alle Stakeholder* und *Strukturierung und Verwaltung anforderungsbezogener Aktivitäten* zu bewerten ist [FV19].

3.2.2 Spezifikation von Anforderungen in strukturierter natürlicher Sprache

Wie zuvor angedeutet, existieren bereits zahlreiche Ansätze, die eine Qualitätsverbesserung von Anforderungsspezifikationen adressieren. Neben der Standardisierung sind dabei Satzmuster oder Satzschablonen naheliegende Ansätze (vgl. [MWH+09; Hol10; FH15; FHK+16; Sop16; MW19]), die durch Einschränkung der natürlichen Sprache die Freiheitsgrade für die Spezifikation verringern und so die Anforderungsqualität positiv beeinflussen sollen [WAB+10].

Aus der Motivation, trotz erhöhter Formalisierung eine möglichst einfach anzuwendende Spezifikationstechnik bereitzustellen, ist der EARS (Easy Approach to Requirements Syntax) Ansatz entstanden. Diese Syntax basiert auf der in Listing 3.1 beschriebenen grundlegenden Struktur und einem Regelsatz, aus dem hervorgeht, dass es keine oder mehrere *Vorbedingungen*, keinen oder einen *Trigger*, einen *Systemnamen*, sowie eine oder mehrere *Systemantworten* geben kann.

```
While <optional pre-condition>, when <optional trigger>,
   the <system name> shall <system response>
```

Listing 3.1: Grundlegene EARS-Syntax [MWH+09].

Aus dieser grundlegenden Syntax leiten *Mavin* et al. in Kombination mit dem Regelsatz Anforderungsmuster[2] ab, die im folgenden aufgelistet sind [MWH+09]:

- Allgemeingültige Anforderungen

  ```
  The <system name> shall <system response>
  ```

- Zustandsgetriebene Anforderungen

  ```
  While <precondition(s)>, the <system name> shall
     <system response>
  ```

- Eventgetriebene Anforderungen

  ```
  When <trigger>, the <system name> shall <system
     response>
  ```

- Optionale Feature Anforderungen

[2] https://alistairmavin.com/ears/, zuletzt abgerufen am 7.11.2022

```
Where <feature is included>, the <system name>  shall
   <system response>
```

- Anforderungen für unerwünschtes Systemverhalten

```
If <trigger>, then the <system name> shall <system
   response>
```

- Komplexe Anforderungen

```
While <precondition(s)>, When <trigger>, the <system
   name> shall <system response>
```

In der von MAVIN et al. durchgeführten Fallstudie wurden Anforderungen aus abgeschlossenen Entwicklungsprojekten in die EARS Syntax überführt. Die Ergebnisse der Fallstudie zeigen, dass die Verwendung der Syntax einen positiven Effekt auf die oben genannten Probleme hat [MWH+09] [MW19].

Die SOPHISTEN[3] verfolgen einen ähnlichen Ansatz (s. Bild 3-2) und stellen einen umfangreichen Satz an Satzschablonen für die Spezifikationen von Anforderungen bereit [Sop16], wobei explizit zwischen Schablonen für funktionale und nicht-funktionale Anforderung unterschieden wird. Zudem ist hier eine Vorgehensweise für die Anwendung der Satzschablonen beschrieben. Nach dieser Vorgehensweise wird empfohlen, zunächst

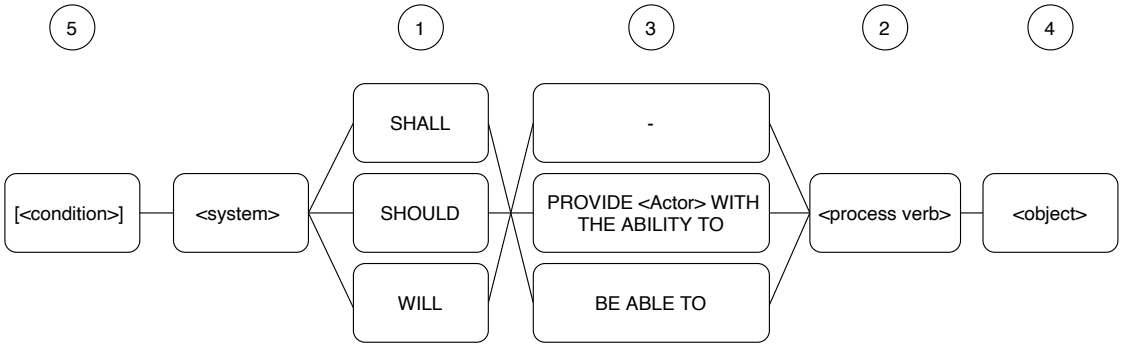


Bild 3-2: Satzschablone für die Spezifikation von funktionalen Anforderungen [Sop16, S. 20]

die rechtliche Verbindlichkeit einer Anforderung über die Schlüsselwörter *shall*, *should* und *will* zu definieren. So kann unterschieden werden, ob eine Anforderung verpflichtend umgesetzt werden muss, ob die Anforderung den Wunsch eines Stakeholders ausdrückt und nicht verpflichtend ist oder ob eine Anforderung in der Zukunft verpflichtend sein wird und entsprechend in der Planung berücksichtigt werden muss. Im zweiten Schritt folgt die Festlegung eines Prozesswortes, das die geforderte Funktion des Systems definiert.

[3] https://www.sophist.de, zuletzt abgerufen am 7.11.2022

Im dritten Schritt muss die Art der Funktion festgelegt werden, wobei zwischen den drei Funktionsarten *selbstständige Systemaktivität*, *Benutzerinteraktion* und *Schnittstellenanforderung* unterschieden wird. Schritt vier umfasst die Definition des Objekts, das in direktem Zusammenhang mit dem Prozesswort steht. Zuletzt erfolgt durch den optionalen Schritt 5 die Festlegung einer Bedingung [Sop16, S. 10 ff.].

Einen weiteren Ansatz über die Verwendung von kontrollierter natürlicher Sprache zeigen FOCKEL & HOLTMANN [FH14]. Hier ist ausgehend von strukturierter natürlicher Sprache eine bidirektionale Transformation der textuellen Anforderungen in Anforderungsmodelle beschrieben, um mit Hilfe eines Werkzeugs [FH15] bei der Spezifikation der Anforderungen zu unterstützen und die Verständlichkeit der Anforderungen zu erhöhen.

Diese und vergleichbare Ansätze haben den Vorteil, dass die resultierenden Spezifikationen allgemein verständlich bleiben. Ebenso erleichtert die Verwendung von strukturierter natürlicher Sprache und Satzschablonen eine nachgelagerte Transformation der Anforderungen in formale Modelle für eine automatisierte Anforderungsanalyse [YBL11]. Jedoch entsteht durch die Strukturierung der natürlichen Sprache ein zusätzlicher Aufwand. Hinzu kommt, dass die Verwendung von einheitlichen Schablonen schwer einzuhalten ist, wenn Anforderungen über Unternehmensgrenzen hinweg und zwischen unterschiedlichen Disziplinen ausgetauscht werden [LTK19].

3.2.3 Modellbasierte Spezifikation und Analyse von Anforderungen im Automobilbereich

Eine weitere Steigerung der Formalisierung erfolgt innerhalb des RE über die Modellierung von Anforderungen. In dieser Arbeit werden Modelle als verkürzte Abbildungen von Originalen verstanden, die zu bestimmten Zeitpunkten für spezifische Aufgaben verwendet werden. Diese Sichtweise basiert auf dem Modellbegriff nach STACHOWIAK, wonach ein Modell durch die drei folgenden Hauptmerkmale charakterisiert werden kann [Sta73, S.131 f.]:

- **Abbildungsmerkmal**: *"Modelle sind stets Modelle von etwas, nämlich Abbildungen, Repräsentationen natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können."*
- **Verkürzungsmerkmal**: *"Modelle erfassen im allgemeinen nicht alle Attribute des durch sie repräsentierten Originals, sondern nur solche, die den jeweiligen Modellerschaffern und/oder Modellbenutzern relevant scheinen."*
- **Pragmatisches Merkmal**: Das Modell als verkürzte Abbildung des Originals dient zu einem bestimmten Zeitpunkt der Erfüllung einer spezifischen Aufgabe.

Demnach kann bspw. in der frühen Phase der Systementwicklung innerhalb des RE die Definition von Entwicklungszielen durch Modelle unterstützt werden (z.B. durch die i* [Yu95; DFH16] oder KAOS [Van01; Van04] Modellierungssprache). Im weiteren Verlauf

der Entwicklung kann sich der Fokus von der initialen Zielmodellierung auf die Modellierung des zu realisierenden Systemverhaltens verschieben (z.B. über die Verwendung von *Message Sequence Charts* (MSCs) [ITU]). Hier wird deutlich, dass zu unterschiedlichen Entwicklungszeitpunkten im RE verschiedene Modelle die Entwicklungsarbeit unterstützen, wobei unterschiedliche Inhalte der jeweiligen Originale abgebildet werden. So steht in der frühen Phase die Interaktion zwischen Stakeholdern, Unternehmen und Systemen im Vordergrund, was im weiteren Verlauf durch konkretisiertes Systemverhalten einzelner Systeme zu beschreiben ist.

Im Kontext dieser Arbeit steht die Modellierung von *funktionalen Systemanforderungen* im Vordergrund. Demnach wird hier die textuelle Spezifikation von Anforderungen in natürlicher Sprache als Original gesehen, das mit Hilfe von Modellen in eine formale Darstellung zu überführen ist, um Entwickler bei der Analyse der Anforderungen zu unterstützen. Ein vereinfachtes Beispiel für die Überführung von textuellen Anforderungen in ein Modell ist in Bild 3-3 dargestellt. Auf der linken Seite sind Anforderungen aufgelistet, die ein Systemverhalten spezifizieren, auf der rechten Seite ist das Verhalten durch ein Modell abgebildet. Der Argumentation des IREB folgend, stellt das abgeleitete Modell die Anforderungen in einer strukturierten und verständlichen Weise dar [CHQ+16].

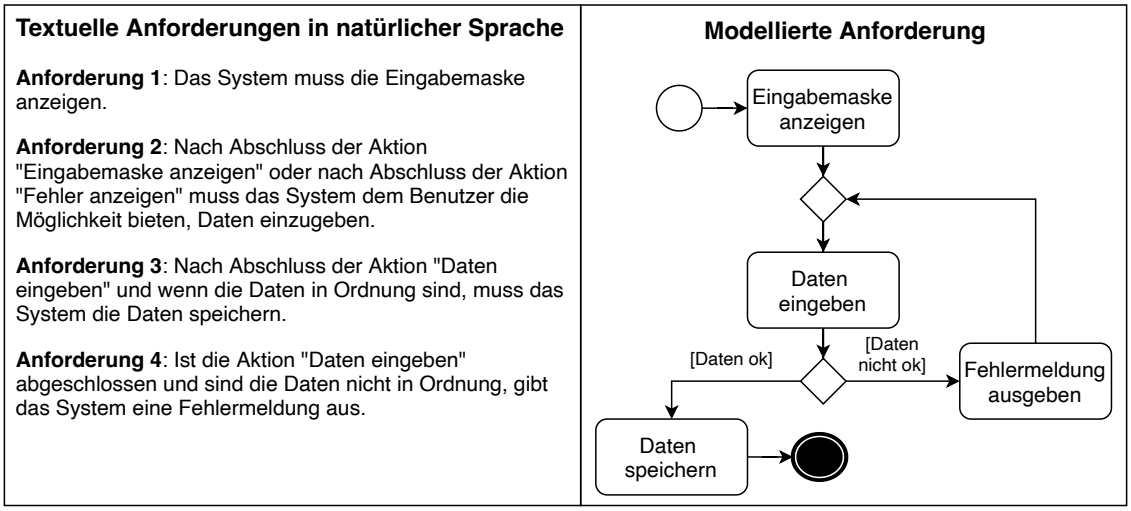


Bild 3-3: Vereinfachte Gegenüberstellung von textuellen Anforderungen und eine entsprechende modellbasierte Darstellung [CHQ+16]

LIEBEL et al. untersuchen die Anwendung von Modellen im RE und zeigen Potentiale und Probleme für den Automobilbereich auf [LTK19]. Die Ergebnisse der Studie bestätigen, dass in den beteiligten Unternehmen die Spezifikation von Anforderungen überwiegend textuell und in natürlicher Sprache erfolgt, obwohl der Einsatz von Modellen im RE von den befragten Personen als eine aussichtsreiche Verbesserung angesehen wird, insbesondere bezogen auf die Verifikation und Validierung von Anforderungen.

Im Gegensatz zu verwandten Arbeiten, die textuelle Anforderungen als unzureichend [HP00] oder als nicht zufriedenstellend [STP11] beschreiben, geben die Befragten in der

von LIEBEL et al. durchgeführten Studie an, dass auf höheren Abstraktionsstufen die textuelle Spezifikation als geeignet erscheint, da so der Aufwand für die Spezifikation und das Management der Anforderung verringert werden kann. Als ein weiterer wichtiger Grund für die textuelle Spezifikation von Anforderungen wird angegeben, dass hierdurch Unsicherheit besser ausgedrückt werden kann [LTK19], was insbesondere in der frühen Entwicklungsphase und auf höheren Abstraktionsstufen hilfreich ist.

3.2.4 Herausforderungen für die Spezifikation und Analyse von Anforderungen

In den vorangegangenen Kapiteln wird deutlich, dass die Spezifikation von Anforderungen in natürlicher Sprache in der Automobilindustrie einen hohen Stellenwert hat, insbesondere für den rechtsverbindlichen Austausch von Anforderungen zwischen den Unternehmen und dem damit verbundenen Management der Anforderungen. Gleichzeitig wird auch deutlich, dass der Einsatz von modellbasierten Ansätzen im RE von den Unternehmen zwar als geeignetes Mittel zur Analyse von komplexen Spezifikationen gesehen wird, diese jedoch in der Breite nicht angewendet werden.

Da sowohl die informale als auch formale Spezifikation von Anforderungen einen Mehrwert liefern, besteht die Herausforderung in einer geeigneten Kombination beider Ansätze. Der Argumentation von LIEBEL et al. folgend, wird eine ausschließliche Konzentration auf die Formalisierung und auf Analysen, die auf formalen Modellen beruhen, den aktuellen Bedürfnissen der Industrie nicht gerecht [LTK19]. Demnach wird von modellbasierten Ansätzen erwartet, dass die Modelle leicht zugänglich und auch für komplexe Systeme anwendbar und wartbar sind. Ebenso ist die Überprüfbarkeit der Modelle, z.B. im Rahmen von Reviews zu bestimmten Zeitpunkten im Entwicklungsprojekt ein wichtiges Kriterium für die Akzeptanz in den Industrieunternehmen. Weitere entscheidende Punkte für modellbasierte Ansätze im RE sind die Integrierbarkeit der verwendeten Methoden und Werkzeuge in die bestehenden IT-Infrastrukturen, sowie die Generierung von Artefakten aus den erstellen Modellen, bspw. zur Dokumentation von Systemanforderungen [LK23].

Hinzu kommt, dass insbesondere im Kontext des ASE die Berücksichtigung von Iterationen in der Analyse und Spezifikation von Anforderungen erforderlich ist. Unabhängig davon, ob Anforderungen textuell in natürlicher Sprache oder modellbasiert spezifiziert werden, besteht die Notwendigkeit, Entwicklungsziele und Anforderungen kontinuierlich abzustimmen und zu analysieren [NL18], da Anforderungsspezifikationen in der frühen Entwicklungsphase häufig unvollständig sind [FWK+17] und erst durch das Schließen von Wissenslücken zur Entwicklungszeit vervollständigt werden können.

3.3 Entwurf und Spezifikation von Tests

Das zweite Handlungsfeld adressiert den Entwurf und die Spezifikation von Tests. Die Tätigkeiten zum Entwurf und der Spezifikation von Tests sind die Basis für V&V und Teil der Systementwicklung. Dabei hat die V&V innerhalb der Automobilindustrie einen

hohen Stellenwert, da diese maßgeblich die Qualität und Sicherheit der zu entwickelnden Systeme beeinflusst und einen hohen Anteil an den gesamten Entwicklungskosten besitzt [ABK+16].

3.3.1 Verifikation und Validierung (V&V)

Die *Verifikation* wird in dieser Arbeit als eine Überprüfung verstanden, ob das entwickelte System mit der Spezifikation des Systems übereinstimmt. Unter *Validierung* ist *„(...) die Prüfung zu verstehen, ob das Produkt für seinen Einsatzzweck geeignet ist bzw. den gewünschten Wert erzielt [VDI].“* Die Validierung berücksichtigt demnach insbesondere auch die Sicht des Fachexperten und des Anwenders [ABK+16].

Das *Testen* ist dabei ein elementarer Bestandteil, sowohl für die Verifikation als auch für die Validierung (vgl. [ABK+16] und [ISOg]). Bild 3-4 verdeutlicht diesen Zusammenhang und strukturiert die für das Testen im Rahmen der V&V zum Einsatz kommenden Methoden nach dem internationalen Standard *ISO/IEC/IEEE 29119* in die vier Felder: 1) *statisches Testen*, 2) *dynamisches Testen*, 3) *Demonstration* und 4) *V&V Analyse* [ISOg]. Hierbei wird

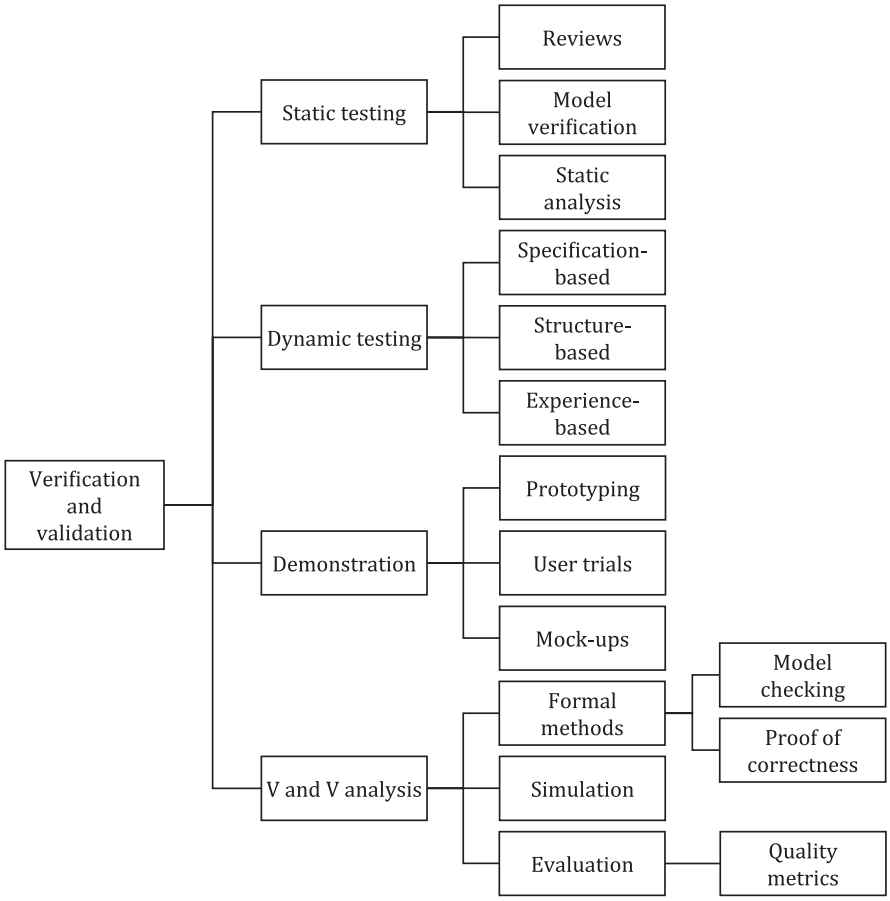


Bild 3-4: Methoden für die V&V nach ISO/ IEC/IEEE 29119-1 [ISOg]

deutlich, dass eine Vielzahl von Methoden für das Testen existieren und eine eindeutige Zuweisung einer Methode zur Unterstützung einer Verifikations- oder Validierungstätigkeit

nicht immer möglich ist. Wird bspw. in einem Test das Systemverhalten simuliert, unterstützt die Analyse der Ergebnisse die Bewertung, ob die Systemspezifikation eingehalten wird (Verifikation). Ebenso ermöglicht die Interpretation und Konsolidierung der Ergebnisse eine Aussage darüber, ob das zu entwickelnde System seinen Einsatzzweck erfüllen kann (Validierung). In der Literatur und in der Praxis führt diese Überschneidung zu einer sehr heterogenen Verwendung der Begriffe Validierung und Verifikation [ABK+16].

3.3.2 Spezifikation von Tests in der Automobilindustrie

Wie zu Beginn bereits angedeutet (Kapitel 3.1.2), sind in Entwicklungsprozessen in der Automobilindustrie Standards zu befolgen, die eine Entwicklung von qualitativ hochwertigen und sicheren Systemen adressieren [Web09; Hil12]. Von besonderer Bedeutung sind hier u.a. ASPICE® [VDA] sowie Standards, die eine funktionale Sicherheit [ISOa] und Cybersicherheit [ISOb] adressieren. Diese Standards fokussieren unterschiedliche Schwerpunkte, haben jedoch gemeinsam, dass auf der Makro-Ebene Testprozesse auf Basis des V-Modells implementiert sein müssen. Ausgehend von diesen Vorgaben veranschaulicht Bild 3-5 die Beziehungen zwischen den einzelnen Prozessen. Hier wird deutlich, dass zu jedem Prozess auf der linken Seite ein entsprechender Testprozess auf der rechten Seite des V-Modells vorgesehen ist [VDA, S. 127].

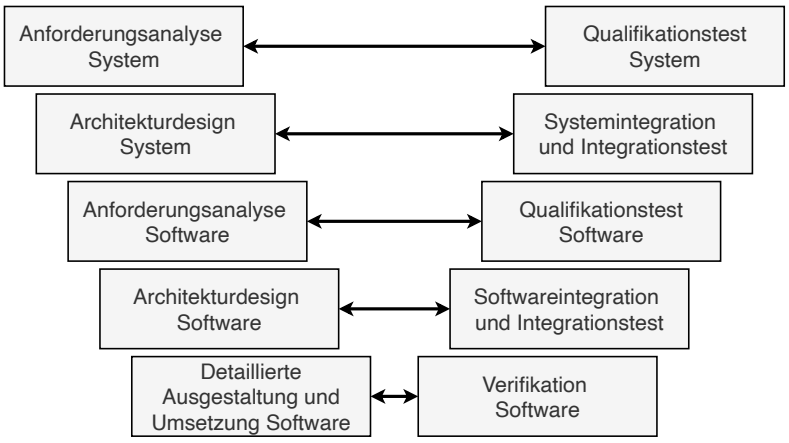


Bild 3-5: ASPICE Engineering-Referenzprozesse strukturiert nach dem V-Modell mit einer eindeutigen Zuordnung der Testprozesse (nach [VDA, S. 127]).

Dieser Darstellung folgend können die Resultate der Prozesse auf der linken Seite als *Testbasis* für die Erstellung von *Testspezifikationen* gesehen werden. Bild 3-6 verdeutlicht diese Abhängigkeiten, wobei die Pfeile in der Abbildung angeben, zwischen welchen Resultaten eine Konsistenz und bidirektionale Traceablity bestehen muss. So ist bspw. der Nachweis zu erbringen, dass Stakeholder-Anforderungen durch Systemanforderungen adressiert werden, und diese Systemanforderungen wiederum durch eine Systemarchitektur realisiert werden. Diese Resultate stehen wiederum in direkter Beziehung zu den Testspezifikationen auf unterschiedlichen Ebenen, wobei jede Testspezifikation *Testfälle* als Basis für die Testdurchführung beinhaltet. Die Ausführung dieser Testfälle führt wiederum zu

Testergebnissen, die in einer direkten Beziehung zum Testfall stehen. Hieraus folgt, dass auch für die Testfälle und Testergebnisse eine Konsistenz und bidirektionale Traceability nachzuweisen ist.

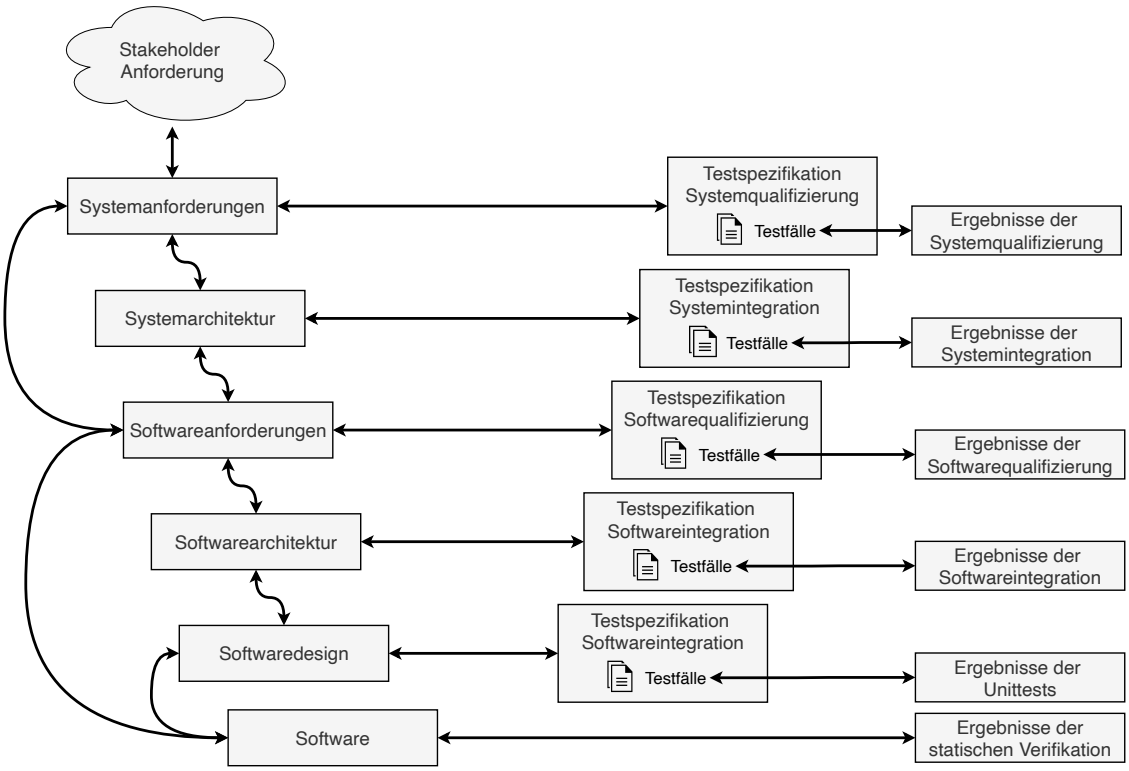


Bild 3-6: Beziehungen zwischen den Resultaten bei einer erfolgreichen Durchführung der Prozesse (nach [VDA, S. 128]).

Neben den Beziehungen zwischen den einzelnen Resultaten, sowohl auf der linken als auch auf der rechten Seite des V-Modells, erfordern die Standards die Entwicklung von übergeordneten *Teststrategien* und *Testplänen* (vgl. [VDA; ISOg]), um die einzelnen Testaktivitäten verteilt über unterschiedliche Disziplinen miteinander zu verzahnen und so im Entwicklungsprojekt ein systematisches Testen zu ermöglichen.

Die Basis für eine einheitliche Interpretation der zuvor genannten Resultate der einzelnen Testprozesse in den Unternehmen ist u.a. das ISTQB® (International Software Testing Qualifications Board) Glossar[4] (vgl. [SLS11, S. 243 ff.]). Hiernach sind die für diese Arbeit wesentlichen Begriffe im folgenden aufgeführt:

Test: *„Eine Menge von einem oder mehreren Testfällen.“*

Testfall: *„Eine Menge von Vorbedingungen, Eingaben, Aktionen (falls anwendbar), erwarteten Ergebnissen und Nachbedingungen, welche auf Basis von Testbedingungen entwickelt wurden.“*

[4] https://glossary.istqb.org/ zuletzt abgerufen am 1.9.2022

Testen: *„Der Prozess, der aus allen statischen und dynamischen Lebenszyklusaktivitäten besteht, die sich mit der Planung, Vorbereitung und Bewertung einer Komponente oder eines Systems und zugehörigen Arbeitsergebnissen befassen, um festzustellen, ob sie festgelegte Anforderungen erfüllen, für den Zweck geeignet sind, sowie um etwaige Fehlerzustände zu finden.“*

Teststrategie: *„Dokumentation, die an der Testrichtlinie[5] ausgerichtet ist, und welche die allgemeinen Anforderungen für das Testen und Details für die Durchführung von Tests in einer Organisation beschreibt.“*

Testplan: *„Eine Liste von Aktivitäten, Aufgaben oder Ereignissen des Testprozesses, mit Angabe ihrer geplanten Anfangs- und Endtermine, sowie ihrer gegenseitigen Abhängigkeiten.“*

Testbasis: *„Alle Informationen, die als Basis für die Testanalyse und den Testentwurf verwendet werden können.“*

Testobjekt: *„Das zu testende Arbeitsergebnis.“*

Testspezifikation: *„Dokument, das eine Menge von Testfällen für ein Testobjekt spezifiziert (inkl. Testdaten und Vor-/Nachbedingung), bei dem die Testfälle jeweils Ziele, Eingaben, vorausgesagte Ergebnisse und Vorbedingungen für die Ausführung enthalten.“*

Hierbei ist die Testspezifikation als ein zentrales Ergebnis der Testanalyse und des Testdesigns von besonderer Bedeutung für die Testdurchführung. Bild 3-7 veranschaulicht die Abhängigkeiten zwischen der Erstellung und Verwendung der Testspezifikation in Entwicklungsprojekten im Automobilbereich. Auf der einen Seite wird die Testspezifikation durch Test-Designer erstellt, auf der anderen Seite nutzen Tester die Testspezifikation als Basis für die Testautomatisierung oder auch zur Durchführung von manuellen Tests. Dabei können die Testspezifikationen auf unterschiedlichen Testumgebungen realisiert werden, wie bspw. Hardware-in-the-Loop (HIL) Systeme, Fahrzeugprototypen oder auch Testumgebungen als Teil eines Produktionssystems. Auf diese Weise dient die Testspezifikation als zentrales Dokument für die verteilte Umsetzung der notwendigen Testaktivitäten. Neben der verteilten Umsetzung erfolgt auch die Erstellung der Spezifikation durch unterschiedliche Rollen in den Organisationen. Üblicherweise ist ein Testverantwortlicher innerhalb eines Unternehmens für die Erstellung und Pflege der Testspezifikation zuständig, wobei für die Ausarbeitung der Spezifikation sowohl interne als auch externe Entwickler beauftragt werden können [JTH20].

Somit ist die Testspezifikation das Ergebnis eines verteilten Testanalyse und -Designprozesses und dient als Basis für die verteilte Durchführung von heterogenen Testaktivitäten. Dabei geben die zuvor genannten Standards vor, dass die Testspezifikation als Prozessergebnis vorliegen und definierte Anforderungen erfüllen muss, wie bspw. die Traceability von Anforderungen, Testfällen und Testergebnissen (s. [VDA; ISOg]). Die Standards definieren

---

[5] Abstrakte Beschreibung von Testprinzipien, -ansätzen und -zielen einer Organisation.

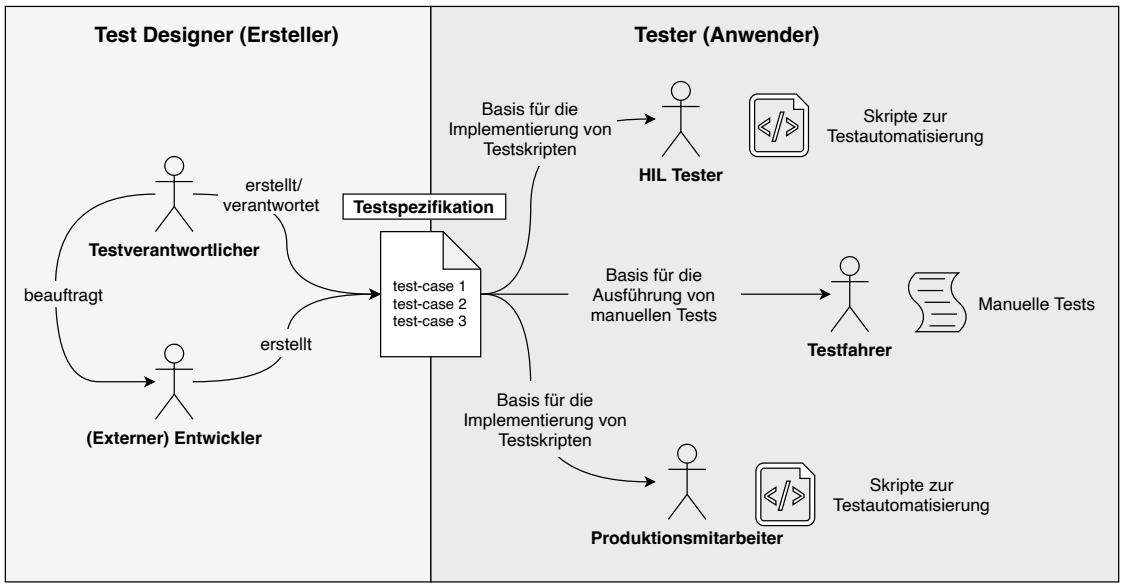


Bild 3-7: Exemplarische Darstellung der Beziehungen zwischen den Erstellern von Testspezifikationen und den Anwendern, die die Spezifikationen zur Testautomatisierung oder zur Durchführung von manuellen Tests verwenden (nach [JTH20]).

jedoch nicht, wie die Testspezifikationen zu erstellen sind. Aus diesem Grund beschreiben SPILLNER & LINZ einen fundamentalen Testprozess zur Strukturierung der Testaktivitäten, der als Teilprozess in einem übergeordnetem Vorgehensmodell verstanden werden kann [SLS11, S. 20 f.]. Bild 3-8 veranschaulicht diesen Testprozess, der aus den Phasen *Testplanung*, *Testanalyse und -design*, *Testimplementierung und -ausführung*, *Auswertung und Bericht* und *Testabschluss* besteht. Das *Test Management* wird hier als begleitende und kontinuierliche Steuerung der Testaktivitäten verstanden, wobei der Testplan und die zur Verfügung stehenden Ressourcen Ausgangspunkt für die Steuerung der Testaktivitäten sind (vgl. [SLS11, S. 21]). Der Fokus dieser Arbeit liegt auf der Erstellung von Testspezifikationen als Resultat der Testanalyse und -designphase auf der Basis der vorangegangenen Testplanung.

3.3.3 Spezifikation von Testfällen in natürlicher Sprache

In den vorangegangenen Kapiteln wird deutlich, dass die Testspezifikation aus unterschiedlichen Gesichtspunkten ein wichtiges Resultat in Entwicklungsprojekten innerhalb der Automobilindustrie ist. Die Testspezifikation ist eine Kommunikationsbasis für die am Test Management, dem Testdesign und der Testimplementierung beteiligten Rollen. Zusätzlich ist die Testspezifikation eine Vertragsgrundlage, wenn die einzelnen Aktivitäten innerhalb des Testprozesses durch unterschiedliche Unternehmen in einer Entwicklungspartnerschaft ausgeführt werden. Diese Rahmenbedingungen führen dazu, dass Testfälle zu einem großen Anteil in natürlicher Sprache spezifiziert werden [Juh21, S. 23]. Wie auch für die Spezifikation von Anforderungen, entsteht so eine allgemein verständliche Testspezifikation, die

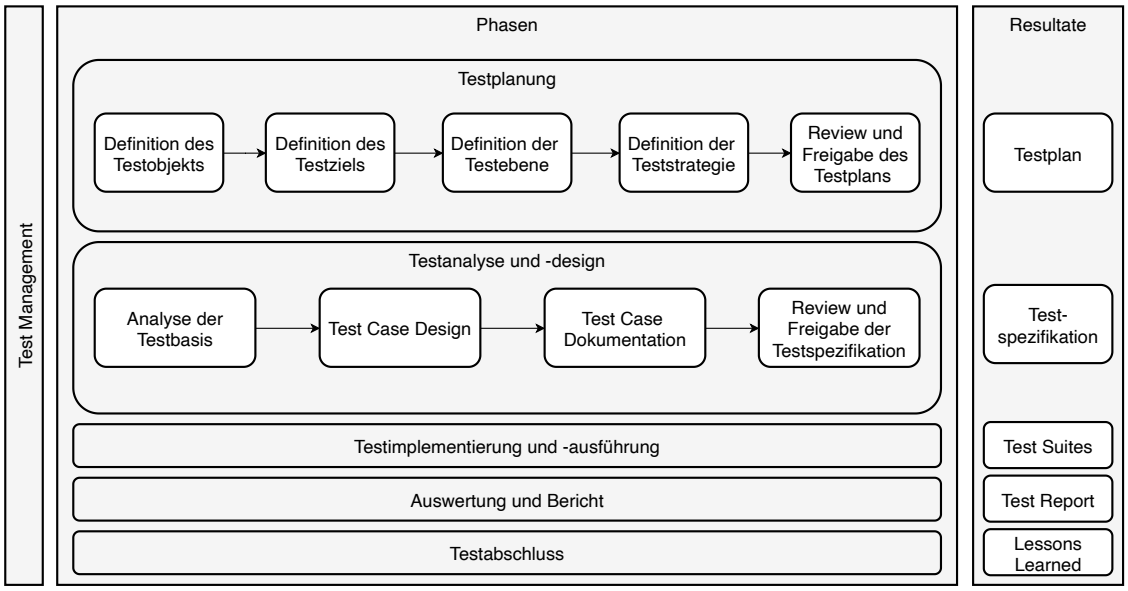


Bild 3-8: Vereinfachte Darstellung des fundamentalen Testprozesses nach [JTH20] auf Basis von [SLS11, S. 21].

niederschwellig editierbar ist und leicht zwischen den Entwicklungspartnern ausgetauscht werden kann.

Bild 3-9 veranschaulicht eine in der Automobilindustrie übliche Form für die Spezifikation von Testfällen nach den Vorgaben in der ISO/IEC/IEEE 29119 [ISOg]. Hier erfolgt eine Unterteilung des Testfalls in einzelne Testschritte, die durch mehrere Attribute beschrieben sind. Jeder Eintrag besitzt eine eindeutige Identifikationsnummer, eine Typendefinition, sowie einen beschreibenden Text. Der eigentliche Testfall ist definiert durch eine Vorbedingung, eine durchzuführende Aktion, ein erwartetes Ergebnis und eine Nachbedingung. Ergänzend wird eine Testumgebung angegeben, auf der die einzelnen Testschritte auszuführen sind.

Obwohl die Prozesse für die Erstellung der Spezifikationen, sowie deren Aufbau definiert und in den Testprozessen der Unternehmen implementiert sind, ist die Spezifikation der *Testfälle* wenig systematisch, und die Form der Testfallbeschreibung hängt stark von dem jeweiligen Entwickler und der Testebene ab [UPL12]. So sind die Spezifikationen zwar niederschwellig editierbar, die wenig systematisierte Vorgehensweise zur Testfallspezifikation führt jedoch dazu, dass die Spezifikationen schlecht wartbar, schwer nachvollziehbar und ineffizient ausführbar sind [HJE+13]. Die Verwendung von natürlicher Sprache für die Spezifikation von Testfällen führt somit zu sehr heterogenen Spezifikationen in unterschiedlicher und schwer zu bewertender Qualität [JTH20]. Für eine Identifikation von Inhalten geringer Qualität geben HAUPTMANN et al. die in Tabelle 3-2 aufgeführten Anzeichen an.

Anzeichen	**Beschreibung**
Fest kodierte Werte	Ein Testfall enthält fest kodierte Werte oder Zeichenketten, die für die Ausführung des Testfalls relevant sind (z.B. Testdaten). Diese können auch in mehreren Testfällen vorkommen. Bei der Pflege von Testfällen ist es oft schwierig, (alle) von einer Änderung betroffenen Werte zu finden und anzupassen.
Lange Testschritte	Ein Testschritt, der aus in natürlicher Sprache beschriebenen Aktionen und erwarteten Ergebnissen besteht, ist sehr lang, wodurch die Verständlichkeit beeinträchtigt ist.
Bedingte Tests	Testfälle enthalten bedingte Logik, die in natürlicher Sprache beschrieben wird. Dies beeinträchtigt die Verständlichkeit und die daraus resultierende Komplexität erschwert die Überprüfung der Korrektheit des Testfalls.
Schlecht strukturierte Testsuiten	Testfälle, die dieselbe Funktion testen, werden nicht in einem Abschnitt in einer Testsuite zusammengefasst. Dadurch wird es schwieriger, den Zweck eines Tests zu verstehen, bestimmte Testfälle für die Ausführung auszuwählen, und die Wartbarkeit wird umständlicher, da die von einer Änderung betroffenen Testfälle ermittelt werden müssen.
Test Klone	Testfälle enthalten Duplikate, z. B. identische Aktionen oder erwartete Ergebnisse. Dies beeinträchtigt die Verständlichkeit, da ähnliche Testfälle schwieriger zu unterscheiden sind und die Wartbarkeit, da im Falle einer Änderung mehrere Stellen betroffen sind.
Mehrdeutige Tests	Dem Testfall fehlen Informationen, die einen Interpretationsspielraum bei der Ausführung des Testfalls zulassen. Dies wirkt sich auf die Nachvollziehbarkeit und die Ausführung aus, da fehlende Informationen zu einem nicht deterministischen Testverhalten beitragen, das zu unterschiedlichen Testergebnissen führt.
Uneinheitliche Formulierung	Disziplinspezifische Konzepte und zugehöriges Vokabular werden in den Testfallbeschreibungen nicht einheitlich verwendet, was die Verständlichkeit beeinträchtigt.

Tabelle 3-2: Anzeichen für geringe Qualität von Testfallspezifikationen in natürlicher Sprache [HJE+13] (vgl. [Juh21, S. 48])

ID	Object Type	Object Text	Precondition	Action	Expected Result	Postcondition	Test Platform
WL-TS-145	*test case*	**Window manual opening**	- ISw_Stat = IGN_ON - All windows closed - Key inside			- None	Cluster HiL
WL-TS-146	*test step*	Press and hold driver's MANUAL_OPN		Press and hold driver's MANUAL_OPN window switch for each window	- Button pressed is detected and send on bus W_Adj_FL = 1 (MANUAL_OPN) W_Adj_FR = 1 (MANUAL_OPN) W_Adj_RL = 1 (MANUAL_OPN) W_Adj_RR = 1 (MANUAL_OPN) - Key Inside is detected KG_Key_Rs_ST3 = 1 (INSIDE) - Windows start to open W_FL_Pos_Stat = 5 (WINDOW_RUNNING) W_FR_Pos_Stat = 5 (WINDOW_RUNNING) W_RL_Pos_Stat = 5 (WINDOW_RUNNING) W_RR_Pos_Stat = 5 (WINDOW_RUNNING)		
WL-TS-147	*test step*	Release switch		Release the MANUAL_OPN window switch for each window	- Button released is detected and send on bus W_Adj_FL = 0 (NPSD) W_Adj_FR = 0 (NPSD) W_Adj_RL = 0 (NPSD) W_Adj_RR = 0 (NPSD) - All moving windows stop immediately W_FL_Pos_Stat = 0 (INTERMEDIATE) W_FR_Pos_Stat = 0 (INTERMEDIATE) W_RL_Pos_Stat = 0 (INTERMEDIATE) W_RR_Pos_Stat = 0 (INTERMEDIATE)		

Bild 3-9: Beispiel für die Spezifikation von Testfällen [Juh21, S. 27]

3.3.4 Herausforderungen für die Spezifikation von Tests

Ebenso wie die Spezifikation von Anforderungen im RE, erfolgt innerhalb der Automobilindustrie im Bereich V&V die Spezifikation von Testfällen häufig textuell und in natürlicher Sprache, wobei die Gründe hierfür ähnlich sind. Auch hier besteht der Bedarf Testfälle so zu spezifizieren, dass sie allgemein verständlich sind und durch unterschiedliche Rollen für die Automatisierung von Testfällen oder für die manuelle Testdurchführung verwendet werden können (vgl. Bild 3-7). Dabei besteht die Herausforderung, präzise, vollständige und konsistente Spezifikationen zu erstellen, die dennoch niederschwellig durch unterschiedliche Rollen in einer Organisation editiert werden können.

Eine weitere Herausforderung ist die explizite Modellierung von Validierungszielen. Wie oben deutlich wird, liegt bei der Entwicklung von Testspezifikationen der Fokus auf dem anforderungsbasierten Testen [JTH20], wobei die Traceability zwischen den Anforderungen, den Testfällen und den jeweiligen Testergebnissen einen hohen Stellenwert hat [BRB+13] [DAR+21, S. 57] (s. Kapitel 3.3.2). Somit liegt bei der Erstellung der Testspezifikationen der Fokus auf der Verifikation, um den Nachweis zu erbringen, dass das zu entwickelnde System mit den Elementen der Anforderungsspezifikation und der Systemarchitektur übereinstimmt. Eine alleinige Fokussierung auf die Verifikation ist jedoch für die erfolgreiche Systementwicklung nicht ausreichend. Die Aspekte der Validierung, also der Nachweis, ob das Produkt für seinen Einsatzzweck geeignet ist bzw. den gewünschten Wert erzielt, sind ebenfalls bei der Erstellung der Testspezifikation zu berücksichtigen [ABK+16].

Die präzise Spezifikation von Validierungszielen als Teil der Testspezifikation ist jedoch insbesondere für die Entwicklung von *Advanced Systems* herausfordernd [DAR+21, S. 52 f.]. Der hohe Vernetzungsgrad aktueller und zukünftiger Systeme führt zu komplexen Systemverbünden, wie zu Beginn anhand eines *Smart-Charging* Beispiels verdeutlicht

(s. Bild 1-1). Hieraus resultiert die Herausforderung, die Validierungsziele für einzelne Systeme abzuleiten und mit den übergeordneten Zielen des Systemverbundes in Einklang zu bringen [Hon13; AMY+18]. Für die Entwicklung von Testspezifikationen für einzelne Systeme ist ein Problembewusstsein erforderlich, das auch die vernetzten Systeme und die resultierende Gesamtfunktionalität einschließt.

3.4 Externalisierung und Wiederverwenden von Lösungswissen

In den beiden vorangegangenen Kapiteln 3.2 und 3.3 wird deutlich, dass in der Automobilindustrie sowohl die Spezifikation von Anforderungen als auch die Spezifikation von Tests überwiegend textuell erfolgt und auf manuellen Analysen basiert. Hierdurch entstehen bei der Spezifikation hohe Freiheitsgrade, und das Wissen zur Erstellung und Interpretation der Spezifikationen ist in der Regel personengebunden und entsteht durch einen kontinuierlichen Problemlösungsprozess in der Produktentwicklung. POLANY bezeichnet dieses personengebundene Wissen als *implizites Wissen* [Pol85]. Im Kontext dieser Arbeit ist dabei das implizite Wissen aus Sicht des Systemarchitekten und des Test-Designers relevant. Hier besteht das Problem, dass die Stakeholder-Anforderungen in Systemanforderungen zu überführen sind (Systemarchitekt) und die erwarteten Systemeingenschaften verifiziert und validiert werden müssen, um die erfolgreiche Umsetzung der Systemanforderungen nachzuweisen (Test-Designer). Die dafür jeweils notwendigen Anforderungs- und Testspezifikationen werden hier als Lösung verstanden. Das Wissen, wie diese Spezifikationen effektiv und zielführend auf Basis von textuellen Stakeholder-Anforderungen erstellt werden können, kann als Lösungswissen beschrieben werden. Nach PROBST et al. kann dieses Wissen wie folgt definiert werden:

> *"[Lösungs]wissen bezeichnet die Gesamtheit der Kenntnisse und Fähigkeiten, die Individuen zur Lösung von Problemen einsetzen. Dies umfasst sowohl theoretische Erkenntnisse als auch praktische Alltagsregeln und Handlungsanweisungen. Wissen stützt sich auf Daten und Informationen, ist im Gegensatz zu diesen jedoch immer an Personen gebunden. Es wird von Individuen konstruiert und repräsentiert deren Erwartungen über Ursache-Wirkungs-Zusammenhänge [PRR10, S. 23]."*

Dabei ist von großer Bedeutung, wie implizites Lösungswissen für die Anforderungs- und Testspezifikation externalisiert und so innerhalb einer Organisation verfügbar wird.

3.4.1 Wissenstransfer in Unternehmen

Die Unternehmen in der Automobilindustrie und verwandten Branchen nutzen unterschiedliche Ansätze, um das implizite Wissen einzelner Experten innerhalb einer Organisation verfügbar zu machen. Hierbei kommen oft dokumentenbasierte Ansätze wie Projekthandbücher, Templates, Methodenbeschreibungen oder *Lessons Learned* Programme zum Einsatz. Ein großer Teil des Wissens bleibt dennoch implizites Wissen einzelner Experten (vgl.

[Clo06, S. 6]). Das führt dazu, dass diese Experten innerhalb der Unternehmen zu einem Engpass für die Systementwicklung werden können, da die Expertise einzelner Individuen für die erfolgreiche Systementwicklung notwendig ist [Hol05] [PRR10, S. 19].

Den Übergang von implizitem Expertenwissen zu explizitem Wissen in einer Organisation veranschaulichen NONAKA & TAKEUCHI mit Hilfe des in Bild 3-10 dargestellten SECI-Modells (Socialisation, Externalization, Combination, Internalization) [NT97, S. 78ff.]. Der Aufbau von Wissen in einer Organisation wird hier durch die Phasen Sozia-

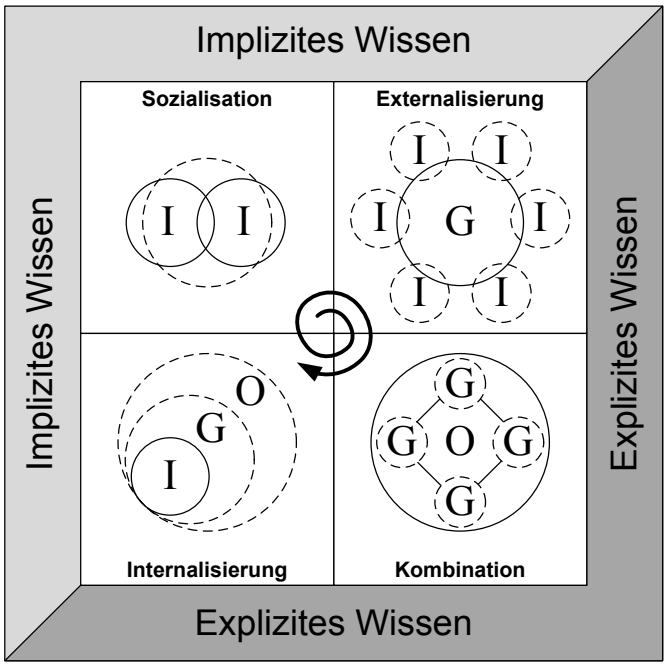


Bild 3-10: SECI Modell [Dum10, S. 129] nach [NT97, S. 84]

lisation, Externalisierung, Kombination und Internalisierung beschrieben, die in einem kontinuierlichem Prozess durchlaufen werden. Wie in Bild 3-10 skizziert, beinhaltet die *Sozialisation* den Austausch von implizitem Wissen zwischen einzelnen Individuen, z.B. durch gemeinsam gemachte Erfahrungen, Beobachtung und Nachahmung. Hierbei erfolgt der Wissensaustausch überwiegend stillschweigend. Die *Externalisierung* bezeichnet den Übergang vom impliziten Wissen einzelner Individuen zu explizitem Wissen, das allen Individuen einer Gruppe zur Verfügung steht. Hierbei ist es notwendig, komplexe Sachverhalte allgemein verständlich zu beschreiben, bspw. in Form von Szenarien oder Analogien zu in der jeweiligen Gruppe bekannten Sachverhalten. Die *Kombination* bezeichnet die Phase, in der das durch die Externalisierung gewonnene Wissen zwischen unterschiedlichen Gruppen ausgetauscht und kombiniert wird. Das Wissen einzelner Gruppen einer Organisation, bspw. örtlich verteile Fachabteilungen, wird so zugänglich und erweitert. Ausgehend vom explizit verfügbaren Wissen und der Kombination des Wissens in der Organisation umfasst die *Internalisierung* das Aufnehmen, Verstehen und Lernen des expliziten Wissens, wodurch dieses wiederum zu implizitem Wissens einzelner Individuen wird ([NT97, S. 84 f.], vgl. [Ana15, S. 29f.] und [Dum10, S. 128f.]).

Ausgehend vom allgemeinen Wissenstransfer innerhalb einer Organisation soll im folgenden die Externalisierung und Wiederverwendung von *Lösungswissen für die Anforderungs- und Testspezifikation* untersucht werden. GAROUSI et al. zeigen, dass es einen starken Bezug zwischen dem Wissen einzelner Entwickler und der Qualität der entwickelten Tests gibt [GFK+17]. Hinzu kommt, dass die Anforderungsspezifikation als eine wichtige Testbasis für den Testentwurf dient und die Qualität der Tests von der Qualität der Anforderungsspezifikation abhängt [BRB+13] [KPM13] [GFK+17] [JTH20]. Da die beiden Bereiche stark miteinander in Beziehung stehen, ist im Kontext dieser Arbeit von Bedeutung, wie das personengebundene Wissen für die Spezifikation von Anforderungen und Tests verfügbar gemacht, dokumentiert und wiederverwendet werden kann, sodass beide Bereiche von dem identifizierten Lösungswissen profitieren.

3.4.2 Lösungswissen für die Anforderungsspezifikation

FRANCH et al. zeigen, dass die Wiederverwendung von Anforderungen in der industriellen Praxis eine große Relevanz hat [FPQ20]. Ebenso ist die Wiederverwendung von Anforderungen ein relevanter Forschungszweig innerhalb des RE [CA09]. PALOMARES et al. zeigen jedoch, dass eine große Lücke von möglichen Ansätzen aus der Forschung und der Übernahme dieser Ansätze in die industrielle Praxis besteht [PQF17].

Obwohl anhand einiger Beispiele eine erfolgreiche Anwendung von Ansätzen im RE nachgewiesen werden kann (vgl. [KC02] [IPC+15]), verdeutlichen PALOMARES et al., dass die Wiederverwendung der Anforderungen überwiegend auf einfachen Techniken basiert. Ein Beispiel ist hier das Kopieren von Anforderungen aus vorangegangenen Projekten und die nachgelagerte projektspezifische Anpassung [PQF17] (vgl. [IPP18]). HEUMESSER & HOUDEK argumentieren, dass diese Art der Wiederverwendung von Anforderungen zwar sinnvoll ist, da über die vorhandenen Spezifikationen Erfahrungen und Wissen gesammelt wurden. Sie geben jedoch auch an, dass die Übernahme von vorhandenen Anforderungen in einen neuen Projektkontext zu widersprüchlichen und inkonsistenten Spezifikationen führen kann. Insbesondere gilt dies für umfangreiche textuelle Spezifikationen, in denen die Abhängigkeiten beim Editieren, Hinzufügen oder Löschen von Anforderungen schwer zu überschauen sind [HH03]. Hinzu kommt, dass der Übernahmeanteil von Anforderungen im Vergleich zu weitergehenden Ansätzen relativ gering ist und der Fokus überwiegend auf nicht funktionalen Anforderungen liegt [PQF17].

FRANCH et al. bestätigen, dass einfache *Copy-Paste-Tailoring* Ansätze am weitesten verbreitet sind und führen die geringe Übernahme von bspw. musterbasierten Ansätzen oder Anforderungskatalogen auf eine fehlende Werkzeugunterstützung und unzureichende Integration von geeigneten Methoden in den RE Prozess zurück [FPQ20]. Ein möglicher Ausgangspunkt für die Adressierung dieser Punkte ist von KONRAD & CHENG beschrieben, die Anforderungsmuster auf Basis der UML Notation in Kombination mit textuellen Beschreibungen vorstellen [KC02]. Ausgehend von musterbasierten Ansätzen in der Softwaretechnik [GHJ+94] adaptieren die Autoren vorhandene Muster, um eine Anwendung

für die Anforderungsspezifikation zu unterstützen. Ein Anforderungsmuster ist demnach durch folgende Inhalte beschrieben [KC02]:

- **Name und Klassifizierung des Muster**: Der Mustername besteht aus einer Beschreibung des Musters, und die Klassifizierung gibt den Zweck des Musters an.
- **Absicht**: Enthält eine kurze Beschreibung der Probleme, die durch die Anwendung des Musters gelöst werden sollen.
- **Motivation**: Eine Beschreibung von exemplarischen Zielen eines Systems, die die Verwendung des Musters motivieren. Problemrahmendiagramme werden verwendet, um den Kontext des Problems zu beschreiben (vgl. [Jac01]). Außerdem beschreiben Anwendungsfälle und Anwendungsfall-Diagramme die Ziele der Anwendung des Musters.
- **Beschränkungen**: Alle umwelt-, disziplin- oder umsetzungsspezifischen Einschränkungen sollten hier aufgenommen werden, was sowohl funktionale als auch nichtfunktionaler Einschränkungen sein können.
- **Anwendbarkeit**: Enthält eine Beschreibung der Bedingungen, unter denen das Muster angewendet werden kann.
- **Struktur**: Eine Darstellung der Klassen und ihrer Beziehungen, die mittels UML-Klassendiagrammen dargestellt werden.
- **Verhalten**: Bietet eine anschauliche Darstellung von Szenarien für die Interaktion von Klassen und Objekten. Außerdem wird das Verhalten des Musters anhand von UML-Zustands- und Sequenzdiagrammen beschrieben.
- **Teilnehmer**: Auflistung der Klassen/Objekte einschließlich ihrer Verantwortlichkeit innerhalb des Anforderungsmusters.
- **Kollaborationen**: Beschreibt, wie Objekte und Klassen interagieren und welche Rolle sie bei der Ausführung verschiedener Aufgaben spielen.
- **Konsequenzen**: Beschreibt, welche Ziele durch das Muster erreicht werden und welche Kompromisse bei der Verwendung des Musters eingegangen werden müssen.
- **Entwurfsmuster**: Anwendbare Entwurfsmuster, die zur Verfeinerung der Anforderungsmuster verwendet werden können.
- **Auch bekannt als**: Beinhaltet eine Liste mit alternativen Namen für das Anforderungsmuster.
- **Verwandte Anforderungsmuster**: Beschreibt, welche Anforderungsmuster mit diesem Muster verwandt sind und beschreibt die Vor- und Nachteile des Musters in Bezug auf die verwandten Muster.

Basierend auf diesen Inhalten beschreiben KONRAD & CHENG einen Anforderungsmus-

terkatalog bestehend aus 10 Anforderungsmustern. Ein Muster, das im Automobilbereich eine hohe Relevanz hat, ist bspw. das *Sensor-Actuator* Muster. Die Autoren ordnen dieses Muster als ein strukturelles Muster ein (*Name und Klassifizierung des Musters*), das bei der Spezifikation von unterschiedlichen Typen von Sensoren eines eingebetteten Systems unterstützen soll (*Absicht*). Ein Beispiel sind Temperatursensoren, die erfasste Temperaturwerte entweder aktiv bereitstellen, oder die Abfrage von Temperaturwerten ermöglichen. In beiden Fällen ist die Auswertung der Signale über eine zu entwickelnde Komponente zu realisieren (*Motivation*). Dabei wird bereits über das Muster festgelegt, welche Rahmenbedingungen bei der Ausgestaltung einzuhalten sind. So ist bspw. definiert, dass der aktuelle Status von Sensoren und Aktuatoren abgefragt werden muss (*Beschränkungen*). Die *Struktur* des *Sensor-Actuator* Musters ist in Abbildung 3-11 über ein Klassendiagramm

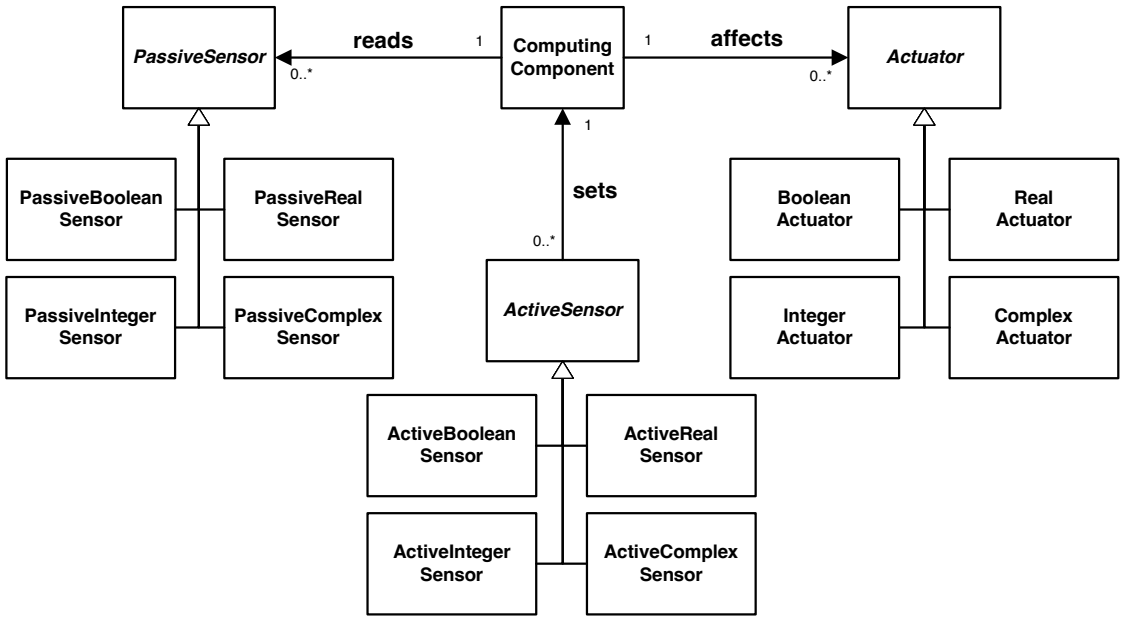


Bild 3-11: Struktur des Sensor-Actuator *Musters beschrieben über ein UML Klassendiagramm [KC02].*

dargestellt. Das *Verhalten* ist über das in Bild 3-12 dargestellte Sequenzdiagramm spezifiziert. Die Auflistung der *Teilnehmer* resultiert aus den Elementen des Klassendiagramms. Die *Kollaborationen* können ausgehend von der Verhaltensbeschreibung ausformuliert werden. Mögliche *Konsequenzen* bei der Anwendung des Muster können bswp. sein, dass die Klassen für Sensoren und Aktuatoren eine gemeinsame Schnittstelle verwenden. Als *Entwurfsmuster* kann u.a. ein *Observer* Muster aus der Softwaretechnik verlinkt werden, das es ermöglicht, die zentrale Komponente zu benachrichtigen, wenn sich Sensorwerte eines aktiven Sensors ändern [KC02].

Weitere ausführliche Beispiele der Anforderungsmuster sind in [KC02] beschrieben. Ebenso ist die erfolgreiche Anwendung innerhalb des Automobilbereichs am Beispiel eines *Adaptive Cruise Control* (ACC) und eines *Electronically Controlled Steering* (ECS) Systems beschrieben. Die Autoren kommen zu der Schlussfolgerung, dass der Musterkatalog die Anforderungsmodellierung erheblich vereinfacht und die Wiederverwendung der Mo-

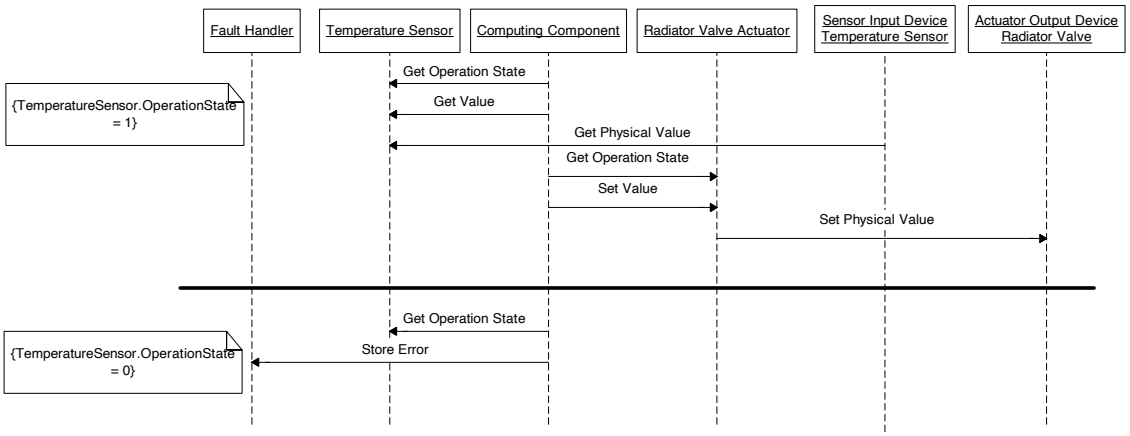


Bild 3-12: Verhalten des Sensor-Actuator *Musters beschrieben durch ein Sequenzdiagramm [KC02].*

delle unterstützt, da die Anwendung der Muster zur Spezifikation des ersten Systems (ACC) dazu geführt hat, dass ein Großteil der resultierenden Spezifikation für das zweite System (ECS) wiederverwendet werden konnte. Zudem fördern die Muster nach Ansicht der Autoren eine frühzeitige Identifikation von Elementen des Systems, die ohne die Verwendung von Mustern schneller übersehen und erst zu späteren Zeitpunkten spezifiziert würden. Das betrifft bspw. die Einbindung und Ausprägung von Sensoren entsprechend dem vorherigen Beispiel oder die Beschreibung von Schnittstellen [KC02].

Dieser frühe Ansatz aus dem RE zeigt, dass Anforderungsmuster eine geeignete Technik zur Wiederverwendung von Lösungswissen sein können. Da jedoch in aktuelleren Studien deutlich wird, dass die Übernahme und Anwendung dieser Techniken in der industriellen Praxis gering ist [PQF17; IPP18; FPQ20], besteht weiterer Forschungsbedarf hinsichtlich einer niederschwelligen Spezifikation und Anwendung der Anforderungsmuster und einer besseren Verzahnung der dafür notwendigen Methoden mit den vorhandenen RE Prozessen. PALOMARES et al. geben hierzu folgendes an:

> *„The existence of a well-defined method for using SRPs [Software Requirements Pattern] as well as the existence of tool support were considered to be the most critical factors for the introduction of SRPs by all types of respondents (...). Remarkably, the respondents who had used SRPs gave more relevance to the existence of a community of users supporting SRPs (...) [PQF17].“*

Wesentlich für den Erfolg von musterbasierten Ansätzen ist demnach die aktive Nutzung und Weiterentwicklung der Muster (vgl. [Clo06]).

3.4.3 Lösungswissen für die Testspezifikation

Für die Wiederverwendung von Testfällen existieren zahlreiche kommerzielle Werkzeuge mit unterschiedlichen Schwerpunkten (vgl. [SLS11, S. 209]). Häufig kommen Datenbanken

zum Einsatz, die über unterschiedliche Sichten die Testplanung und das Test-Management unterstützen (s. Phasen des Testprozesses in Bild 3-8). Hier steht vor allem die *Traceability* von Artefakten des RE und der V&V im Vordergrund (vgl. [BRB+13]). Das Erfassen und Wiederverwenden von Lösungswissen, wie aus einer bestehenden Testbasis geeignete Testfälle abgeleitet und miteinander in Beziehung gesetzt werden können, wird hierdurch nicht oder nur indirekt unterstützt.

Der eigentliche Entwurf von Testfällen, für den das Lösungswissen relevant ist, erfolgt in der Praxis häufig manuell und hängt stark von der Expertise des jeweiligen Entwicklers ab. UTTING et al. geben hierzu folgendes an:

> *"(...) traditionally, the process of deriving tests tends to be unstructured, barely motivated in the details, not reproducible, not documented, and bound to the ingenuity of single engineers [UPL12]."*

Als Antwort darauf ist das modellbasierte Testen ein Ansatz zur Strukturierung und Automatisierung des Testentwurfs (vgl. [DSV+07; BRB+13]). Hierbei kann ein formales Modell eines Systems für die automatisierte Generierung von Testfällen verwendet werden (bspw. auf Basis von UML Modellen, vgl. [NFL+06; HGB08]). Wie oben beschrieben (Kapitel 3.2), erfolgt jedoch die Anforderungsspezifikation in der Automobilindustrie überwiegend in unstrukturierter natürlicher Sprache. Somit ist es notwendig, die Anforderungen in einem Zwischenschritt in geeignete Modelle zu überführen [YBL11], um aus diesen Modellen Testfälle generieren zu können. Ein geeigneter Ansatz, der dieser Vorgehensweise folgt, ist von FISCHBACH beschrieben [Fis22]. Hiernach werden zunächst kausale Abhängigkeiten in natürlich-sprachlichen Anforderungen identifiziert und in einem Zwischenschritt in Ursache-Wirkungs-Graphen transformiert. Im letzten Schritt werden diese Ursache-Wirkungs-Graphen für die Generierung der Testfälle genutzt [FFV+23]. Dieser Ansatz zeigt, dass modellbasierte Ansätze den Testfallentwurf unterstützen und in Teilen automatisieren können, was zu einer besseren Reproduzierbarkeit des Testentwurfs beiträgt. Ebenso wird deutlich, dass die Qualität der resultierenden Testfälle im Vergleich zu manuellen Vorgehensweisen verbessert werden kann [FFV+23]. Das Wissen, wie aus einer Anforderung eine geeignete Menge an Testfällen abgeleitet werden kann, ist für dieses Beispiel Teil der Methode und des zugehörigen Werkzeugs (s. [FFS+21]) und somit nicht an einzelne Entwickler gebunden.

Bei dieser und vergleichbaren Techniken liegt jedoch der Fokus auf einzelnen Anforderungen und Testfällen. Für eine systematische Externalisierung von Lösungswissen, das einen Test-Designer bei der Erstellung einer Testspezifikation unterstützt, ist eine weitergehende Vernetzung und Kontextualisierung einzelner Testfälle notwendig. Wichtige Aspekte sind hier Informationen zu dem Testobjekt, sowie zu den jeweiligen Testumgebungen. Wie oben beschrieben (s. Kapitel 3.3.2), sind diese Testumgebungen in der Testspezifikation zu definieren und mit den Testfällen zu vernetzen.

In der von JUHNKE et al. durchgeführten Studie innerhalb der Automobilindustrie wird deutlich, dass unzureichendes *Wissen über das Testobjekt* eine wesentliche Fehlerquelle in

den Testspezifikationen ist. Demnach kann fehlendes Wissen über das Testobjekt dazu führen, dass die spezifizierten Testfälle für die Umsetzung in einem realen Kontext ungeeignet sind. Das betrifft zum einen die Spezifikation der Testfälle, zum anderen aber auch deren Interpretation bei der Umsetzung [JTH20]. Da hier unterschiedliche Rollen und Unternehmen beteiligt sein können (s. Kapitel 3.3.2), ist eine geeignete Repräsentation des für die Testspezifikation relevanten Wissens über das Testobjekt von großer Bedeutung. Ebenso ist fehlendes *Wissen über die Testumgebungen* eine Herausforderung für die Testspezifikation. So fehlt bspw. in verteilten Entwicklungsprojekten das Wissen über die zur Verfügung stehenden Testumgebungen und deren Eigenschaften und Funktionalitäten. Dieses Wissen ist jedoch bei der Erstellung und Umsetzung der Testspezifikation von entscheidender Bedeutung [YMB+19] [JTH20].

3.4.4 Herausforderungen für die Externalisierung und Wiederverwendung von Lösungswissen

Die Wiederverwendung von Lösungswissen hat in der Automobilindustrie einen hohen Stellenwert. Das Lösungswissen ist dabei implizit in den Spezifikationen enthalten, und es kommen überwiegend dokumentenbasierte *Copy-Paste-Taloring* Ansätze zum Einsatz, um das Lösungswissen wiederzuverwenden. Der Wiederverwendungsgrad ist durch diese Ansätze jedoch gering [PQF17], und insbesondere für große Spezifikation kann die Übernahme und Anpassungen von Spezifikationen eine Fehlerquelle sein, insbesondere wenn die Anpassung auf manuellen Analysen beruht [HH03].

Wie ANACKER zeigt [Ana15, S. 32 f.], können Lösungsmuster eine geeignete Form für die explizite Beschreibung von Lösungswissen sein. Im Bereich der Anforderungsspezifikation existieren bereits musterbasierte Ansätze [KC02; PQF17; FPQ20], die jedoch in der industriellen Praxis nicht zur Anwendung kommen. Hier besteht die Herausforderung, eine niederschwellige Beschreibung und Editierbarkeit von Lösungsmustern zu ermöglichen. Dafür sind geeignete Methoden notwendig, um das Vorgehen in bestehende Entwicklungsprozesse zu integrieren und so eine aktive Nutzung und kontinuierliche Weiterentwicklung der Lösungsmuster zu fördern [PQF17; Clo06].

Im Bereich der Testspezifikation ist Wissen über die Erstellung der Testfälle, des Testobjekts und der Testumgebungen relevant. Wie oben beschrieben, ist die Testfallerstellung stark abhängig von der individuellen Expertise des einzelnen Entwicklers (vgl. [UPL12; ABK+16]) Es existieren zwar geeignete Techniken, um aus textuellen Anforderungen geeignete Testfälle zu generieren, jedoch ist eine weitergehende Kontextualisierung und Vernetzung der Testfälle notwendig, um das Wissen für die Erstellung und Interpretation der Testspezifikation zugänglich zu machen. Insbesondere im Kontext der *Advanced Systems* ist Wissen über die verteilten Testumgebungen einschließlich deren Eigenschaften notwendig.

Weitergehend existieren umfassende Ansätze zur Qualitätskontrolle von Testspezifikationen, bspw. mittels Review-Leitfäden und Checklisten als analytische Qualitätssiche-

rungsmethode [Juh21]. Es besteht jedoch die Herausforderung, das Wissen zur qualitativ hochwertigen Erstellung von Testspezifikationen systematisch zu dokumentieren und bereits frühzeitig innerhalb des Spezifikationsprozesses nutzen und fortlaufend aktualisieren zu können.

3.5 Problemabgrenzung

Zukünftige Systeme im Automobilbereich können als *Advanced Systems* charakterisiert werden. Diese Systeme zeichnen sich durch eine starke Vernetzung und einen hohen Grad an interaktiver soziotechnischer Interaktion aus [DAR+21, S. 28]. Die soziotechnische Interaktion basiert dabei auf der Integration von internet- und plattformbasierten Diensten mit dem Fahrzeugsystem. Auf der einen Seite entstehen hierdurch enorme Potentiale für Entwicklung von innovativen Produkt-Service-Systemen [GDE+18], auf der anderen Seite hat die starke Vernetzung mit heterogenen Systemen einen großen Einfluss auf das System der Marktleistungsentstehung [DAR+21, S. 102] und ist von den an der Entwicklung beteiligten Unternehmen entsprechend zu berücksichtigen. Das ist insbesondere für die Unternehmen in der Automobilindustrie mit den historisch gewachsenen Zuliefererstrukturen und einem hohen Grad an regulatorischen Vorgaben herausfordernd.

Dem RE und der V&V kommt hierbei eine besondere Bedeutung zu, da diese Bereiche entscheidende Schnittstellen im System der Marktleistungsentstehung sind, sowohl innerhalb einzelner Organisationen als auch organisationsübergreifend. Vor diesem Hintergrund bestehen im Kontext der Systementwicklung im Automobilbereich aus der Perspektive eines Zulieferunternehmens vier wesentliche **Herausforderungen**:

- **Spezifikation und Analyse von Anforderungen**: Die Basis für die Entwicklung von Systemen in komplexen Entwicklungspartnerschaften sind rechtsverbindliche Anforderungen. Für den Austausch und das Management dieser überwiegend textuell spezifizierten Anforderungen sind die Unternehmen in der Automobilindustrie an Standards gebunden, die Referenzprozesse wie die *Anforderungsanalyse* definieren. Ziel der Anforderungsanalyse ist eine umfassende Bewertung von *Stakeholder-Anforderungen* und eine anschließende Überführung der darin enthaltenen Informationen in *Systemanforderungen*, die wiederum die Basis für den Systementwurf sind (vgl. Kapitel 3.2.4). Durch den Wandel der Systeme, von komplexen monolitischen Systemen hin zu dynamisch vernetzten *Advanced Systems*, gelingt es kaum, der Forderung nach einer frühen und umfassenden Anforderungsanalyse nachzukommen. Gründe hierfür sind, neben der Komplexität der Anforderungsspezifikationen, ein hoher Grad an Unsicherheit und häufige Anforderungsänderungen, die eine Vielzahl von Änderungsanfragen für bereits realisierte Systeme und damit zusätzliche Entwicklungsiterationen zur Folge haben [BRB+13]. Obwohl zahlreiche Ansätze existieren, die eine frühe Anforderungsanalyse auf Basis von Anforderungsmodellen unterstützen, kommen diese Ansätze in der industriellen Praxis kaum zum Einsatz [LTK19]. Es fehlen geeignete und anwendbare Methoden, Modellierungssprachen und Werk-

zeuge für die Anforderungsanalysephase in komplexen Entwicklungsprojekten mit den in der Automobilindustrie geltenden Rahmenbedingungen.

- **Entwurf und Spezifikation von Tests in volatilen Entwicklungsprojekten**: Im Kontext des ASE müssen Validierungsziele klar definiert und zwischen den Entwicklungspartnern abgestimmt werden. Eine erfolgreiche Systementwicklung ist nur möglich, wenn die Rolle des zu entwickelnden Systems in einem Systemverbund verstanden ist und die damit zusammenhängenden Validierungsziele für das jeweilige System definiert sind (vgl. [Hon13]). Durch die starke Vernetzung entsteht eine Vielzahl neuer Anwendungsfälle, wobei diese jedoch möglichst frühzeitig auf Basis von abgestimmten Validierungszielen bei dem Entwurf und der Spezifikation von Tests zu berücksichtigen sind. Gleichzeitig erfordern die Rahmenbedingungen in der Automobilindustrie den Nachweis einer durchgängigen Traceability von Anforderungen und den Elementen des Architekturentwurfs zu den Testfällen und den dazugehörigen Testergebnissen (vgl. Kapitel 3.3.2). Die große Zahl der zu verifizierenden Anforderungen an heutige und zukünftige Systeme führt dabei zu komplexen Testspezifikationen, die auf Basis von dokumentenzentrierten Ansätzen kaum überschaubar und wartbar sind (vgl. Kapitel 3.3.4). Da sich Anforderungen und damit verbundene Validierungsziele im ASE Kontext häufig ändern können, ist die Erstellung von vollständigen und konsistenten Testspezifikationen mit etablierten Ansätzen kaum zu leisten. Der Prozess für den Entwurf und die Spezifikation von Tests muss durch geeignete Methoden und Werkzeuge unterstützt werden, um die Komplexität der Spezifikationen beherrschbar zu machen. Dabei muss ein iteratives Vorgehen unterstützt werden, um auf Anforderungsänderungen aus dem Entwicklungsumfeld möglichst frühzeitig reagieren zu können.

- **Unzureichende Verzahnung von RE und V&V**: Das RE verfolgt das Ziel, die Bedarfe unterschiedlicher Stakeholder zu verstehen, zu analysieren und zu dokumentieren, um das Risiko zu minimieren, dass diese Bedarfe durch das zu entwickelnde System nicht erfüllt werden (s. Kapitel 3.2). Der Nachweis, dass die dokumentierten Anforderungen richtig umgesetzt sind und die Bedarfe und Erwartungen durch das entwickelte System erfüllt werden, erfolgt durch die V&V (s. Kapitel 3.3). Beide Bereiche haben somit eine starke Beziehung zueinander und eine hohe Relevanz für die erfolgreiche Systementwicklung mit einem hohen Einfluss auf die Entwicklungskosten [FWK+17] [ABK+16]. Trotz der starken Beziehung zwischen dem RE und der V&V stellen BJARNASON et al. jedoch fest, dass Herausforderungen in beiden Disziplinen sowohl in den Industrieunternehmen als auch in der Forschung überwiegend getrennt voneinander betrachtet werden [BRB+13]. BARMI et al. zeigen, dass in den Forschungszweigen des RE und der V&V formale Methoden, bspw. zur automatisierten Analyse von Anforderungen (vgl. [YBL11]) oder zur Generierung von Testfällen, untersucht werden [BEF11]. Nach LIEBEL et al. ist jedoch eine ausschließliche Konzentration auf die Formalisierung und modellbasierte Analysen nicht ausreichend, um den aktuellen Bedürfnissen der Industrie gerecht zu werden

[LTK19]. Demnach wird zwar die Verwendung von Modellen zur Unterstützung der Analyse- und Spezifikationstätigkeiten nicht infrage gestellt, wohl aber die Anwendbarkeit und Pflege von großen formalen Modellen [LK23]. Folglich sind Methoden zu beschreiben, die mit Hilfe von geeigneten Modellierungstechniken das RE und V&V enger miteinander verzahnen. Wie BJARNASON et al. feststellen, ist dabei die Kommunikation und Abstimmung zwischen den im RE und der V&V beteiligten Rollen entscheidend und entsprechend in den methodischen Ansätzen zu adressieren [BRB+13; DHK+17].

- **Wiederverwendung von Lösungswissen**: Wie oben ausgeführt, basiert die Wiederverwendung von Lösungswissen im Bereich der Anforderungsspezifikation überwiegend auf *Copy-Paste-Tailoring* Ansätzen (s. Kapitel 3.4.2). Hier wird deutlich, dass etablierte Vorgehensweisen wenig systematisch sind und der Wiederverwendungsgrad vergleichsweise gering ist. Zusätzlich sind *Copy-Paste-Tailoring* Ansätze eine potentielle Fehlerquelle bei manuellen Adaptierungen von Anforderungen in komplexen und dokumentenbasierten Spezifikationen. Da bereits Ansätze existieren, um bspw. über Muster Lösungswissen wiederverwendbar zu machen, besteht die Herausforderung darin, diese Techniken insbesondere für den Bereich der Anforderungs- und Testspezifikation zu adaptieren und über geeignete methodische und modellbasierte Techniken anwendbar zu machen.

Diese Herausforderungen machen deutlich, dass im Kontext des ASE ein **Bedarf** für eine *Systematik zur integrativen Anforderungsanalyse und Testspezifikation für die Entwicklung von Systemen im Automobilbereich* besteht, um Systemarchitekten und Test-Designer im Spezifikationsprozess zu unterstützen und durch die engere Verzahnung von Aktivitäten innerhalb des RE und der V&V Lösungswissen systematisch wiederzuverwenden. Zur Adressierung der beschriebenen Herausforderungen sollte die Systematik folgende Inhalte berücksichtigen:

- **Vorgehensmodell**: Es wird ein Vorgehensmodell benötigt, das einerseits die für ein Zulieferunternehmen in der Automobilindustrie geltenden Rahmenbedingungen für die Spezifikation von Anforderungen und Tests berücksichtigt, andererseits aber auch die aus den Eigenschaften der *Advanced Systems* resultierenden und oben genannten Herausforderungen adressiert. Das bedeutet, dass ein Vorgehensmodell einerseits einen Phasen-Meilenstein orientierten Entwicklungsprozess adressieren muss, um zu definierten Meilensteinen qualitativ hochwertige Spezifikationen bereitzustellen. Andererseits muss jedoch auch ein iteratives, den Problemlösungsprozess unterstützendes Vorgehen adressiert werden, das Systemarchitekten und Test-Designer eine kontinuierliche Abstimmungen von Anforderungen und Validierungszielen in komplexen Entwicklungspartnerschaften ermöglicht. Dabei ist eine engere Verzahnung des RE und V&V nötig, die über die Betrachtung von Aspekten der Traceability und der Testfallgenerierung hinausgeht und die Zusammenarbeit zwischen Systemarchitekten und Test-Designer methodisch unterstützt.

- **Techniken zur integrativen Test- und Anforderungsmodellierung**: Zur Adressierung der unzureichenden Verzahnung von des RE und der V&V sind geeignete Techniken notwendig, die bei der Erstellung und Analyse von komplexen Anforderungs- und Testspezifikationen unterstützen. In der Problemanalyse wird deutlich, dass dabei einerseits eine niederschwellige Spezifikation in natürlicher Sprache möglich sein muss, andererseits aber auch modellbasierte Techniken notwendig sind, um die Komplexität der Spezifikationen beherrschbar zu machen und Analyse- und Spezifikationstätigkeiten zu automatisieren. Da bereits zahlreiche Techniken existieren, ist zu untersuchen, wie geeignete Techniken miteinander kombiniert und methodisch unterstützt werden können, um eine integrative, anwendungsorientierte und modellbasierte Spezifikation und Analyse von Anforderungen und Tests zu ermöglichen.
- **Technik zur Externalisierung und Wiederverwendung von Lösungswissen**: Für die Entwicklung von *Advanced Systems* ist die Wiederverwendung von Lösungswissen von herausragender Bedeutung. Im Kontext dieser Arbeit betrifft das insbesondere das Wissen für die Erstellung von Anforderungs- und Testspezifikationen aus Sicht eines Systemarchitekten und Test-Designers. Es besteht der Bedarf für eine Technik, die die Anwender bei einer systematischen Externalisierung und Wiederverwendung von Lösungswissen unterstützt.
- **Werkzeuge zur Umsetzung der Modellierungstätigkeiten**: Die Methoden und modellbasierten Ansätze zur Unterstützung einer engeren Verzahnung des RE und der V&V sowie der systematischen Wiederverwendung von Lösungswissen sind nur realisierbar, wenn diese durch geeignete Werkzeuge unterstützt werden. Demnach besteht der Bedarf, Werkzeuge bereitzustellen und geeignet miteinander zu kombinieren.

3.6 Anforderungen an die Arbeit

Das Ergebnis der Problemanalyse sind die im folgenden beschriebenen Anforderungen an die *Systematik zur integrativen Anforderungsanalyse und Testspezifikation für die Entwicklung von Systemen im Automobilbereich*:

A1) Verständlichkeit: Die Systematik soll Entwicklungsingenieure ohne Expertenwissen zur formalen Modellierung von Anforderungen und Testfällen unterstützen. Hieraus resultiert die Notwendigkeit, eine verständliche und intuitive Modellierung zu ermöglichen, um die Spezifikations- und Analysetätigkeiten zu unterstützen und nicht zu erschweren.

A2) Ganzheitliche Betrachtung des Systemkontextes: Aufgrund der zunehmenden Vernetzung von Systemen im Automobilbereich besteht die Notwendigkeit, den Systemkontext ganzheitlich und über Fahrzeuggrenzen hinweg zu betrachten. Da die von einzelnen Unternehmen entwickelten Lösungen einen Beitrag für die Realisierung von *Advanced Systems* liefern, ist es für das RE und die V&V wichtig, den für die Entwicklung des jeweiligen Systems relevanten Kontext zu erfassen.

A3) Orientierung an etablierten Standards in der Steuergeräteentwicklung: Zur Anwendung der Systematik ist eine Orientierung an den in der Automobilindustrie relevanten Standards notwendig. Aus Sicht eines Tier1 Zulieferers betrifft dies insbesondere die Vorgaben zur *Anforderungsanalyse* und *Testspezifikation* in ASPICE [VDA].

A4) Unterstützung von funktionalen Anforderungen in natürlicher Sprache: Die Systematik muss berücksichtigen, dass Anforderungen in natürlicher Sprache spezifiziert werden und in dieser Form zwischen den an der Entwicklung einer Funktion eines *Advanced Systems* beteiligten Unternehmen ausgetauscht werden.

A5) Unterstützung einer iterativen Spezifikation von Anforderungen: Durch die starke Vernetzung der *Advanced Systems*, wechselnden Entwicklungspartnern und einer sich ständig weiterentwickelnden Gesamtfunktionalität ist der Zulieferer eines Systems mit häufigen Anforderungsänderungen konfrontiert, die iterativ spezifiziert und analysiert werden müssen.

A6) Unterstützung einer frühen und kontinuierlichen Validierung von Anforderungen: Da sich Anforderungen im Kontext der Entwicklung von *Advanced Systems* häufig ändern und der Einfluss dieser Änderungen auf das zu entwickelnde System möglichst früh zu bewerten und abzustimmen ist, muss die Validierung und nicht nur die Verifikation der Anforderung frühzeitig und kontinuierlich erfolgen.

A7) Unterstützung einer iterativen Spezifikation von Testfällen: Analog zu einer iterativen Spezifikation von Anforderungen und zur Realisierung einer frühen und kontinuierlichen Verifikation und Validierung ist es erforderlich, Testfälle iterativ zu spezifizieren.

A8) Wiederverwenden von Lösungswissen: Die Erstellung von qualitativ hochwertigen Anforderungs- und Testspezifikationen erfordert die Wiederverwendung von bereits erfolgreich eingesetztem Lösungswissen. Hierzu sind geeignete Methoden und Werkzeuge erforderlich, um das Lösungswissen zu erfassen und verfügbar zu machen.

A9) Werkzeugunterstützung: Die Spezifikation und Analyse von Anforderungen, der Entwurf und die Spezifikation von Testfällen sowie die Wiederverwendung von Lösungswissen in einem ASE-Kontext, ist ohne eine geeignete Werkzeugunterstützung nicht realisierbar. Es ist eine Werkzeugunterstützung erforderlich, die eine verständliche und kollaborative Modellierung unterstützt und so die Zusammenarbeit der beteiligten Entwicklungsingenieure fördert.

4 Stand der Technik

Dem zu Beginn (Kapitel 2.3) vorgestellten Aufbau der Arbeit entsprechend, beschreibt dieses Kapitel die für die Entwicklung der einzelnen DSR Artefakte notwendigen Grundlagen. Diese sind strukturiert durch die drei Handlungsfelder 1) *Spezifikation und Analyse von Anforderungen*, 2) *Entwurf und Spezifikation von Tests* und 3) *Externalisierung und Wiederverwendung von Lösungswissen*.

Ausgehend von der zuvor beschriebenen Problemabgrenzung sind die in Kapitel 4.1 beschriebenen Ansätze für die Systemmodellierung die Grundlage für die Entwicklung der *Systematik zur integrativen Anforderungsanalyse und Testspezifikation für die Entwicklung von Systemen im Automobilbereich*. Darauf aufbauend beschreibt Kapitel 4.2 die grundlegenden Konzepte und Sprachen für die Modellierung von Anforderungen. Kapitel 4.3 beinhaltet die Grundlagen für den Entwurf und die Spezifikation von Tests. Daran anschließend beschreibt Kapitel 4.4 verwandte Arbeiten und methodische Ansätze für eine integrative Betrachtung der Anforderungs- und Testspezifikation. Schließlich beinhaltet Kapitel 4.5 die für diese Arbeit relevanten Ansätze für die Wiederverwendung von Lösungswissen.

4.1 Ansätze für die Systemmodellierung

Das SE dient in dieser Arbeit als verbindende Disziplin für die drei zuvor genannten Handlungsfelder und ist somit die Basis für die entwickelte Systematik. Sowohl die Aspekte des RE als auch der V&V sind Teil des SE, was auch in der Definition des internationalen Dachverbands für SE (INCOSE)[1] zum Ausdruck kommt.

> *„Systems engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem (...) [DRF+15, S. 11]. “*

DUMITRESCU et al. haben eine umfassendere Sichtweise auf das SE, die über den Entwurf und die Ausgestaltung der technischen Systeme hinausgeht und insbesondere auch das Projektmanagement und die Unternehmensorganisation mit einschließt. Demnach umfasst SE

> *„(...) die Gesamtheit aller Entwicklungsaktivitäten. Es erhebt den Anspruch, die Akteure in der Entwicklung komplexer Systeme zu managen. Dazu integriert es die Systemgestaltung und das Projektmanage-*

[1] https://www.incose.org, zuletzt abgerufen am 27.10.2022

> *ment unter Berücksichtigung der unternehmensspezifischen Organisation [DAR+21, S. 59]. “*

Dabei steht das ganzheitliche Systemdenken im Vordergrund, um die Erarbeitungen von dispziplinübergreifenden Lösungen zu ermöglichen. Bei der Anwendung von SE sehen Unternehmen in der Automobilindustrie und verwandten Branchen wesentliche Nutzenpotentiale durch ein *verbessertes Systemverständnis*, eine bessere *Rückverfolgbarkeit und Transparenz*, sowie eine Verbesserung der *Komplexitätsbeherrschung* [DAR+21, S. 64] [FDN+22]. Um diese Nutzenpotentiale zu erschließen ist die Systemmodellierung ein wesentlicher Aspekt für eine erfolgreiche Anwendung des SE, was durch das Model-Based Systems Engineering (MBSE) adressiert wird.

4.1.1 Model-Based Systems Engineering (MBSE)

Wie auch für das SE existieren für das Model-Based Systems Engineering (MBSE) je nach Kontext unterschiedliche Definitionen. Eine weit verbreitete Definition stellt auch hier die INCOSE bereit. Demnach kann MBSE als

> *„(...) formalized application of modeling to support system requirements, design, analysis, verification, and validation activities, beginning in the conceptual design phase and continuing throughout development and later life cycle phases [DRF+15] “*

verstanden werden. HOLT & PERRY kritisieren jedoch, dass diese Definition zu sehr auf das Engineering beschränkt und demnach zu eng gefasst ist. Zudem wird kritisiert, dass die Modellierung und somit das resultierende Systemmodell lediglich als Unterstützung der SE Aktivitäten gesehen wird. Die Autoren geben indessen an, dass das Systemmodell idealerweise als *single source of truth* im Zentrum der Entwicklung zu verstehen ist, wodurch dieses zum Treiber der Systementwicklung wird. Folglich hat das zentrale Systemmodell einen hohen Stellenwert, um die oben genannten Nutzenpotentiale des SE zu erschließen, was auch in der MBSE Definition nach HOLT & PERRY deutlich wird:

> *Model-based Systems Engineering is an approach to realising successful systems that is driven by a model that comprises a coherent and consistent set of views that reflect multiple viewpoints of the system [HP19, S. 8].*

Dieser Definition folgend sind das *Systemmodell*, sowie zusammenhängende und widerspruchsfreie *Sichten* auf das Modell aus unterschiedlichen *Standpunkten*, zentrale Elemente für die Entwicklung eines MBSE Ansatzes. Bild 4-1 verdeutlicht diese Zusammenhänge. Exemplarisch sind hier zwei Anwender eines MBSE Ansatzes dargestellt, die aus zwei unterschiedlichen Standpunkten auf das Systemmodell blicken. Die zwei Standpunkte spiegeln sich hier in zwei Sichten wider, die eine Teilmenge des Systemmodells darstellen. Diese Herangehensweise ermöglicht die Erstellung eines Systemmodells als Basis für die transdisziplinäre Systementwicklung [DAR+21, S. 60 f.]. Einerseits ermöglichen die

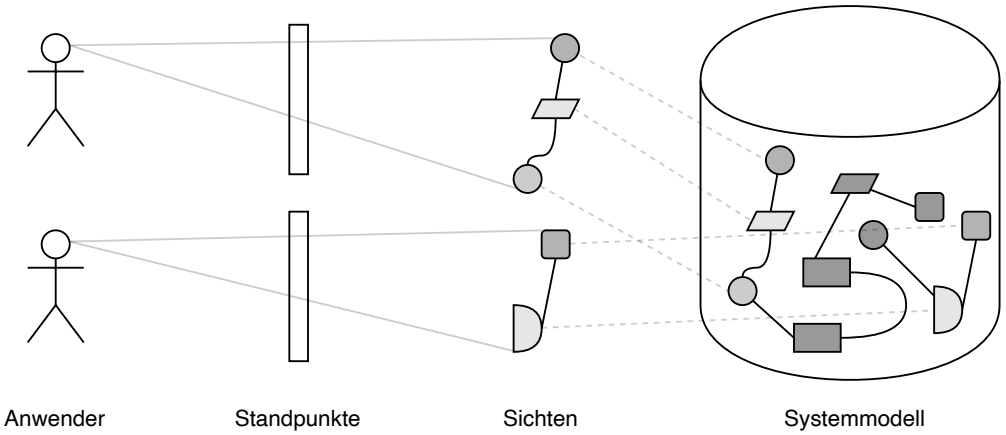


Bild 4-1: Standpunkte und Sichten für die aufgabenspezifische Bearbeitung eines zentralen Systemmodells nach [ISOd].

unterschiedlichen Standpunkte eine arbeitsteilige und aufgabenspezifische Bearbeitung des Modells mit unterschiedlichen Schwerpunkten. Andererseits wird die Erarbeitung von disziplinübergreifenden Lösungen unterstützt, da die einzelnen Sichten über das zentrale Systemmodell miteinander in Beziehung stehen und so Abhängigkeiten zwischen den Sichten bzw. Schwerpunkten in der Modellierung berücksichtigt werden.

Um auf dieser Basis einen MBSE Ansatz für Unternehmen anwendbar zu machen, ist wie in Bild 4-2 dargestellt eine geeignete Kombination von *Modellierungssprache*, *Modellierungswerkzeug* und *Methode* zu identifizieren, um eine zielgerichtete und effiziente Bearbeitung des zentralen Systemmodells zu ermöglichen [Wei16]. Eine allgemein verständliche Mo-

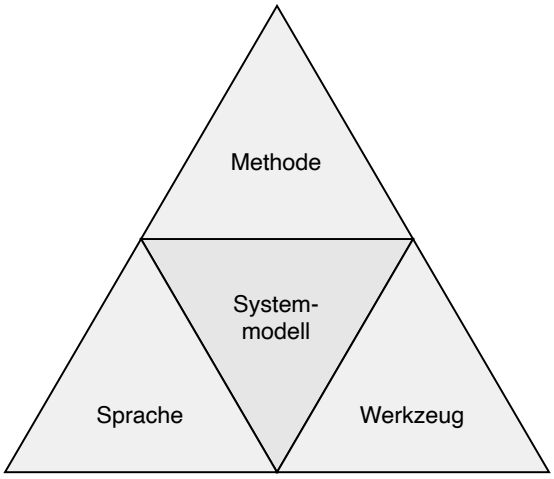


Bild 4-2: Zusammenhang von Modellierungssprache, Softwarewerkzeug und Methode zur Beschreibung eines zentralen Systemmodells nach [Kai13, S. 26]

dellierungssprache ermöglicht dabei die fachdisziplinübergreifende Modellierung des Systems, wobei die eigentliche Modellierungstätigkeit über ein Softwarewerkzeug erfolgt. Wie eine Modellierungssprache über ein Softwarewerkzeug eingesetzt werden kann, um

ein bestimmtes Modellierungsziel für einen bestimmten Zweck zu erreichen, wird über die Methode festgehalten (vgl. [Kai13, S. 26] und [Bre21, S. 32 f.]).

Eine formalere Beschreibung dieser Zusammenhänge erfolgt ebenfalls durch HOLT & PERRY. Die Grundsätze der Herangehensweise für die Entwicklung eines MBSE Ansatzes sind in Bild 4-3 dargestellt. Im Zentrum steht hier ein System, das durch ein Systemmodell abstrahiert wird. Ein Systemmodell besteht wiederum aus einer Sicht oder mehreren Sichten. Durch diese Zusammenhänge von System, Systemmodell und Sicht wird zum einen festgelegt, was modelliert werden soll, und zum anderen können die Bedarfe unterschiedlicher Anwender über unterschiedliche Sichten des Systemmodells berücksichtigt werden [HP19, S. 18 ff.].

Die Autoren heben hervor, dass ein Unterschied zwischen der Sicht als Teil des Systemmodells und einer Visualisierung über Diagramme besteht. Demnach kann eine Sicht über unterschiedliche Diagrammtypen auf Basis unterschiedlicher Notationen visualisiert werden, wobei sowohl textuelle als auch grafische Notationen zur Anwendung kommen können. Über die Beziehung eines Diagramms zu einer Sicht wird sichergestellt, dass ein konsistentes Systemmodell und keine Sammlung von inkonsistenten Diagrammen entsteht [HP19, S. 20].

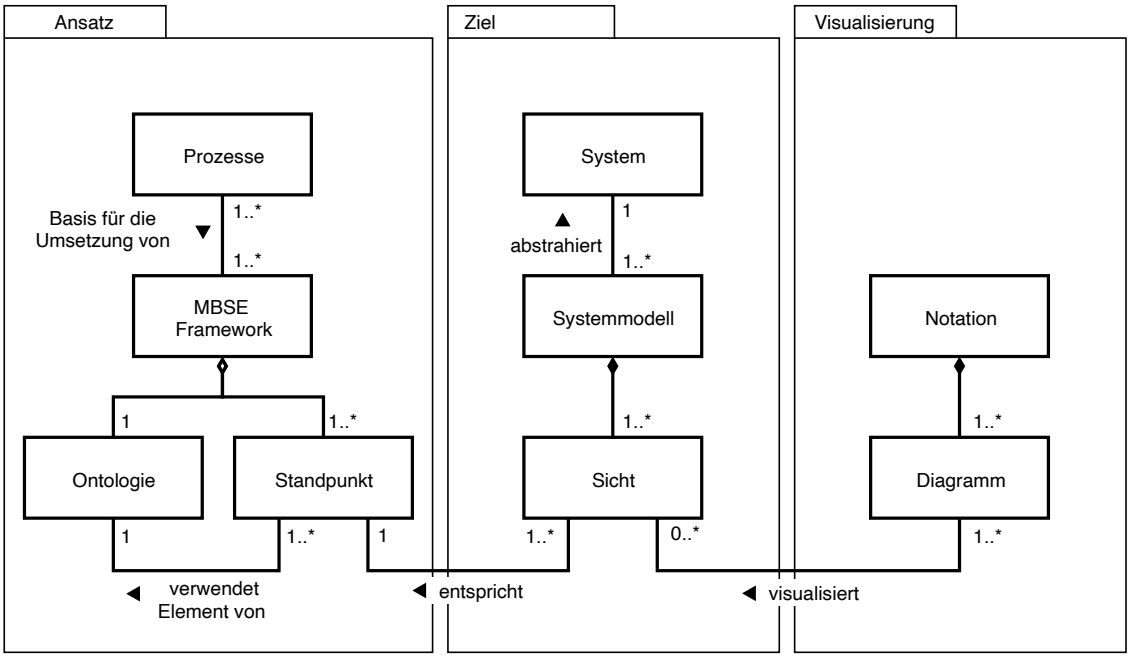


Bild 4-3: Grundlegender Ansatz für die Entwicklung eines MBSE Frameworks nach HOLT & PERRY *[HP19, S. 21]*

Wie in Bild 4-3 dargestellt, sind Prozesse innerhalb eines Unternehmens der Ausgangspunkt und die Basis für die Entwicklung eines MBSE Frameworks. Das MBSE Framework umfasst eine Ontologie und aufgabenspezifische Standpunkte. Der Begriff Ontologie wird hier verstanden als eine strukturierte Visualisierung von Konzepten und Begriffen, sowie deren Beziehungen untereinander. Somit entsteht durch die Ontologie eine domänenspe-

zifische Sprache für einen spezifischen MBSE Ansatz. Die Standpunkte verwenden die Elemente der Ontologie und dienen als Schablone für die Erstellung von Sichten.

Diese Zusammenhänge sind in Bild 4-4 konkretisiert. Das MBSE Framework besteht aus Standpunkten, die wiederum aus einzelnen Elementen des Standpunkts bestehen. Die Standpunkte verwenden Elemente einer Ontologie, wobei die einzelnen Elemente eines Standpunkts den Elementen der Ontologie entsprechen. Somit wird eine Konsistenz zwischen den Standpunkten und der Ontologie hergestellt. Eine aufgabenspezifische Sicht stimmt wiederum mit einem Standpunkt überein und beseht aus dem Element einer Sicht. Die Sicht und deren Elemente können somit als konkrete Umsetzung eines Standpunktes und als Basis für dessen Visualisierung verstanden werden [HP19, S. 27].

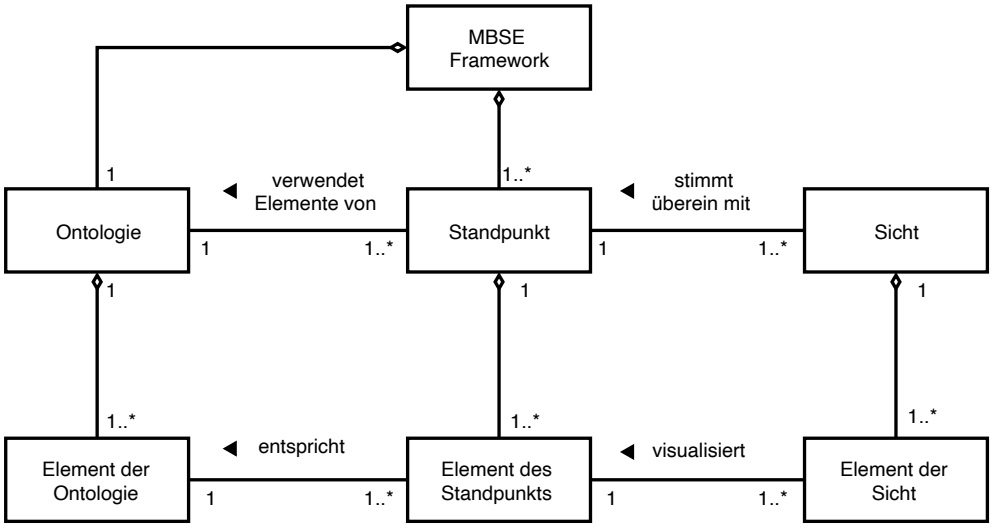


Bild 4-4: Zusammenhänge zwischen Ontologie, Standpunkt und Sicht zur Entwicklung eines MBSE Frameworks HOLT & PERRY *[HP19, S. 27]*

4.1.2 Validierung und Verifikation in der Produktentwicklung

Ausgehend von den zuvor beschriebenen MBSE Grundlagen sind Prozesse die Basis für die Umsetzung eines MBSE Frameworks (s. Bild 4-3). Innerhalb von stark regulierten Branchen wie der Automobilindustrie haben dabei Prozesse für die V&V einen besonders hohen Stellenwert. In der Praxis wird die V&V dabei überwiegend als eine abschließende Phase in der Produktentwicklung zur Absicherung der Produkteigenschaften verstanden [AMB+15] (vgl. Kapitel 3.1.2). Der Sichtweise von ALBERS et al. folgend ist die Validierung jedoch ein zentrales Element der Produktentwicklung und die einzige Aktivität, durch die neues Wissen generiert wird. Die Validierung hat demnach eine Initiierungs- und Koordinierungsfunktion für die Entwicklung neuer Produkte. Die Autoren sprechen in diesem Zusammenhang vom *Pull-Prinzip der Validierung* und heben hervor, dass die Validierung nicht als abschließende Phase in der Produktentwicklung, sondern als ein kontinuierlicher Prozess verstanden werden muss [AMB+15]:

> *"Validierung und nicht nur Verifizierung muss von Anfang an und kontinuierlich über die Entwicklung hinweg systematisch durchgeführt, die Erkenntnisse müssen aufbereitet und dokumentiert und diese in Form von neuen oder modifizierten Entwicklungszielen in das Zielsystem des Produktentwicklungsprozesses zurückgeführt werden [ABK+16]."*

Zur Einordnung dieser Aussage veranschaulicht Bild 4-5 die Validierung als zentrale Aktivität in der Produktentwicklung, die basierend auf den Arbeiten von ROPOHL [Rop75] und PATZAK [Pat82] ein System-Triple bestehend aus einem Zielsystem, Objektsystem und Handlungssystem miteinander verbindet. Dieses System-Triple wird als ZHO-Modell bezeichnet und ermöglicht die Beschreibung der Wechselwirkungen innerhalb der Produktentwicklung [ALE11]. Das *Zielsystem* beinhaltet alle Ziele für das zu entwickelnde Produkt sowie deren Beziehungen untereinander. Ebenso sind hier der Kontext sowie Randbedingungen und für die Entwicklung relevante Beschränkungen beschrieben. Das *Objektsystem* umfasst virtuelle und physische Artefakte, wie dokumentierte Zwischenergebnisse, Prototypen und das entwickelte Produkt. Das *Handlungssystem* wird hier als ein soziotechnisches System bestehend aus strukturierten Aktivitäten, Methoden und Prozessen beschrieben, das alle für die Produktentwicklung benötigen Ressourcen wie Personal oder Budget enthält. Es erstellt sowohl das Zielsystem als auch das Objektsystem, wobei diese allein über das Handlungssystem miteinander in Beziehung stehen [Ebe15, S. 18].

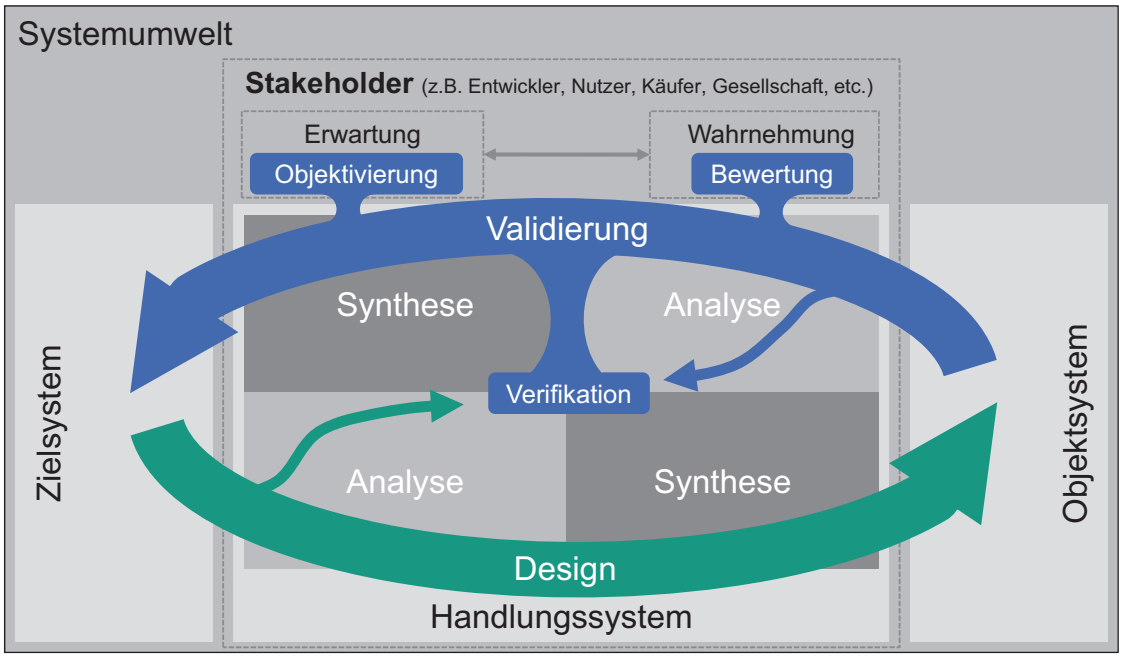


Bild 4-5: Einordnung von Validierung und Verifikation in der Produktentwicklung auf Basis des ZHO-Modells [ABK+16]

Ergänzend zum ZHO-Modell integriert Bild 4-5 zusätzlich die *Stakeholder*. Die Stakeholder haben ein Interesse an der Entwicklung des Produkts und demzufolge einen entscheidenden Einfluss auf den Prozess der Produktentwicklung und die daraus resultie-

renden Ergebnisse. Dabei können Stakeholder sowohl Teil des Handlungssystems (z.B. Entwickler, Produktmanager) als auch Teil der Systemumwelt sein. Wobei nach ALBERS et al. die Systemumwelt alles umfasst, was mit dem System der Produktentwicklung wechselwirkt, jedoch nicht Teil von Ziel-, Handlungs- und Objektsystem ist (z.B. Nutzer, Käufer, Gesellschaft) [ABK+16].

Wie in Bild 4-5 verdeutlicht, umfasst die Validierung die Aktivitäten Verifikation, Bewertung und Objektivierung, worüber eine Beziehung zwischen dem System der Produktentwicklung und den Stakeholdern hergestellt wird. Ausgangspunkt ist dabei die *Verifikation*, um über einen *Vergleich von Elementen des Objektsystems mit Elementen Zielsystems deren Konformität zu beurteilen* [AMB+15]. Darauf aufbauend werden im Rahmen einer *Bewertung* die Elemente des Objektsystems aus Stakeholder-Sicht untersucht. Die Bewertung ist dabei subjektiv und spiegelt die Wahrnehmung individueller Stakeholder wider, sie basiert jedoch auf einer überwiegend objektiven Analyse der Verifikationsergebnisse. Über die *Objektivierung* wird im Rahmen der Validierung untersucht, ob die Elemente des Zielsystems die Erwartungen der Stakeholder widerspiegeln, wobei gleichzeitig die Objektivität der Elemente des Zielsystems überprüft und erhöht wird. *Je objektiver die Ziele festgeschrieben sind, desto klarer ist die Ausgangslage für die Transformation in Objekte und desto besser können entstandene Objekte in Bezug auf das Zielsystem verifiziert werden* [AMB+15].

Diese Zusammenhänge zwischen Verifikation, Bewertung und Objektivierung ermöglichen die Beschreibung der Wechselwirkung zwischen den Stakeholdern und dem System der Produktentwicklung. ALBERS et al. betonen, dass die *Validität* [der entwickelten Artefakte] *nur über das Zusammenspiel aller beschriebenen Teilaktivitäten erreicht werden kann*, und einzelne Aktivitäten wie bspw. die Verifikation alleine dafür nicht ausreichend sind [ABK+16].

4.1.3 Kontinuierliche Validierung in der Produktgenerationenentwicklung

Auf Basis des zuvor beschriebenen Verständnisses der Validierung in der Produktentwicklung nach ALBERS und dem MBSE Verständnis nach HOLT & PERRY, beschreiben MANDEL et al. einen MBSE Ansatz für eine *kontinuierliche Validierung*, die in der *frühen Phase* der *Produktgenerationenentwicklung* beginnt [MBB+21].

> *„Die frühe Phase der Produktgenerationsentwicklung ist eine Phase im Entwicklungsprozess einer neuen Produktgeneration, die mit der Initiierung eines Projektes beginnt und mit einer bewerteten technischen Lösung endet, die schließlich das initiale Zielsystem hinsichtlich seiner wesentlichen Elemente abdeckt. Die zur technischen Lösung gehörende Produktspezifikation als Teil des Zielsystems enthält u.a. Informationen bzgl. der verwendeten Technologien und Subsysteme sowie deren Übernahme- und Neuentwicklungsanteile. Sie ermöglicht eine valide Bewertung des zu entwickelnden technischen Systems hinsichtlich der*

relevanten Parameter wie beispielsweise der Produzierbarkeit, der notwendigen Ressourcen oder des technischen und ökonomischen Risikos [ARB+17]“.

Das Modell der Produktgenerationenentwicklung adressiert in diesem Zusammenhang, dass Systeme in der industriellen Praxis überwiegend auf Basis von bereits vorhandenen Referenzprodukten entwickelt werden, um Risiken und Kosten zu minimieren [AHH+19]. Das betrifft die Entwicklung des Produkts, aber auch die zur Entwicklungszeit benötigten Validierungssysteme [ABR17; YBB+19]. Bild 4-6 veranschaulicht das in der Automobilindustrie etablierte Vorgehen anhand unterschiedlicher Produktgenerationen aus Sicht eines Zulieferer-Unternehmens. Dargestellt ist ein OBC (s. Kapitel 1.2) in drei aufeinander aufbauenden Produktgenerationen.

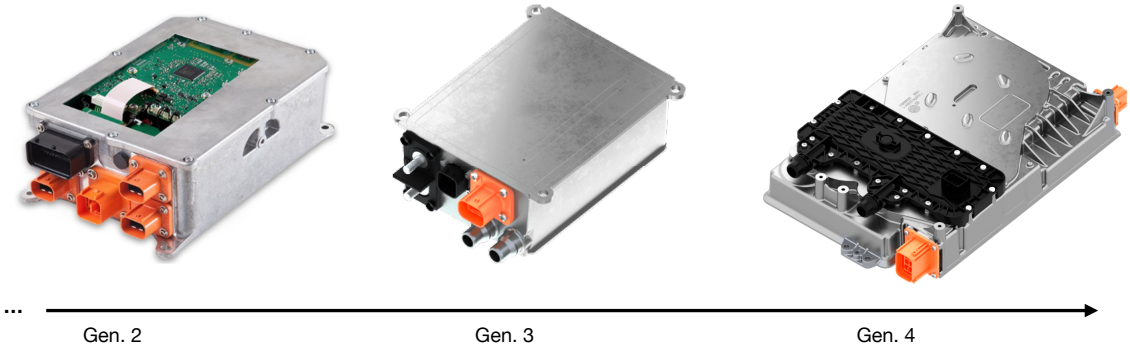


Bild 4-6: Auszug von Produktgenerationen des OBCs.

Bei der Entwicklung einer neuen Produktgeneration werden die wesentlichen Strukturen, Teilsysteme und Lösungsprinzipien von den jeweiligen Referenzprodukten übernommen, wobei hier eine Bewertung der Übernahme- und Neuentwicklungsanteile erfolgt,

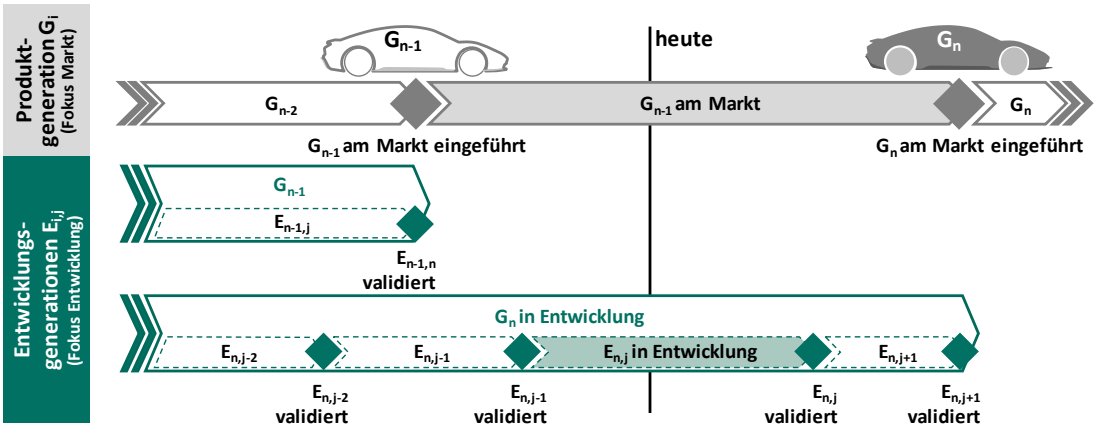


Bild 4-7: Entwicklungsgenerationen im Modell der Produktgenerationenentwicklung [AHH+19]

die durch eine Klassifizierung in eine Gestaltvariation (GV), Prinzipvariation (PV) oder eine Übernahmevariation (ÜV) strukturiert wird [ABW15]. Bild 4-7 zeigt das Prinzip

der Produktgenerationenentwicklung aus Sicht eines OEMs am Beispiel unterschiedlicher Fahrzeuggenerationen G_n. Dabei ist die Entwicklung der einzelnen Produkte durch Entwicklungsgenerationen gegliedert. Die Entwicklungsgeneration $E_{n,j}$ dient somit zur Einordnung der Entwicklungsaktivitäten für die als nächstes am Markt einzuführende Produktgeneration G_n [AHH+19].

Dieser Sichtweise folgend ist das Modell der Produktgenerationenentwicklung in Kombination mit einer kontinuierlichen und in der frühen Phasen beginnenden Validierung ein potentiell geeigneter Ansatz, um in realen Entwicklungsprojekten vorhandenes Wissen über Produktgenerationen hinweg wiederzuverwenden und neues Wissen zu generieren. Bild 4-8 veranschaulicht diese Konzepte und mögliche Vorteile durch deren Anwendung. Hier ist

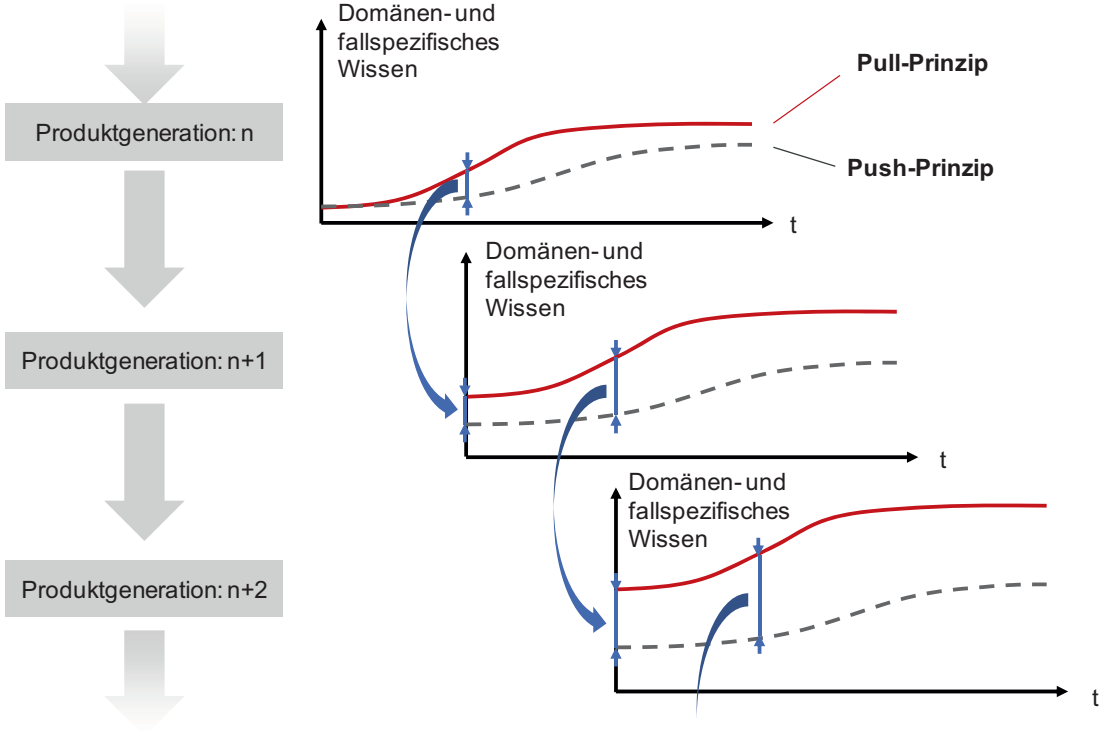


Bild 4-8: Schematische Darstellung des Vorteils, der durch die Anwendung des Pull-Prinzips der Validierung entsteht [AMB+15].

dargestellt, dass durch die frühe und kontinuierliche Validierung durchgängig Wissen aufgebaut wird und im Vergleich zu einer nachgelagerten Validierung in etablierten Prozessen (Push-Prinzip) das initiale domänen- und fallspezifische Wissen von Produktgeneration zu Produktgeneration ansteigt. ALBERS et al. begründen das dadurch, dass die Validierung bei der Anwendung des Pull-Prinzips zu einer durchgängigen, prozessbegleitenden Aktivität wird und den zuvor in Bild 4-5 skizzierten Validierungs- und Designzyklus miteinander verzahnt. Als einen weiteren wichtigen Punkt geben die Autoren an, dass die Parallelisierung aus primären und sekundären Entwicklungsaktivitäten möglich wird, wobei primäre Entwicklungsaktivitäten einen direkten Produktbezug haben und sekundäre Entwicklungsaktivitäten bswp. die Entwicklung von Testumgebungen umfassen [YBB+19]. Durch die Parallelisierung soll das Zielbild für die primären Entwicklungsaktivitäten kontinuierlich

eingeschränkt und erweitert werden. Da die Validierung von Beginn an mitgedacht und durchgeführt wird, sollen Aufwände, die nicht zur Wissensgenerierung beitragen, reduziert werden können. So soll bspw. der Aufwand für einen nachgelagerten Entwurf komplexer Testumgebungen reduziert werden können, da durch die Validierung die frühzeitige Identifikation ungeeigneter Lösungen unterstützt wird und somit auch die Variantenvielfalt benötigter Testumgebungen eingeschränkt werden kann [AMB+15; ABK+17; YBB+19].

MANDEL et al. erweitern diese Konzepte und unterstützen die frühe und kontinuierliche Validierung durch einen MBSE Ansatz [MBB+21], um die Wiederverwendung des Domänen- und fallspezifischen Wissens über modellbasierte Techniken zu fördern [MWB+20]. Die Autoren stellen fest, dass trotz der zentralen Bedeutung der Validierung innerhalb der Produktentwicklung diese in bestehenden MBSE Ansätzen nicht ausreichend berücksichtigt wird (vgl. [ABK+16] und [AMB+15]). Um diese Lücke zu schließen, schlagen die Autoren einen anwendungsorientierten MBSE Ansatz vor, der die Validierung in den Fokus setzt.

Auf Basis einer Literaturanalyse beschreiben MANDEL et al. zunächst, welche Begriffe und Informationen für die Validierung von technischen Systemen verwendet werden und wie diese miteinander in Beziehung stehen. Die Analyse und Konsolidierung der einzelnen Begriffe führt zu der in Bild 4-9 dargestellten Ontologie, worüber die vernetzten

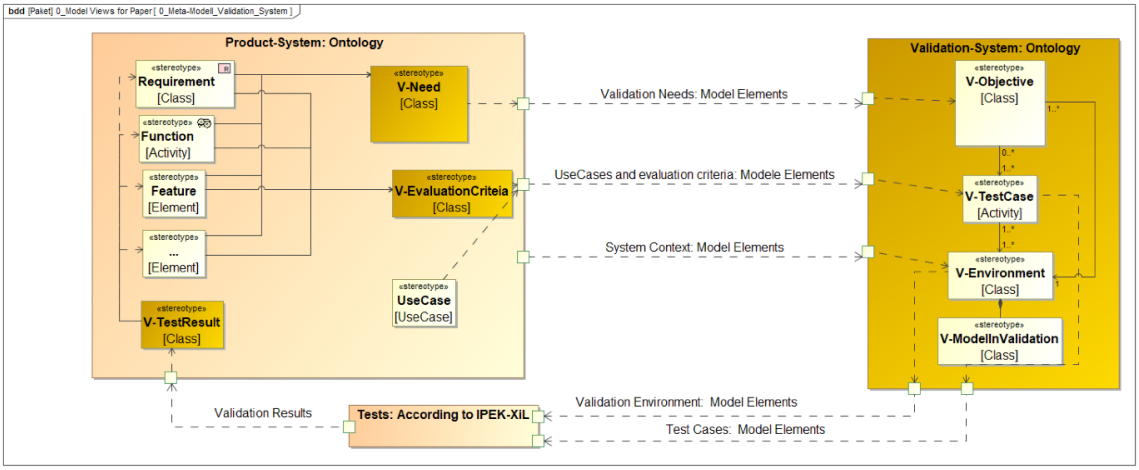


Bild 4-9: Auszug der Ontologie zur Unterstützung der kontinuierlichen Validierung [MBB+21].

Informationen in einer abstrahierten Form veranschaulicht werden. Wesentliche Elemente sind hier u.a. *Validierungsbedarfe* (V-Need), *Validierungsziele* (V-Objective), *Evaluationskriterien* (V-EvaluationCriteria), *Testfälle* (V-TestCase), *Testergebnisse* (V-TestResult), *Testumgebungen* (V-Environment) und das zu validierende *Modell des Testobjekts* (V-ModelInValidation). Über die Verwendung dieser Ontologie-Elemente ist die explizite Modellierung von notwendigen Eigenschaften der Validierungssysteme möglich, die wiederum entsprechend der Darstellung in Bild 4-9 mit den Eigenschaften des Produktsystems vernetzt werden können. Durch diese explizite Modellierung der Eigenschaften des Validierungssystems und der Vernetzung mit dem Produktsystem werden beide Systeme miteinander verzahnt, wodurch eine frühzeitige und integrative Modellierung beider Syste-

me adressiert wird [MBB+21].

Ergänzend zu der Ontologie beschreiben die Autoren einen methodischen Ansatz, der in dem Aktivitätsdiagramm in Bild 4-10 veranschaulicht ist. Ausgangspunkt der Metho-

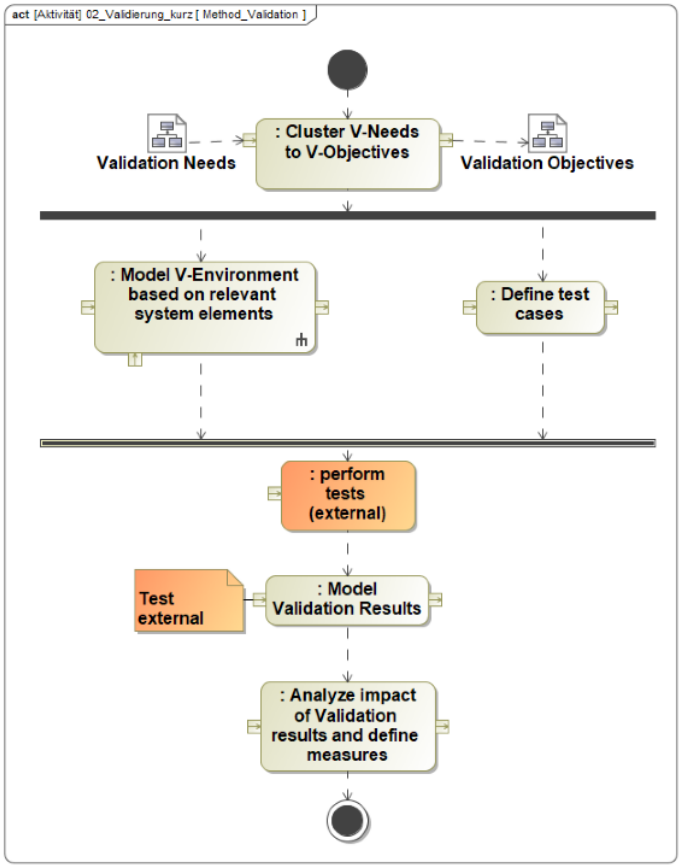


Bild 4-10: Methode zur Unterstützung einer kontinuierlichen Validierung [MBB+21].

de sind Validierungsbedarfe, die aus den Informationen des Produktsystems abgeleitet werden. Die erste Aktivität umfasst die Gruppierung der einzelnen Validierungsbedarfe zu Validierungszielen. Neben den Validierungszielen erfolgt eine Definition von Hypothesen und Evaluationskriterien über das Ontologie-Element V-EvaluationCriteria. Diese Kriterien sind zusammen mit den Validierungszielen die Basis für die Aktivitäten zur Formulierung von Testfällen und dem Entwurf der Testumgebungen. Die eigentliche Ausführung der Testfälle ist nicht Teil der Methode und wird durch externe Methoden beschrieben. Die Ergebnisse der Testdurchführung werden jedoch erfasst und mit den Elementen des Systemmodells vernetzt. Somit kann der Einfluss der Testergebnisse auf die zu Beginn identifizierten Validierungsbedarfe analysiert werden, was die Definition weiterer Entwicklungsschritte unterstützt [MBB+21].

4.1.4 System of Systems Engineering (SoSE)

Die zuvor beschriebenen Ansätze betrachten die Entwicklung eines Systems in seiner Umwelt. Das ZHO-Modell adressiert hierbei explizit eine iterative Entwicklung und einen

kontinuierlichen Abgleich des Objekt- und Zielsystems. Über die kontinuierliche Validierung in der Produktgenerationenentwicklung ist ein modellbasierter Ansatz beschrieben, der die Wiederverwendung von Wissen fördert und sekundäre Entwicklungsaktivitäten, wie die Entwicklung von Validierungssystemen, unterstützt.

Für die Entwicklung von *Advanced Systems* sind diese und vergleichbare Ansätze jedoch nicht ausreichend. Ausgangspunkt der Entwicklungsaktivitäten sind bei den oben genannten Ansätzen komplexe monolitische Systeme (z.B. Fahrzeug) und deren Subsysteme (z.B. OBC), die in einer definierten Systemumwelt eine Funktionalität bereitstellen. Wie oben beschrieben, umfasst die Systemumwelt nach der Darstellung in Bild 4-5 alles, was mit dem System der Produktentwicklung wechselwirkt, aber nicht Teil von Ziel-, Handlungs- und Objektsystem ist. Diese Abgrenzung ist jedoch nicht eindeutig einzuhalten, wenn für die Entwicklung von *Advanced Systems* einzelne Systeme unabhängig voneinander in unterschiedlichen Organisationen entwickelt werden, aber dennoch als Teil eines *Advanced Systems* eine integrierte und durch Stakeholder erlebbare Funktionalität realisieren (vgl. [HR18; KDB+19]).

Für den Kontext dieser Arbeit resultiert hieraus die Notwendigkeit, Anforderungs- und Testspezifikationen als Schnittstellenobjekte [WHK+20] zwischen den einzelnen an der Entwicklung beteiligten Handlungssysteme [ALE11] bzw. den Systemen der Marktleistungsentstehung [DAR+21] zu spezifizieren. Hierfür ist es zunächst notwendig, die Rolle eines zu entwickelnden Systems innerhalb eines *Advanced Systems* zu verstehen und einordnen zu können. Ein Bereich, der hierbei unterstützen kann, ist das *System of Systems Engineering* (SoSE) [NLF+15]. Gegenstand des SoSE ist das System of Systems (SoS), das die INCOSE wie folgt definiert:

> *„A system of systems is a system whose elements are managerially and/or operationally independent systems. These interoperating and/or integrated collections of constituent systems usually produce results unachieveable by the individual systems alone [DRF+15, S.8].“*

Betrachtet man bspw. die zu Beginn eingeführte Funktion *Timer-Charging* (s. Kapitel 1.2), wird deutlich, dass die Ladefunktionalität ein integraler Bestandteil einer übergeordneten *Smart-Charging* Funktionalität ist (s. Bild 1-1). Die Koordinierung der Ladefunktionalität innerhalb des Fahrzeugs erfolgt dabei durch das Subsystem OBC. Das Fahrzeug kann wiederum als ein Constituend System (CS) in einem SoS Kontext betrachtet werden, das mit dynamisch wechselnden Bediengeräten, Ladeinfrastrukturen und weiteren, den Ladevorgang direkt oder indirekt beeinflussenden CSs interagiert [KDB+19; SL20]. Im Gegensatz zu etablierten hierarchisch strukturierten OEM (System Fahrzeug) - Zulieferer (Subsystem OBC) Beziehungen entstehen dynamischere Entwicklungsnetzwerke [DAR+21, S. 102 f.], da bspw. die Ladefunktionalität von externen Bediengeräten konfiguriert wird, die von unterschiedlichen Entwicklungspartnern bereitgestellt werden.

Durch die Vernetzung des Fahrzeugs mit weiteren CSs entsteht dabei eine Vielzahl von Anwendungsfällen, die dem Konzept der kontinuierlichen Validierung folgend zu analysieren

sind, um notwendige Entwicklungsschritte der beteiligten Entwicklungspartner definieren und koordinieren zu können. Dabei ist es erforderlich, die jeweilige Entwicklungspartnerschaft zu klassifizieren. Einen Ausgangspunkt hierfür beschreibt MAIER durch die in Bild 4-11 dargestellten SoS-Typen mit unterschiedlichen Beziehungen zwischen den CSs (A,B,C) und koordinierenden Organisationen (SoSE Team) [Mai99] (vgl. [DB08; NLF+15]).

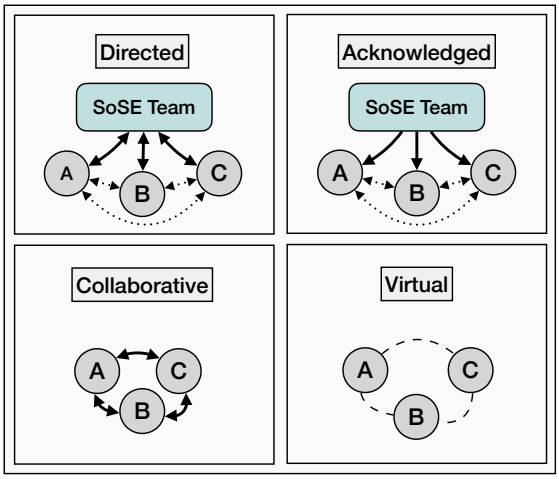


Bild 4-11: SoS-Typen Directed*,* Acknowledged*,* Collaborative *und* Virtual *nach [Mai99] und [DB08]*

Nach dieser Darstellung kann ein *Directed* SoS dadurch charakterisiert werden, dass es für einen spezifischen Zweck entwickelt wird. Der oben genannten SoS Definition folgend, können die einzelnen Systeme zwar unabhängig voneinander betrieben werden, aber es existiert ein übergeordnetes SoSE-Team zur Identifikation, Definition und Koordinierung von gemeinsamen, das SoS betreffenden Zielen. Im Gegensatz dazu identifiziert und definiert ein SoSE-Team eines *Acknowledged* SoS zwar übergeordnete SoS Ziele, die einzelnen CSs verfolgen jedoch eigene Ziele und behalten ihre unabhängige Steuerung. Dabei basiert die kontinuierliche Weiterentwicklung der SoS Funktionalität auf der Kollaboration der einzelnen CSs. Das *Collaborative* SoS wird verstanden als ein SoS, das kein zentrales SoS Management besitzt. Die Realisierung von SoS Zielen basiert hier auf einer freiwilligen Kollaboration zwischen den beteiligten CSs. Die letzte in Bild 4-11 skizzierte Ausprägung ist das *Virtual* SoS. Hier existiert kein zentrales SoS Management, und es gibt auch keine explizit definierten gemeinsamen Ziele der CSs. Hierdurch ist der Aufbau, der die Funktionalität bereitstellt, schwer zu identifizieren, und es besteht ein hoher Grad an emergentem Verhalten [NLF+15; WGW+21].

Die genannten Ausprägungen können eine Hilfestellung und Ausgangspunkt für die Entwicklung eines SoS oder für die Entwicklung eines Systems als Teil eines SoS sein. In der Praxis ist es jedoch nicht möglich, eine eindeutige Einteilung vorzunehmen, sodass reale Systeme durch die Kombinationen der 4 Typen zu beschreiben sind [NL18; NLF+15]. Das im Kontext dieser Arbeit verwendete Beispiel lässt sich überwiegend als *Acknowledged* SoS einordnen (vgl. [WGW+21]).

4.2 Ansätze für die modellbasierte Spezifikation und Analyse von funktionalen Anforderungen

Ist die Rolle eines Systems innerhalb eines *Advanced Systems* beschrieben, sind innerhalb einer Organisation zahlreiche Anwendungsfälle zu erfassen, die einen Rahmen für die Systementwicklung eines spezifischen Systems bereitstellen. Zur Ausgestaltung von Anwendungsfällen sind innerhalb des RE Szenarien etabliert. JARKE et al. [JBC98] und SUTCLIFFE [Sut03] zeigen jedoch, dass der Begriff Szenario unterschiedlich verstanden und verwendet wird. Auf der einen Seite werden Szenarien verwendet, um Erfahrungen aus der realen Welt über informale Beschreibungen wiederzugeben. Auf der anderen Seite existieren formalere Beschreibungen, die bspw. mit Hilfe von Modellen (s. Kapitel 3.2.3) eine Abfolge von Ereignissen beschreiben und damit ein Systemverhalten spezifizieren (vgl. [Sut03]). Der Duden definiert ein Szenario als eine *„(...) hypotetische Aufeinanderfolge von Ereignissen, die zur Beachtung kausaler Zusammenhänge konstruiert wird“* [Dud22], was sowohl eine abstrakte Beschreibung in natürlicher Sprache, als auch eine modellbasierte Spezifikation einschließt. Für die modellbasierte Beschreibung von Szenarien ist die Verwendung von Sequenzdiagrammen etabliert [Bal09, S. 332], wobei Sequenzdiagramme in unterschiedlichen Ausprägungen existieren. Die für diese Arbeit relevanten Ausprägungen sind im folgenden beschrieben.

4.2.1 UML Sequenzdiagramme

Als Teil der *Unified Modeling Language*[2] (UML) sind Sequenzdiagramme eine Möglichkeit für die Modellierung von Interaktionen zwischen unterschiedlichen Kommunikationspartnern. Die UML Sequenzdiagramme sind zurückzuführen auf die von der *International Telecommunication Union* (ITU) standardisierten *Message Sequence Charts* (MSCs) [ITU] (vgl. [HM03a, S. 13]). Die wesentlichen Elemente dieser Sequenzdiagramme sind in Bild 4-12 dargestellt. Basis für Modellierung von Interaktionen sind die beteiligten Kommunikationspartner, die in der Softwaretechnik oft durch Objekte von Klassen dargestellt werden. In diesem Fall sind die Objekte als Rechtecksymbole dargestellt, wobei der Name jedes Kommunikationspartners dem Klassennamen entspricht, dem ein Doppelpunkt vorangestellt ist. Der grundlegende Aufbau des Diagramms ist unterteilt in eine horizontale und vertikale Ebene. Die horizontale Ebene ermöglicht eine Modellierung der einzelnen Interaktionen durch den Austausch von Nachrichten zwischen den Kommunikationspartnern. Die vertikale Ebene repräsentiert den zeitlichen Verlauf der Interaktionen. Der zeitliche Verlauf ist dabei durch Lebenslinien an die einzelnen Kommunikationspartner gebunden. Hierbei führt der Empfang einer Nachricht zum Aufruf einer Operation. Die Zeitspanne für die Ausführung einer bestimmten Operation eines Kommunikationspartners wird als Aktionssequenz bezeichnet, die in diesem Beispiel mit dem Senden einer Rückantwort beendet wird [Bal09, S. 333].

---

[2] https://www.omg.org/spec/UML/ zuletzt abgerufen am 19.9.2022

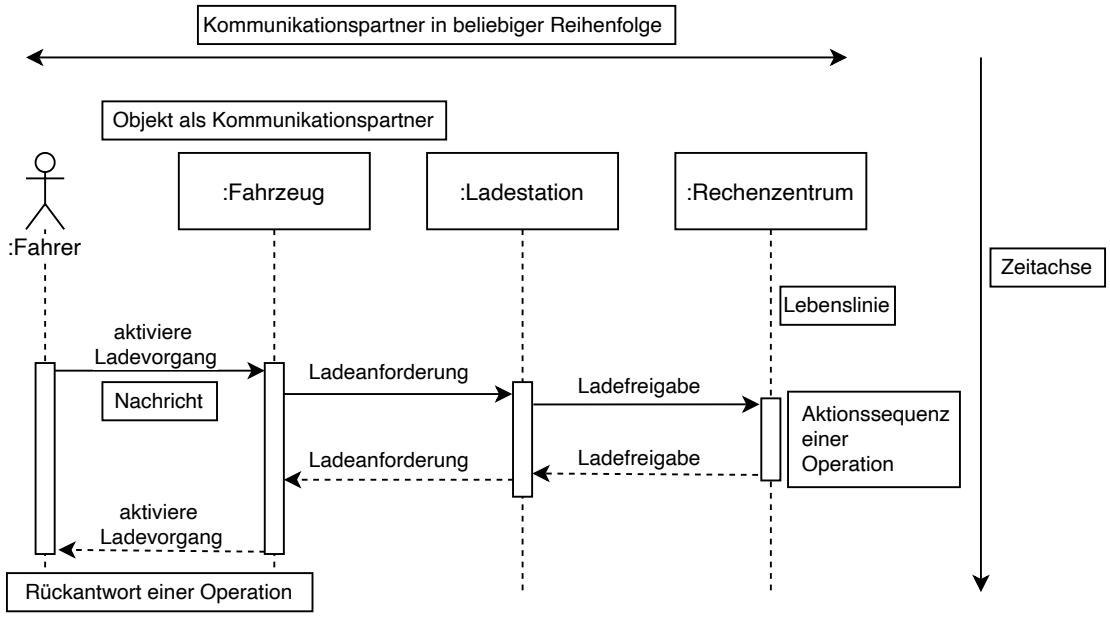


Bild 4-12: Wesentliche Elemente der Notation für UML Sequenzdiagramme nach [Bal09, S. 333]

Diese grafische Darstellung ermöglicht die Modellierung des Systemverhaltens. Das in Bild 4-12 dargestellte Szenario zeigt bspw., dass ein Akteur (Fahrer) einen Ladevorgang aktiviert. Die Aktivierung des Ladevorgangs führt im Kontext des Kommunikationspartners Fahrzeug zum Aufruf einer Operation. Die Operation sendet eine Nachricht an eine Ladestation, die wiederum mit einem Rechenzentrum interagiert. Der Empfang der Rückantworten vom Rechenzentrum, der Ladestation und schließlich des Fahrzeugs beenden das Szenario, das durch die Aktivierung des Ladevorgangs durch den Fahrer angestoßen wurde.

4.2.2 Modale Sequenzdiagramme

Die grafische Modellierung des Systemverhaltens auf Basis der MSCs ist weit verbreitet (vgl. [MW11]). DAMM & HAREL kritisieren jedoch, dass die Aussagekraft dieser MSCs begrenzt ist und für eine umfassende Spezifikation des Systemverhaltens nicht ausreicht [DH01]. Als eine wesentliche Limitierung geben die Autoren an, dass nicht ausgedrückt werden kann, was das implementierte System tatsächlich ausführt (vgl. [HM03a, S. 15]). Demnach kann die in Bild 4-12 dargestellte Interaktion zwar ein mögliches Systemverhalten darstellen, sie zeigt jedoch nicht, wie sich das System tatsächlich verhält. Möglicherweise kann das Rechenzentrum die Ladefreigabe nicht erteilen oder die Ladestation sendet im Fehlerfall keine Rückantwort der Ladeanforderung. Somit führt das implementierte System eine Reihe weiterer Aktionen aus, die in dem Sequenzdiagramm nicht berücksichtigt werden, jedoch das im Sequenzdiagramm beschriebene Verhalten beeinflussen können. Nach HAREL & MARELLY geben UML Sequenzdiagramme folglich *eine Reihe von einfachen Bedingungen für die partielle Reihenfolge möglicher Ereignisse*

in einer möglichen Systemausführung an [HM03a, S. 59].

Wie in Kapitel 3.2.3 beschrieben, wird die Anforderungsmodellierung in dieser Arbeit als verkürzte Abbildungen von Originalen (informale Spezifikation in natürlicher Sprache) verstanden, die zu einem bestimmten Zeitpunkt für eine spezifische Aufgabe verwendet werden. Demnach ist es weder notwendig noch sinnvoll das gesamte implementierte System zu modellieren. Dennoch ist es für eine korrekte Spezifikation notwendig, zwischen möglichem und notwendigem Systemverhalten unterscheiden zu können. Ebenso muss es möglich sein, nicht erwünschtes Systemverhalten zu spezifizieren. Ist bspw. die Ladestation in einem Fehlerzustand, darf der Ladevorgang nicht aktiviert werden. Zudem ist es insbesondere in einer iterativen Entwicklung notwendig, Szenarien schrittweise zur Spezifikation hinzuzufügen, wobei die einzelnen Szenarien voneinander abhängig sein können und sich ggf. gegenseitig beeinflussen, was durch UML Sequenzdiagramme nicht berücksichtigt wird (vgl. [HM03a, S. 15]).

Um diese Beschränkungen zu adressieren, beschreiben DAMM & HAREL die sogenannten *Live Sequence Charts* (LCSs) [DH01]. Die Basis für die Spezifikation von LSCs sind die zwei Diagrammtypen *universal LSCs* und *existential LSCs*. Wie in Bild 4-13 werden diese beiden Typen grafisch durch eine durchgezogene bzw. gestrichelte Linie umfasst und dadurch unterschieden. *Universal LCSs* können dazu verwendet werden, das System-

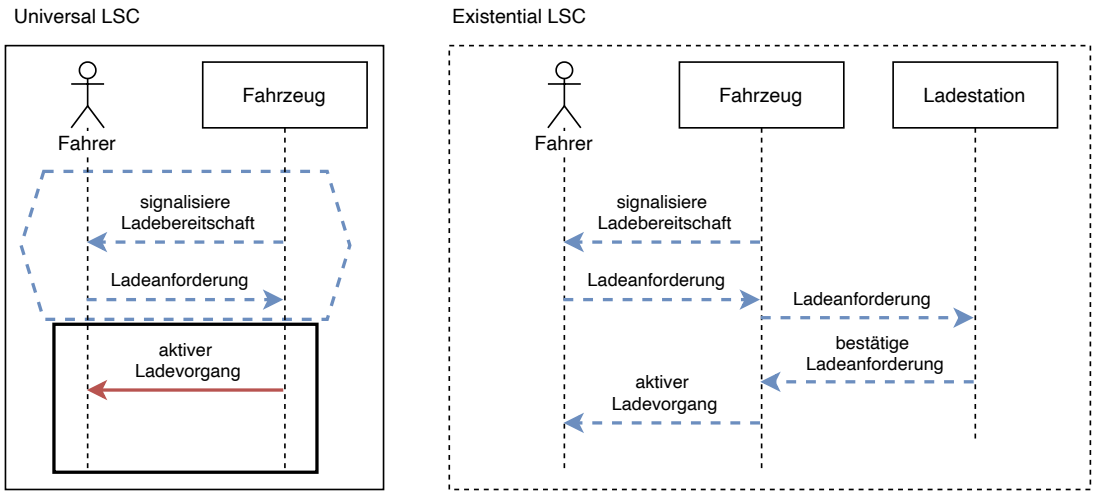


Bild 4-13: Grundlegender Aufbau von universal LSCs *und* existential LSCs *(vgl. [DH01]).*

verhalten für alle möglichen Systemausführungen festzulegen bzw. einzuschränken. Es besteht aus einem *prechart* und einem *main chart*. Das *prechart* ist durch das gestrichelte Hexagon gekennzeichnet und definiert ein Szenario, das zu einer bestimmten im *main chart* definierten Systemreaktion führt. Das Beispiel in Bild 4-13 zeigt, wenn das Fahrzeug die Ladebereitschaft signalisiert und der Fahrer eine Ladeanforderung sendet (*prechart*), *muss* das System mit einem aktiven Ladevorgang antworten (*main chart*). Dies wird durch den roten durchgezogenen Pfeil gekennzeichnet, was im Kontext der LSCs über den Begriff *hot* definiert wird. Im Gegensatz dazu sind die im *prechart* spezifizierten Aktionen als *cold*

definiert, wodurch ausgedrückt wird, dass dieses Verhalten auftreten *kann*. Auf diese Weise kann über *prechart* und *main chart* eine Aktion-Reaktion Beziehung spezifiziert werden, wobei diese für *universal LSCs* für alle möglichen Systemausführungen einzuhalten ist [DH01]. Im Gegensatz zu einem *universal LSC*, das für jede Systemausführung gültig ist, zeigen *existential LSCs* ein Verhalten, das für mindestens einen Systemdurchlauf zutreffen muss, wobei die Reihenfolge der einzelnen Aktionen nicht relevant ist (vgl. [HM03a, S. 61]).

Ausgehend von den LSCs erweitern HAREL & MAOZ die Semantik der UML Sequnzdiagramme, um so die formale Verifikation, Synthese und szenariobasierte Ausführung von Spezifikationen über den weit verbreiteten UML Standard zu ermöglichen [HM08]. Auf Basis dieser MSDs existieren zahlreiche methodische Ansätze und Werkzeuge, insbesondere für die Anforderungsanalyse in der frühen Entwicklungsphase: GREENYER erweitert MSDs, um die Modellierung von Echtzeitanforderungen und Annahmen für das Systemumfeld spezifizieren zu können. Zusätzlich ist eine Methode für die Identifikation von Inkonsistenzen in szenariobasierten Spezifikationen mechatronischer Systeme beschrieben [Gre11]. Dieser Ansatz ist über ein Werkzeug für die automatisierte Analyse von Anforderungen anwendbar [BGP+13]. Eine erfolgreiche Anwendung in der Automobilindustrie ist in [BGH+14] beschrieben. FOCKEL beschreibt einen Ansatz für die Spezifikation von eindeutigen und konsistenten sicherheitsrelevanten Anforderungen [Foc18], motiviert durch den in der Automobilindustrie relevanten ISO26262 Standard (vgl. Kapitel 3.1.2). Die formale Spezifikation mittels MSDs wird dabei über einen Musterkatalog unterstützt [FHK+16]. Ebenfalls über die Anwendung und Erweiterung von MSDs beschreibt HOLTMANN einen Ansatz für den Übergang von multidisziplinären Systemmodellen hin zu Software-Anforderungsmodellen und legt dabei einen besonderen Fokus auf Echtzeitanforderungen [Hol19].

4.2.3 Textuelle Modellierung von Szenarien

Basierend auf den zuvor beschriebenen LSCs/MSDs evaluieren GREENYER et al. eine formale szenariobasierte Spezifikationstechnik über die Anwendung eines Werkzeugs in der Automobilindustrie [GHM+15]. Hier wird am Beispiel einer realen Anwendung deutlich, dass Anforderungen erfolgreich modelliert werden können. Bei der Modellierung sehen die Autoren einen Vorteil darin, dass Entwickler dazu angehalten werden, Anforderungen präzise zu spezifizieren. Ein weiterer Vorteil wird darin gesehen, dass die Realisierbarkeit des entstandenen Anforderungsmodells automatisiert überprüft werden kann. Die beschriebene Fallstudie veranschaulicht, dass so Inkonsistenzen zwischen Anforderungen und implizite Annahmen in der Spezifikation aufgedeckt werden konnten [GHM+15]. Jedoch wird auch deutlich, dass die grafische Modellierung auf Basis eines UML Editors sehr zeitaufwändig ist. Für eine bessere Editierbarkeit und eine leichtere Versionsverwaltung der Modelle schlagen die Autoren daher vor, eine textuelle Anforderungsmodellierung zu verwenden. LIEBEL & KNAUSS untersuchen die Anforderungsmodellierung im Kontext des SE und kommen zu dem gleichen Ergebnis:

> *„To cope with the effort to maintain models over time, study participants suggest to rely on text-based notations that bring the models closer to developers and allow integration into existing software development workflows [LK23].“*

Behavioral Programming (BP)

Ein Ansatz, der eine textuelle Modellierung ermöglicht, basiert auf dem *Behavioral Programming* (BP) Paradigma [HMW+11; HMW12]. BP verwendet die zuvor beschriebenen LSC/MSD-Konzepte und ermöglicht eine szenariobasierte Programmierung. Eine ausführbare Software wird hier als *behavioral application* bezeichnet, die aus unabhängigen *behavioral threads* (*b-threads*) besteht. Bild 4-14 veranschaulicht das Zusammenspiel einzelner b-threads auf Basis eines *Publish/Subsribe-Protokolls*, bei dem Nachrichten zwischen den *b-threads* gesendet und empfangen werden. Hierbei lässt sich ein Ausführungszyklus grundsätzlich in 4 Schritte unterteilen:

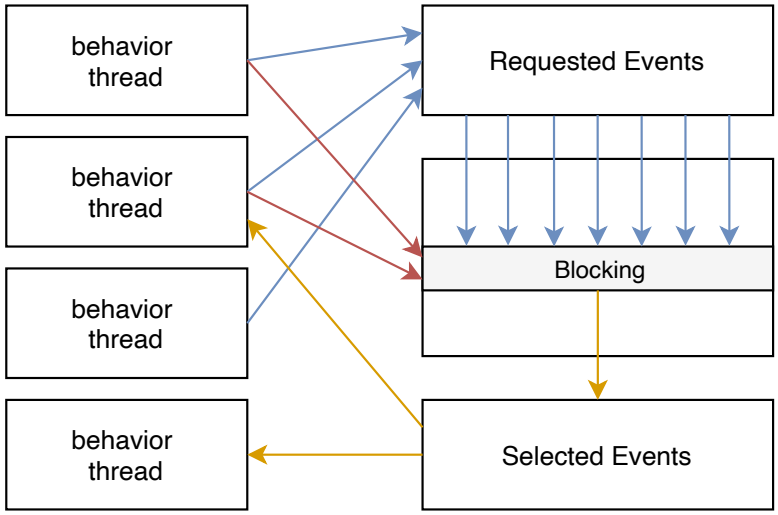


Bild 4-14: Schematische Darstellung des Ausführungszyklus von b-threads unter Verwendung eines Publish/Subscribe-Protokolls [HMW12].

1) Zunächst werden alle *b-threads* ausgeführt und durchlaufen den Programmcode bis ein Synchronisationspunkt erreicht ist. An diesem Synchronisationspunkt kann jeder *b-thread* die Ausführung von Ereignissen (Events) anfordern, auf die Ausführung bestimmter Events warten, oder die Ausführung bestimmter Events blockieren. In Bild 4-14 ist das Anfordern von Events durch die blauen Pfeile gekennzeichnet. Die roten Pfeile verdeutlichen das Blockieren von Events und die gelben Pfeile zeigen an, in welchen b-threads auf die Ausführung bestimmter Events gewartet wird.

2) Im zweiten Schritt wählt der Publish/Subscribe-Algorithmus ein angefordertes und nicht blockiertes Event aus.

3) Anschließend werden die *b-threads*, die die Ausführung des zuvor ausgewählten Events angefordert haben, oder die auf dieses Event warten, über die Auswahl des

Events informiert.

4) Im letzten Schritt wird die Ausführung des Programmcodes in allen zuvor benachrichtigten *b-threads* fortgesetzt, bis diese den nächsten Synchronisationspunkt erreichen, an dem wiederum Events angefordert und blockiert werden, bzw. auf die Ausführung von Events gewartet werden kann.

Das Ziel dieses Ansatzes ist eine inkrementelle und szenariobasierte Programmierung reaktiver Systeme, wohingegen LSCs und MSCs die Erstellung von ausführbaren grafischen Spezifikationen dieser Systeme adressieren.

Über die zuvor beschriebenen Ausführungsschritte soll dabei eine Programmierung möglichst nah an der natürlichen Formulierung und Spezifikation von Anforderungen sein. Der Sichtweise von HAREL et al. folgend, werden Anforderungen üblicherweise inkrementell beschrieben, wobei durch das Hinzufügen von Anforderungen zu einer Spezifikation auch eine nachträgliche Einschränkung des Systemverhaltens möglich ist. BP ermöglicht aus diesem Grund durch inkrementelles Hinzufügen neuer *b-threads* die Umsetzung neuer Anforderungen, wobei die einzelnen *b-threads* durch die enthaltenen Events und

```
class AddHotThreeTimes extends BThread {
    public void runBThread() {
        for (int i = 1; i <= 3; i++) {
            bp.bSync( addHot, none, none );
        }
    }
}

class AddColdThreeTimes extends BThread {
    public void runBThread() {
        for (int i = 1; i <= 3; i++) {
            bp.bSync( addCold, none, none );
        }
    }
}

class Interleave extends BThread
    public void runBThread() {
        while (true) {
            bp.bSync( none, addHot, addCold );
            bp.bSync( none, addCold, addHot );
        }
    }
}
```

Event log of
the coordinated run

```
addHot
  addCold
addHot
  addCold
addHot
  addCold
```

Bild 4-15: Vereinfachtes Beispiel zur Veranschaulichung der Synchronisierungen zwischen b-threads [HMW12].

den Synchronisationsmechanismus miteinander in Beziehung stehen und so ein integriertes Systemverhalten realisieren. Es ist somit möglich, durch Hinzufügen von *b-threads* bestehendes Systemverhalten zu ergänzen oder einzuschränken, ohne die gesamte Implementierung anpassen zu müssen [HMW12] [HM12]. HAREL et al. verdeutlichen diese Vorgehensweise an dem in Bild 4-15 Beispiel. Dieses vereinfachte Beispiel zeigt zwei Wasserhähne, die warmes oder kaltes Wasser bereitstellen. Die Reihenfolge wird durch die *behavioral application* definiert. Diese besteht aus drei *b-threads*. Im ersten *b-thread*

wird an dem Synchronisationspunkt warmes Wasser angefordert. Der zweite *b-thread* fordert kaltes Wasser an. Durch die jeweils enthaltene *for*-Schleife würde jeweils drei mal warmes und drei mal kaltes Wasser hinzugefügt. Da jedoch durch den dritten *b-thread* auf die Events *addHot* bzw. *addCold* gewaretet und das jeweils andere Event blockiert wird, erfolgt die im Bild dargestellte verschachtelte Ausführung. Auf diese Weise beeinflusst der dritte *b-thread* das durch die beiden vorangegangenen *b-threads* definierte Verhalten [HMW12].

Scenario Modeling Language (SML)

Ausgehend von den LSCs/MSCs [DH01; HM08] sowie dem BP Paradigma [HMW+11; HMW12] beschreibt GREENYER die *Scenario Modeling Language* (SML) und das zugehörige Werkzeug SCENARIOTOOLS [GGG+17]. Die Motivation für die Entwicklung der SML sind die oben genannten Punkte, die eine bessere Anwendbarkeit für die textuelle Modellierung von Szenarien erwarten lassen. Als Erweiterung der LSCs und basierend auf [Gre11] unterstützt die SML die Modellierung von sogenannten *guarantee scenarios* und *assumption scenarios*. *Guarantee scenarios* spezifizieren, wie sich das System verhalten muss und über die *assumption scenarios* kann festgehalten werden, wie sich die Systemumwelt verhält [Gre11; GGK+16].

Bild 4-16 zeigt exemplarische zwei Szenarien eines Car-to-X Systems. Dieses System

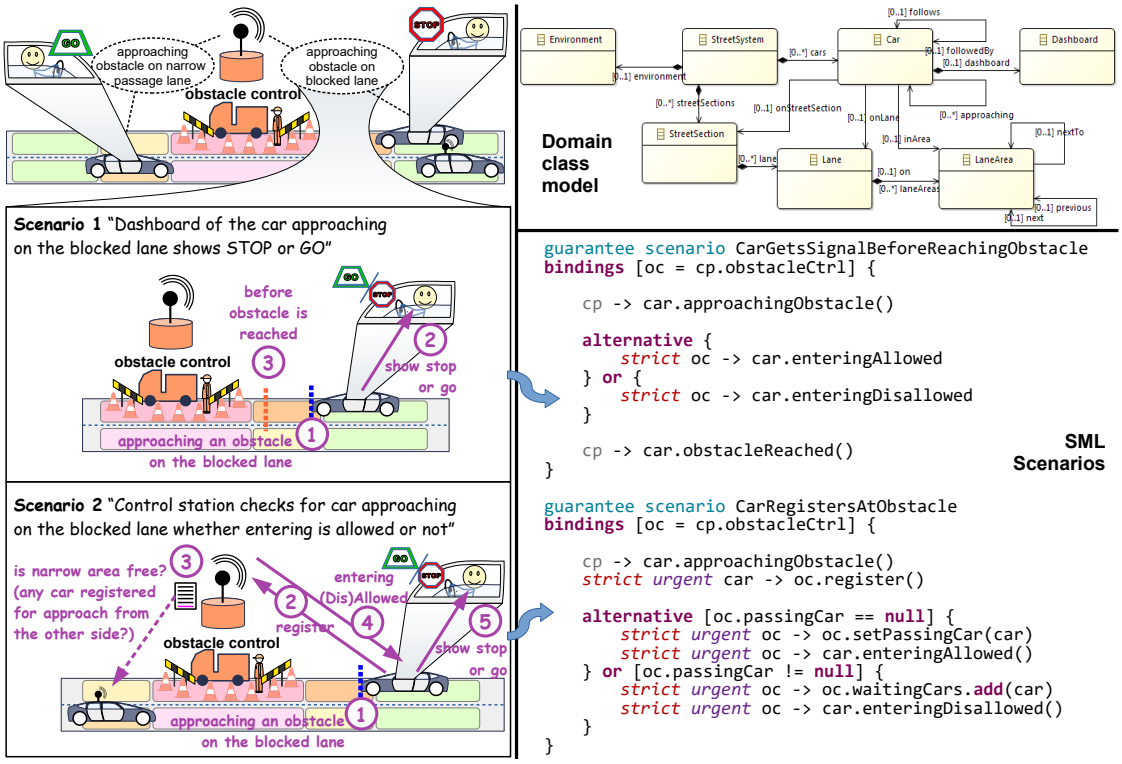


Bild 4-16: Zwei Szenarien eines Car-to-X Systems [GGK+16; Gre21].

hat die Aufgabe, Fahrzeuge zu koordinieren, die sich einem Hindernis nähern, um so ein sicheres Passieren zu ermöglichen. Auf der linken Seite sind diese zwei Szenarien visuali-

siert. Über das Klassenmodell (oben rechts) wird die SML-Spezifikation typisiert. Unten rechts sind die zwei *guarantee scenarios* dargestellt, die das Systemverhalten spezifizieren [GGK+16]. Das erste Szenario betrachtet den Fall, dass sich ein Fahrzeug dem Hindernis auf der blockierten Fahrbahn nähert. Noch bevor das Fahrzeug das Hindernis erreicht, muss dem Fahrer signalisiert werden, ob das Passieren des Hindernisses erlaubt ist oder nicht. Das zweite Szenario erweitert das im ersten Szenario beschriebene Verhalten. Hier wird der Fall berücksichtigt, dass sich ein Fahrzeug, das sich dem Hindernis auf der blockierten Fahrbahn nähert, bei der Kontrollstation anmelden muss. Die Kontrollstation prüft, ob sich bereits ein anderes Fahrzeug aus der Gegenrichtung angemeldet hat. Wenn dies der Fall ist, darf das erste Fahrzeug das Hindernis nicht passieren. Wie GREENYER et al. beschreiben, erweitert das zweite Szenario das erste Szenario und determiniert die Entscheidung, ob das Passieren des Hindernisses erlaubt ist oder nicht. Für eine weitere Spezifikation des Car-to-X Systems sind weitere Szenarien hinzuzufügen [GGK+16]. An diesem Beispiel wird deutlich, wie der Ansatz die iterative Modellierung von Szenarien ermöglicht. Dieses Vorgehen entspricht der Art, wie Anforderungen an Systeme üblicherweise kommuniziert werden. Somit forciert dieser Ansatz eine intuitive Modellierung (vgl. [HMW12; HM12]).

4.2.4 Scenario Modeling Language for Kotlin (SMLK)

Wie oben beschrieben, kann eine textuelle Modellierung für reale Anwendungen geeigneter sein, was durch die zuvor beschriebene SML adressiert wird. Dennoch ist trotz der signifikanten Arbeit zur Entwicklung der Modellierungssprache und zugehörigen Werkzeugen keine Akzeptanz der textuellen Modellierung innerhalb des RE in der Automobilindustrie und verwandten Branchen zu erkennen [LTK19; Gre21; LK23].

Bezogen auf die SML und das Werkzeug SCENARIOTOOLS identifiziert GREENYER drei wesentliche Einschränkungen, die eine Akzeptanz behindern [Gre21]:

1) Die Programmierung innerhalb der Szenarien ist aufwendig. Bspw. kann das Filtern einer Liste oder die Verwendung komplexer Abfragen notwendig sein, um komplexes Systemverhalten zu beschreiben. Dies kann zwar mit SCENARIOTOOLS abgebildet werden, erfordert aber den Aufruf von externen Methoden.

2) Die Integration von SCENARIOTOOLS mit externen Frameworks, die für den jeweiligen Kontext relevant sein können, ist zwar möglich, erfordert aber eine umständliche Entwicklung von Integrationsschichten.

3) Die Ausführung von SML Spezifikationen ist rechen- und speicherintensiv, da sie auf dem *Eclipse Modeling Framework* (EMF)[3] und der Interpretation von SML/EMF Modellen beruht, was insbesondere bei umfangreichen Modellen zu einer langsamen Ausführung führen kann.

Um diese Punkte zu adressieren, schlägt GREENYER die Scenario Modeling Language for

[3] https://www.eclipse.org/modeling/emf/, zuletzt abgerufen am 29.11.2022

Kotlin (SMLK) vor. SMLK nutzt mehrere Konzepte der Programmiersprache Kotlin, wie z.B. Co-Routinen, um Szenarien effizient zu programmieren. Ebenso unterstützt Kotlin die Realisierung von internen *Domain Specific Languages* (DSLs) und erleichtert so die Umsetzung einer szenariobasierten Modellierung. Zudem lassen sich externe Frameworks ohne zusätzlichen Aufwand integrieren (vgl. [Gre21]). Eine erste Anwendung dieser Modellierungssprache im Automobilbereich ist in [WGK19] beschrieben.

SMLK basiert auf den oben beschriebenen LSCs [DH01] und dem BP Paradigma [HMW12]. Analog zu einer *behavioral application* des BP werden SMLK Szenarien in einer *scenario specification* zusammengefasst, die als ein *scenario program* ausführbar ist. Diese Ausführung basiert dabei ebenfalls auf einem zentralen *publish/subscribe* Mechanismus. Wird die *scenario specification* ausgeführt, werden die Szenarien auch hier bis zu einem Synchronisationspunkt durchlaufen. An diesem Punkt kann, entsprechend dem BP Paradigma, über den Aufruf einer *sync* Methode entweder die Ausführung von Events angefordert werden, oder es kann auf spezifische Events gewartet werden ohne deren Ausführung explizit anzufordern. Ebenso können über die *sync* Methode bestimmte Events blockiert werden. Neben einem direkten Aufruf der *sync* Methode ist die Anforderung von Events bzw. das Warten auf Events über `request` und `waitFor` möglich. Besteht die Notwendigkeit, Events über mehrere Synchronisationspunkte hinweg zu blockieren, können diese Events zu der Auflistung der `forbiddenEvents` hinzugefügt werden. Zudem umfasst die Liste der `interruptingEvents` alle Events, durch die die Ausführung eines Szenarios unterbrochen werden kann.

Neben der Verwendung von einfachen Events für die Modellierung von Szenarien unterstützt SMLK sogenannte *Object Events*. Die *Object Events* ermöglichen die Modellierung von Signalen, die zwischen Komponenten oder Objekten ausgetauscht werden und dabei die Eigenschaften der Komponente oder des Objekts ändern.

Die grundsätzlichen Mechanismen für die Anwendung der SMLK sind im folgenden mit Hilfe des in Bild 4-17 dargestellten Beispiels verdeutlicht. Dargestellt ist die Inter-

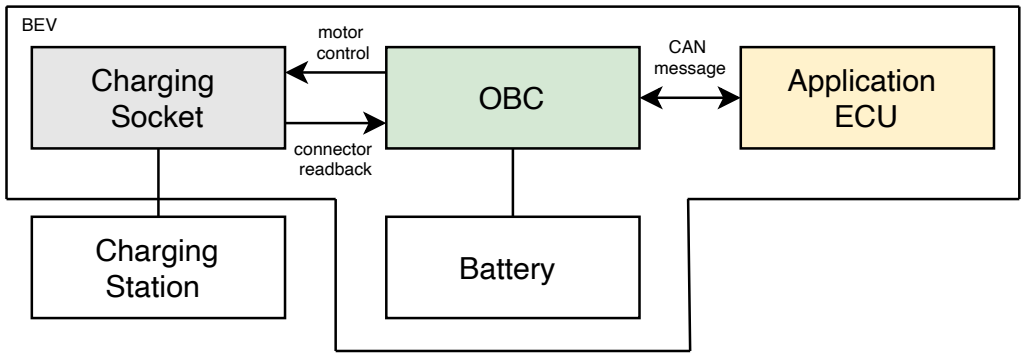


*Bild 4-17: Steuergeräte eines Elektrofahrzeugs (*Battery Electric Vehicle - BEV*) zur Realisierung der Funktion* Plug Interlock *[WJK+20].*

aktion zwischen Steuergeräten innerhalb des Elektrofahrzeugs zur Verriegelung eines Ladesteckers (s. Funktion *Plug Interlock* des zu Beginn eingeführten Beispiels in Kapi-

tel 1.2). Zwischen den Steuergeräten werden dafür Signale ausgetauscht. Zwischen dem OBC und der *Application ECU* als Teil einer CAN Nachricht und zwischen dem OBC und der Ladedose des Fahrzeugs über die direkte Ansteuerung eines Motors. Der SMLK Programmcode in Listing 4.1 und 4.2 abstrahiert von diesen Signalen und modelliert eine mögliche Interaktion zwischen den Steuergeräten. Zunächst ist jedes Steuergerät über eine Klasse beschrieben. Die Klasse `OBC` (Zeile 3-9 in Listing 4.1) beinhaltet bspw. drei Events, von denen zwei Events als *Object Events* bezeichnet werden und entsprechend bei ihrer Ausführung die Werte der in Zeile 2 und 3 angelegten Variablen ändern können.

Über die Instanziierung dieser Klassen wird die Spezifikation des in Listing 4.2 dargestellten Szenarios möglich. Dieses Szenario wird ausgeführt, wenn die `applicationECU` einen `lockRequest("active request")` an den `obc` sendet. Da die Ausführung des Szenarios durch das Event angestoßen wird, kann dieses als *Trigger-Event* bezeichnet werden. Das eigentliche Verhalten ist im Rumpf des Szenarios beschrieben. In Zeile 2 wird zunächst die Verriegelungsanforderung evaluiert, um anschließend den Motor innerhalb der Ladedose anzusteuern (Zeile 3). Das Szenario endet mit der Rückantwort, dass der Ladestecker verriegelt ist (Zeile 4).

```
1  class ApplicationECU{}
2
3  class OBC{
4   var requestStatus = "no request"
5   var lockingStatus = "unlocked"
6   fun lockRequest(requestStatus : String) = event(requestStatus){this.requestStatus =
        requestStatus}
7   fun evaluateRequest() = event(){}
8   fun setStatus(lockingStatus : String) = event(lockingStatus){this.lockingStatus =
        lockingStatus}
9  }
10
11 class ChargingSocket{
12  var direction = "close"
13  fun actuateMotor(direction : String) = event(direction){this.direction = direction}
14
15 }
```

Listing 4.1: Spezifikation eines Szenarios zur Verrigelung des Ladesteckers

```
1  scenario(applicationECU sends obc.lockRequest("active request")){
2   request(obc.evaluateRequest())
3   request(obc sends chargingSocket.actuateMotor("close"))
4   request(chargingSocket sends obc.setStatus("locked"))
5  }
```

Listing 4.2: Spezifikation eines Szenarios zur Verrigelung des Ladesteckers

4.2.5 Bewertung der Ansätze

Verwandte Arbeiten argumentieren, dass die Modellierung von Anforderungen eine frühe und automatisierte Anforderungsanalyse ermöglicht und so insbesondere in komplexen Entwicklungsprojekten die Anforderungsqualität erhöhen kann (u.a. [BGH+14; Foc18; Hol19]). Wie in Kapitel 3.2.4 beschrieben, besteht jedoch die Herausforderung, eine geeignete Kombination von niederschwelliger Spezifikation in natürlicher Sprache und formalen Modellen zu finden. Um eine formale aber dennoch intuitive Modellierung zu

unterstützen, zeigen die zuvor beschriebenen Techniken szenariobasierte Ansätze. Es wird deutlich, dass die grafische Modellierung von Szenarien zwar verständlich und leicht anwendbar ist, in komplexen Entwicklungsprojekten jedoch zu aufwendig und somit ungeeignet ist (vgl. [GHM+15]). Die textuelle und szenariobasierte Modellierung ist eine potentiell geeignete Technik, da hierüber einerseits eine intuitive Modellierung adressiert wird, andererseits aber auch praktische Bedarfe wie die Wartbarkeit der Modelle und die Integrierbarkeit in existierende Entwicklungsprozesse gefördert wird (vgl. [LK23]). Im Vergleich zur grafischen Modellierung (z.B. mittels MSDs) können textuelle Modelle (z.B. mittels SMLK) direkt in die jeweiligen Versionsverwaltungssysteme eingebunden werden. Dennoch sind die beschriebenen Techniken zur Anforderungsmodellierung alleinstehend nicht ausreichend. Es fehlen Hilfsmittel und Methoden, die bei der Modellierungstätigkeit unterstützen. Ein Nachweis, dass die textuelle Modellierung von Anforderungen in realen Entwicklungsprojekten einen Mehrwert liefert, ist in verwandten Arbeiten nicht zu erkennen. Insbesondere vor dem Hintergrund, dass die Modellierung von Anforderungen und der damit erwarteten Qualitätssteigerung keinen Selbstzweck erfüllt, sondern im Kontext des Entwicklungsprojekts zu bewerten ist, besteht ein Bedarf für eine weitere Evaluierung dieser Ansätze (vgl. [FV19; FMV+22]).

4.3 Ansätze für den Entwurf und die Spezifikation von Testfällen

Dieses Kapitel behandelt grundlegende Aspekte für den Entwurf und die Spezifikation von Testfällen. Der Fokus liegt dabei auf dem sogenanntem *Blackbox-Testverfahren*. Hierbei werden die Testfälle aus einer Spezifikation abgeleitet, wobei der innere Aufbau des Testobjekts nicht bekannt ist, bzw. für den Testentwurf nicht berücksichtigt wird [SLS11, S. 114]. Eine wesentliche Testkategorie innerhalb der Blackbox-Testverfahren sind Akzeptanztests. Die Durchführung von Akzeptanztests ist eine Testaktivität, um die Erwartungen der Stakeholder mit den umgesetzten Systemeigenschaften abzugleichen, wobei sich nach SPILLNER & LINZ die Akzeptanzprüfung auch auf Zwischenergebnisse des zu entwickelnden Systems beziehen kann [SLS11, S. 64]. Im Automobilbereich kann bspw. das Erreichen von Sicherheitszielen für ein Steuergerät innerhalb eines Unternehmens durch Akzeptanztests erfasst werden. In diesem Fall liegt die Verantwortung bei dem Unternehmen, das das Steuergerät entwickelt. Weitergehend ermöglichen Akzeptanztests auf Fahrzeugebene das Zusammenspiel der integrierten Steuergeräte, um die Einhaltung übergeordneter Sicherheitsziele zu überprüfen. Die Verantwortung liegt hier bei dem Fahrzeughersteller. Somit existieren Akzeptanztests auf unterschiedlichen Ebenen in unterschiedlichen Organisationen, die jedoch voneinander abhängig sein können.

4.3.1 Dynamischer Test funktionaler Systemeigenschaften

Wird das Testobjekt für die V&V eines Systems ausgeführt, wird die Testaktivität als dynamisches Testen bezeichnet. Hierbei werden Eingabedaten bereitgestellt, mit denen das Testobjekt zur Ausführung gebracht wird. Als Reaktion auf die Eingabedaten stellt

das Testobjekt Testausgabedaten bereit. Die Analyse der Testausgabedaten ermöglicht die Bewertung, ob das Testobjekt den Akzeptanzkriterien genügt. Abbildung 4-18 verdeutlicht diese Herangehensweise auf Basis der zu Beginn eingeführten schematischen Darstellung eines Systems. Das entwickelte System, bestehend aus seinen Elementen und deren Beziehungen untereinander, ist hier das Testobjekt. Wie in der Abbildung dargestellt, existiert ein Point of Control (PoC), über den sich das Systemverhalten über die Bereitstellung der Testeingabedaten beeinflussen lässt. Demgegenüber steht der Point of Observation (PoO), über den die entsprechenden Testausgabedaten bereitgestellt werden. Die Testeingabedaten

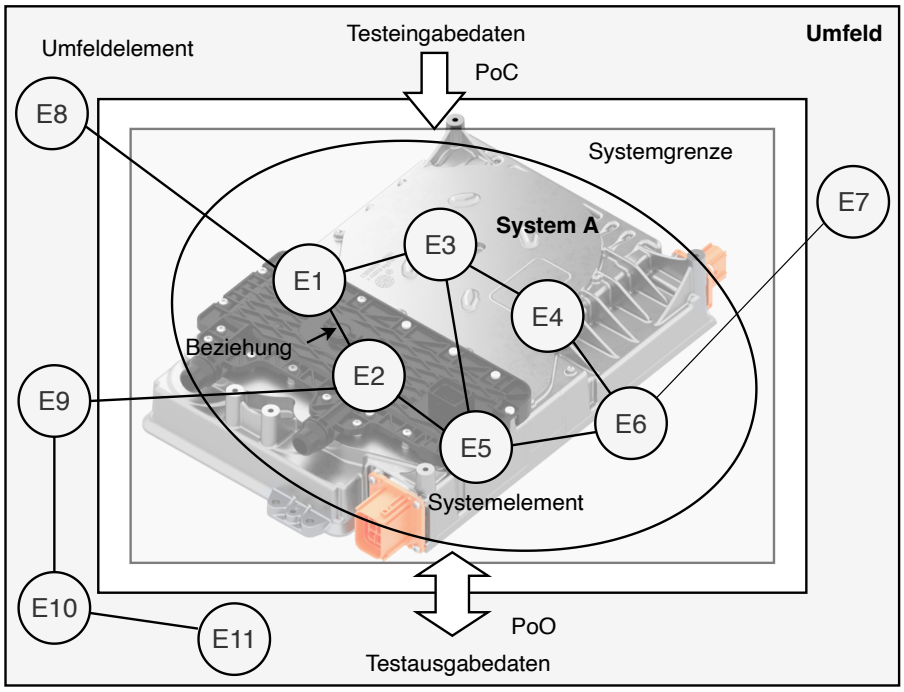


Bild 4-18: Blackbox-Testverfahren: Point of Control *(PoC) und* Point of Observation *(PoO) liegen außerhalb des Testobjekts. Das Testobjekt ist bspw. ein Steuergerät (nach [SLS11, S. 113]).*

leiten sich dabei aus einer Testbasis ab, aus der hervorgeht, wie sich das System in seiner Umwelt verhalten soll. Da hier jedoch eine große Menge an möglichen Testeingabedaten entsteht, die wiederum zu einer großen Menge an möglichen Testfällen führt, ist es notwendig, die Testeingabedaten und somit die Testfälle über geeignete Methoden zu ermitteln (vgl. [SLS11, S. 109 ff.]).

4.3.2 Ursache-Wirkungs-Graphen-Analyse

Ein Verfahren, das bei der Identifikation einer geeigneten Menge von Testfällen unterstützen kann, ist die Ursache-Wirkungs-Graphen-Analyse, die eine Beschreibung des Systemverhaltens über die direkte Beziehung von Testeingabewerten und zugehörigen Testausgabewerten ermöglicht [Elm73] [MSB11]. Im Gegensatz zu anderen Verfahren, wie bspw. der Äquivalenzklassenbildung (vgl. [SLS11, S. 114 ff.]), besteht hier ein direkter Zusammenhang zwischen den Ein- und Ausgabewerten, wodurch es erforderlich ist, dass

dieser Zusammenhang auch aus der Testbasis hervorgeht (vgl. [Fis22, S. 32]). Dabei ist das Ziel, eine möglichst geringe Anzahl an Testfällen zu erzeugen. Hierfür werden die Ursachen und Wirkungen über einen Graphen (Cause Effect Graph (CEG)) in Beziehung gesetzt, wobei jede Ursache als auch Wirkung über eine Bedingung beschrieben ist, die *wahr* oder *falsch* sein kann. Dabei können die Ursachen zusätzlich über logische Operatoren verknüpft werden [NP95].

Um auf Basis des CEG Testfälle abzuleiten, ist der CEG in eine Entscheidungstabelle zu überführen, wobei jede Spalte der Tabelle einem Testfall entspricht. Hierfür sind die folgenden Schritte auszuführen (vgl. [SLS11, S. 142]):

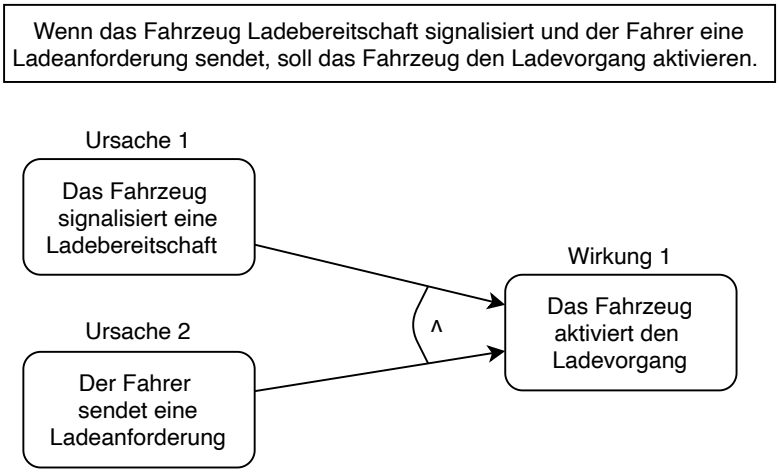
1) Von den im CEG dargestellten Wirkungen ist eine Wirkung auszuwählen.

2) Ausgehend von der ausgewählten Wirkung sind alle Ursachen zu identifizieren, die zu der ausgewählten Wirkung führen bzw. nicht zu der ausgewählten Wirkung führen.

3) Die identifizierten Ursache-Wirkungs Kombinationen sind anschließend in die Entscheidungstabelle zu überführen, wobei jede Kombination über eine Spalte beschrieben ist, die auch die Zustände der anderen Wirkungen beinhaltet.

4) Im letzten Schritt sind Spalten mit identischen Einträgen zu entfernen.

Dieses Vorgehen kann anhand des in Bild 4-19 dargestellten Beispiels verdeutlicht werden. In der dargestellten Anforderung sind zwei Ursachen und eine Wirkung beschrieben. Die



		Testfälle		
		Fall 1	Fall 2	Fall 3
Eingabewerte	Ursache 1	1	0	1
	Ursache 2	0	1	1
Ausgabewerte	Wirkung 1	0	0	1

Bild 4-19: Einfaches Beispiel für die Anwendung der Ursache-Wirkungs-Graph-Analyse.

im CEG dargestellte Beziehung beschreibt somit, dass das Fahrzeug den Ladevorgang

nur aktivieren soll, wenn beide Ursachen erfüllt sind. In diesem Fall muss das Fahrzeug eine Ladebereitschaft signalisieren und der Fahrer eine Ladeanforderung senden. Die Ausführung der vier beschriebenen Schritte führt dann in Kombination mit dem folgenden Regelsatz (vgl. [Fis22, S. 34]) zu der in Bild 4-19 dargestellten Entscheidungstabelle, aus der drei Testfälle hervorgehen.

1) Bei Disjunktionen mit dem Ergebnis 1 (die Wirkung tritt ein) werden nur Eingabekombinationen berücksichtigt, bei denen eine Eingabe den Wert 1 und alle anderen Eingaben den Wert 0 haben.

2) Bei Disjunktionen mit dem Ergebnis 0 (die Wirkung tritt nicht ein) werden alle Eingaben auf 0 gesetzt.

3) Bei Konjunktionen mit dem Ergebnis 0 (die Wirkung tritt nicht ein) werden nur Eingabekombinationen berücksichtigt, bei denen eine Eingabe den Wert 0 und alle anderen Eingaben den Wert 1 haben.

4) Bei Konjunktionen mit dem Ergebnis 1 werden alle Eingänge auf 1 gesetzt.

Das Vorgehen kombiniert mit dem Regelsatz führt zu einer maximalen Wahrscheinlichkeit für das Auffinden von Fehlern, wobei gleichzeitig eine reduzierte Menge an Testfällen definiert werden kann [Lig09]. In dem einfachen Beispiel führt die Anwendung des Verfahrens zu 3 von 4 möglichen Testfällen.

4.3.3 Testfallgenerierung aus natürlich sprachlichen Anforderungen

Die CEG-Analyse ist ein geeigneter Ansatz für die systematische Herleitung von Testfällen. Die Anwendung dieser Technik ist bei komplexen Anforderungsspezifikationen jedoch zeitaufwändig, wenn diese manuell erfolgt. Ein aktueller und für diese Arbeit relevanter Ansatz für die automatisierte Herleitung der Testfälle auf Basis von Anforderungen in unstrukturierter Sprache ist von FISCHBACH beschrieben [Fis22]. Dieser Ansatz basiert auf den Konzepten des modellbasierten Testens (Model-Based Testing (MBT)) (vgl. [Pre03]) und besteht aus einem dreistufigen Verfahren. Zunächst werden Ursache-Wirkungs-Zusammenhänge in Anforderungen über einen Natural Language Processing (NLP) Ansatz automatisiert identifiziert [FFV+23]. Diese werden anschließend in einen CEG übertragen, wobei dieser CEG den Ansätzen des MBT folgend als Modell verstanden wird, aus dem schließlich im letzten Schritt die Testfälle generiert werden [Fis22, S. 165 ff.]. Eine Übersicht des Verfahrens ist in Abbildung 4-20 veranschaulicht.

Die Anwendung des Ansatzes zeigt, dass im Vergleich zu existierenden Testfallspezifikationen ein Großteil (71.8%) der Testfälle automatisiert erstellt werden kann und darüber hinaus Testfälle generiert wurden, die beim manuellen Entwurf nicht berücksichtigt wurden [Fis22, S. 165] [FFV+23].

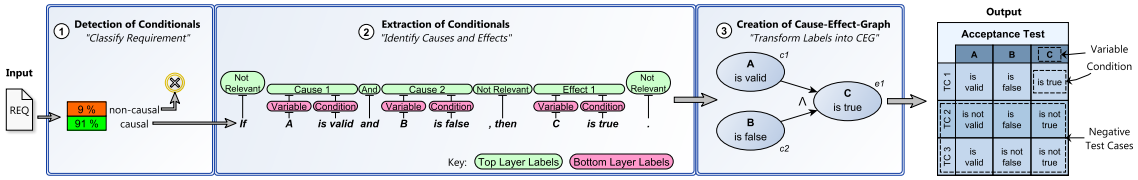


Bild 4-20: Ansatz für für die automatisierte Herleitung von Testfällen aus Anforderungen in unstrukturierter natürlicher Sprache.

4.3.4 Bewertung der Ansätze

Wie in der Problemanalyse beschrieben (Kapitel 3.3.4), besteht die Herausforderung in der Erstellung von präzisen, vollständigen und konsistenten Testspezifikationen, die dennoch leicht editierbar und wartbar sind. Ausgehend von CEGs zur systematische Herleitung einer geeigneten Menge an Testfällen, zeigt der Ansatz von FISCHBACH eine Technik, um aus textuellen Anforderungen Testfälle zu generieren. Die Fallstudien verdeutlichen die Anwendbarkeit innerhalb der Automobilindustrie [FFV+23]. Somit ist dieser Ansatz potentiell geeignet, um Test-Designer bei der Erstellung von Testspezifikationen zu unterstützen. Jedoch betrachtet der Ansatz nur einzelne Anforderungen. Um eine Validierung der Anforderung zu ermöglichen und um die Wiederverwendung von Lösungswissen zu fördern, ist zusätzlich die Vernetzung von Testfällen notwendig. Es können Beziehungen zwischen Testfällen bestehen, um bspw. ein Testszenario zu beschreiben (vgl. [Kan03]). Ebenso kann die Ausführung von Testfällen an bestimmte Ressourcen gebunden sein (vgl. [MBB+21]). Für die Vernetzung der generierten Testfälle ist somit weiterer Modellierungsaufwand nötig, der über geeignete methodische Ansätze zu unterstützen ist.

4.4 Ansätze für die integrative Modellierung von Anforderungen und Tests

BJARNASON et al. stellen fest, dass Herausforderungen im RE und in der V&V in der Forschung und Praxis überwiegend unabhängig voneinander betrachtet werden. Häufig wird die Traceability von Artefakten untersucht, um bspw. die Auswertung der Testabdeckung zu fördern oder Auswirkungsanalysen bei Anforderungsänderungen zu unterstützen. Nach Meinung der Autoren ist die alleinige Betrachtung der Traceability jedoch nicht ausreichend, um das RE und die V&V besser miteinander zu verzahnen. Vielmehr muss die Interaktion zwischen den beteiligten Rollen über die gesamte Entwicklungszeit einbezogen werden, von der Abstimmung der Entwicklungsziele und Teststrategien bis hin zu detaillierten Anforderungen und Testfällen [BRB+13] (vgl. [DHK+17]) (s. Kapitel 3.5). Bild 4-21 verdeutlicht diese Forderung anhand der vielschichtigen Beziehungen zwischen Artefakten auf unterschiedlichen Abstraktionsstufen des RE und Artefakten der V&V auf Basis des in der Automobilindustrie etablierten V-Modells.

Einige der im Kontext dieser Arbeit relevanten Ansätze, die eine engere Verzahnung des RE und der V&V adressieren, sind im folgenden aufgeführt.

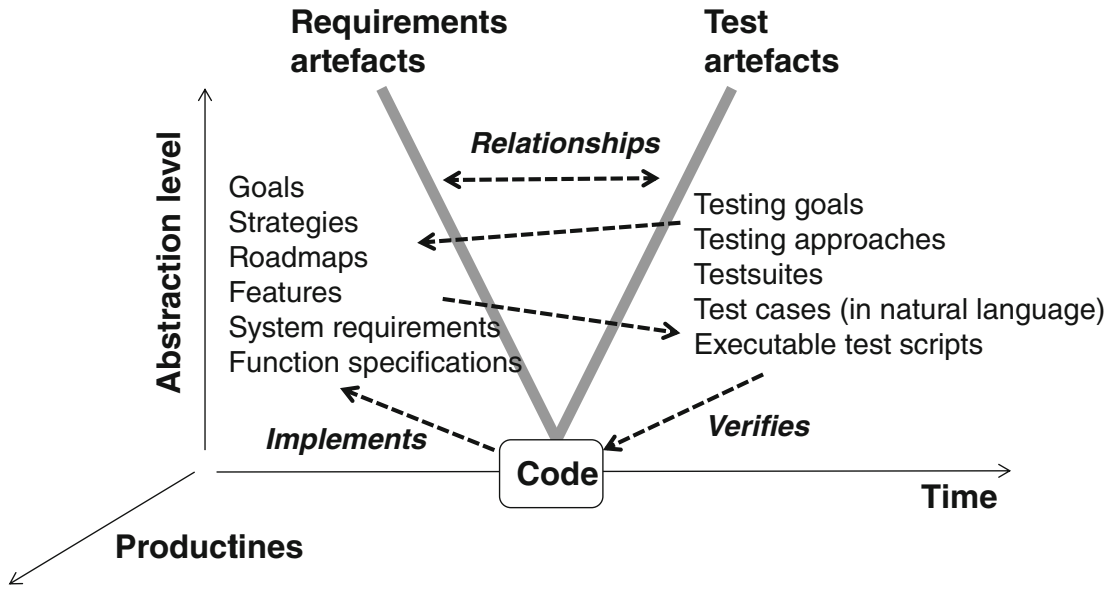


Bild 4-21: Konzeptuelles Modell zur Beschreibung Beziehungen von Artefakten des RE und der V&V [BRB+13].

4.4.1 Test-Driven Modeling (TDM)

Eine ursprünglich in der agilen Softwareentwicklung eingeführte Technik ist die testgetriebene Entwicklung (Test-Driven Development (TDD)). TDD zielt darauf ab, das Testen als reaktive Arbeit nach der Implementierung der Software zu proaktiver Arbeit als Ausgangspunkt für die Implementierung zu machen, um die Entwicklung von zuverlässiger Software in kurzen Iterationen zu unterstützen. Auf diese Weise soll vermieden werden, dass Fehler erst zu einem späten Zeitpunkt in der Entwicklung identifiziert werden, die dann zu langwierigen und vermeidbaren Entwicklungsiterationen führen [Bec03]. Der Ansatz folgt dem *red/green/refactor*-Mantra, bei dem die Implementierung einer neuen Funktionalität mit der Spezifikation eines Testfalls beginnt. Bei der Ausführung dieses Testfalls wird zunächst erwartet, dass dieser Fehler anzeigt, da die geforderte Funktionalität noch nicht implementiert wurde (*red*). Im zweiten Schritt wird die Implementierung der Software so umgesetzt, dass der Testfall bei einer erneuten Ausführung erfolgreich ist (*green*). Der dritte Schritt umfasst die Anpassung der Implementierung, wie bspw. das Entfernen von doppelten oder nicht notwendigem Programmcode, der durch die ausschließliche Fokussierung auf die erfolgreiche Durchführung des Testfalls entstanden ist (*refactor*) [Bec03].

Basierend auf positiven Ergebnissen durch die Anwendung von TDD, bspw. bezogen auf die Fehlerrate, Produktivität oder Häufigkeit der Testdurchführung [MW03], sind einige Ansätze entstanden, die TDD nicht auf die Implementierung von Programmcode beschränken, sondern auch für Modellierungsaktivitäten verwenden. ZUGAL et al. zeigen einen Ansatz für die deklarative Modellierung von Geschäftsprozessen, den sie als testgetriebene Modellierung (Test-Driven Modeling (TDM)) bezeichnen [ZPW12]. Vergleichbar mit Ansätzen für die Anforderungsmodellierung im RE (s. Kapitel 4.2), sehen die Autoren die Herausforderung, dass die deklarative Modellierung in der Forschung zwar als ein

geeignetes Mittel für die flexible Modellierung von Geschäftsprozessen gesehen wird, in der Praxis jedoch wenig Beachtung findet. Die Autoren führen dies auf eine schwere Verständlichkeit und Wartbarkeit der Modelle zurück und adressieren diese Probleme mit der Kombination von TDD und der Modellierung der Geschäftsprozesse [ZPW12; ZHP+13].

Wie ZUGAL et al. in Bild 4-22 darstellen, entsteht durch die spezifizierten Testfälle als Teil des TDM Ansatzes ein zusätzlicher Kommunikationskanal (s. 4, 5 und 6 in Bild 4-22), der die Zusammenarbeit zwischen dem Domänenexperten (DE) und dem Modellersteller (ME) fördert. Üblicherweise stellt der DE Informationen über seine Domäne bereit (2), die dann von dem ME in ein Modell überführt werden (3). Über diesen Kommunikationspfad (2 und 3) kann der ME Resultate der Modellierung an den DE zurückgeben. Dem TDM Ansatz

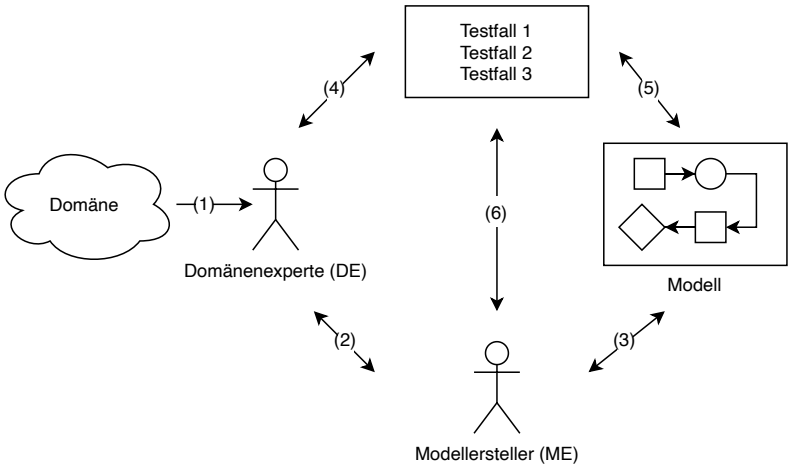


Bild 4-22: Kommunikationsmodell für die testgetriebene Modellierung [ZHP+13].

folgend können die Testfälle und Testergebnisse jedoch sowohl durch den DE als auch den ME interpretiert werden (4 und 6), wodurch der DE einen Zugang zu dem vom ME erstellten Modell erhält (5). Auf diese Weise entsteht über die Testfälle als Ausgangspunkt für die Modellierungsaktivitäten eine Kommunikationsgrundlage zwischen DE und ME [ZHP+13].

Die von ZUGAL et al. durchgeführte Studie zeigt, dass dieser zusätzliche Kommunikationskanal tatsächlich verwendet wird und die Kommunikation zwischen ME und DE mit Hilfe der Testfälle präziser macht [ZHP+13]. Neben einer verbesserten Kommunikation verdeutlicht die Studie zusätzlich, dass TDM im Kontext der Geschäftsprozessmodellierung die kognitive Belastung bei der Modellierungsaktivität verringert und die Qualität positiv beeinflusst [ZHP+13].

ZHANG beschreibt einen weiteren TDM Ansatz. Hier werden die oben beschriebenen MSCs (s. Kapitel 4.2) für die Modellierung verwendet [Zha04]. Ausgehend von Systemanforderungen ist ein durchgängiger Ansatz von der Modellierung der Systemanforderungen bis hin zu der Generierung von Programmcode beschrieben, wobei zwischen einem *High-Level Modeling* (HLM) und einem *Low-Level Modeling* (LLM) unterschieden wird. Das

HLM wird von ZHANG dazu verwendet, Anwendungsszenarien auf Basis der Systemanforderungen über MSCs zu modellieren. Zusätzlich umfasst das HLM die Modellierung des Systemverhaltens mittels MSCs, um das in den Anwendungsszenarien geforderte Verhalten umzusetzen. Somit können die über MSCs modellierten Anwendungsszenarien als Testfälle verstanden werden, die die Modellierung des Systemverhaltens treiben. ZHANG verfolgt dabei einen iterativen Top-Down Ansatz und spezifiziert zunächst im Rahmen des HLM die notwendigen Subsysteme und Signale, die zwischen diesen Subsystemen ausgetauscht werden müssen, um das geforderte Systemverhalten zu realisieren. Die konkreten Daten, die über diese Signale ausgetauscht werden, werden als Teil der LLM ergänzt. Eine industrielle Anwendung dieses TDM Ansatzes wird von ZHANG als positiv bewertet, da sich durch die automatisierte Generierung des Programmcodes die Produktivität erhöht und sich die Anzahl an Defekten in der Software im Vergleich zu dem etablierten Vorgehen verringert hat [Zha04].

Einen TDM Ansatz im Bereich der Anforderungsmodellierung zeigen GLINZ et al. [SM07]. Dieser Ansatz basiert ebenfalls auf MSCs und beschreibt ein zweistufiges iteratives Vorgehen. Zunächst wird auch hier das erwartete Systemverhalten über Anwendungsszenarien erfasst. Vergleichbar mit dem *Play-Out* Ansatz [HM03b] kann über eine Werkzeugunterstützung das Systemverhalten exemplarisch simuliert und schrittweise durch den Anwender konkretisiert werden. Das Ergebnis der Simulation wird über MSCs dokumentiert. Nachgelagert wird das exemplarische Systemverhalten manuell mittels Statecharts [Har87] generalisiert, wobei das resultierende Modell erneut ausgeführt werden kann. Dadurch ist der Nachweis möglich, ob das modellierte Verhalten mit den zuvor dokumentierten MSCs übereinstimmt. Da die schrittweise Konkretisierung des Systemverhaltens und die Modellierung mittels Statecharts durch die Anwendungsszenarien getrieben wird, ist dieser Ansatz vergleichbar mit den zuvor genannten TDM Ansätzen.

4.4.2 Behavior-Driven Development (BDD)

NORTH erweitert den TDD Ansatz, um ausgehend von den oben genannten Potentialen der testgetriebenen Entwicklung (vgl. [Bec03; ZPW12; ZHP+13]), die disziplinübergreifende Zusammenarbeit weiter zu fördern. Anstatt von einzelnen Testfällen setzt er die Beschreibung eines erwarteten Verhaltens in den Vordergrund (Behavior-Driven Development (BDD)). NORTH beschreibt, dass die strukturierte und allgemein verständliche Beschreibung von Akzeptanzkriterien hierfür geeignet ist [Nor06]. Eine Umsetzung dieses Ansatzes ist u.a. über die *Gherkin* Syntax beschrieben[4], die über das Werkzeug *Cucumber*[5] angewendet werden kann.

Die Beschreibung der Akzeptanzkriterien mittels *Gherkin* erfolgt innerhalb eines *Feature Files*, in dem jede Zeile mit einem Schlüsselwort beginnt. Ein Beispiel ist in Listing

[4] https://cucumber.io/docs/gherkin/reference/, zuletzt abgerufen am 19.1.2023

[5] https://cucumber.io, zuletzt abgerufen am 19.1.2023

4.3 beschrieben. Hier beginnt die Verhaltensbeschreibung mit der Umschreibung eines *Features* (Zeile 1). Das Feature kann dabei aus mehreren Szenarien bestehen. Ein Szenario ist exemplarisch in Zeile 2 aufgeführt. Die eigentliche Verhaltensbeschreibung folgt der *Given-When-Then* Struktur (Zeile 3-5).

```
1 Feature: Activate charging process
2  Scenario: The customer sends a charging request
3   Given the vehicle signals that it is ready to charge
4   When the customer sends a charging request
5   Then the vehicle activates the charging process
6   ...
```

Listing 4.3: Spezifikation eines Szenarios in der Gherkin Syntax als Teil eines Feature Files.

Über geeignete Werkzeuge wie bspw. *Cucumber* erlaubt diese strukturierte Beschreibung die Generierung von Testschritten. Wie in Listing 4.4 dargestellt, dienen diese Testschritte als Ausgangspunkt für die durch das erwartete Verhalten getriebene Entwicklung.

```
1 Given("the vehicle signals that it is ready to charge") {
2  // manually added code to set the precondition
3 }
4 When("the customer sends a charging request") {
5    // manually added code to trigger the system
6 }
7 Then("the vehicle activates the charging process") {
8  // manually added code to evaluate the response
9 }
```

Listing 4.4: Generierte Testschritte

Auf diese Weise entsteht eine enge Verzahnung zwischen der allgemeinverständlichen Verhaltensbeschreibung und des zu entwickelnden Systems, da die Testausführung direkt mit der Beschreibung der einzelnen *Feature* in der *Gherkin* Syntax verlinkt ist. Die Ergebnisse der Testausführung können somit direkt zur disziplinübergreifenden Konsolidierung des erwarteten Systemverhaltens und des aktuellen Umsetzungsstandes genutzt werden (vgl. [Nor06]).

4.4.3 Bewertung der Ansätze

Die beschriebenen TDM und BDD Ansätze zeigen, dass das testgetriebene Vorgehen die Kommunikation zwischen den beteiligten Rollen sowie die Qualität der resultierenden Modelle positiv beeinflussen kann. Die Erweiterung durch NORTH [Nor06], der nicht einzelne Testfälle, sondern eine strukturierte Verhaltensbeschreibung voranstellt, zeigt exemplarisch, wie Anforderungen, Testfälle und das zu entwickelnde Artefakt miteinander verzahnt werden können. Hierdurch wird die von BJARNOSON et al. [BRB+13] identifizierten Notwendigkeit für eine bessere Interaktion der Rollen innerhalb des RE und der V&V gefördert. Der TDM Ansatz ist somit potentiell geeignet, um Systemarchitekten und Test-Designer bei den Modellierungstätigkeiten zu unterstützten. Verwandte Arbeiten, die bereits die Anwendung dieser Techniken in der Anforderungsanalysephase untersuchen, sind nicht bekannt.

4.5 Ansätze für die Wiederverwendung von Lösungswissen

Wie bereits in Kapitel 3.4 angedeutet sind Lösungsmuster ein geeigneter Ansatz für die Externalisierung und Wiederverwendung von Lösungswissen (vgl. [Dum10, S. 128 ff.] [Ana15, S. 32 f.] [ADK+20]). Für die Erarbeitung von Lösungsmustern im Bereich der Anforderungs- und Testspezifikation sind in Kapitel 4.5.1 zunächst grundlegende musterbasierte Ansätze beschrieben. Darauf aufbauend sind relevante Ansätze für die Anforderungsspezifikation (Kapitel 4.5.2) und den Testentwurf (Kapitel 4.5.3) aufgeführt. Abschließend sind in Kapitel 4.5.4 methodische Ansätze zur Sicherung von Lösungswissen beschrieben.

4.5.1 Grundlegende Konzepte

Die Arbeiten von ALEXANDER [Ale64; AIS77; Ale79] werden in der Literatur häufig als Ausgangspunkt für den musterbasierten Entwurf von technischen Systemen genannt. Zur Beschreibung eines Musters definiert ALEXANDER vier Kategorien:

1) Der **Name** des Musters.

2) Einen **Kontext** in dem das Muster eingesetzt werden kann, einschließlich der Beziehungen zu weiteren Mustern.

3) Das adressierte **Problem**.

4) Die **Lösung** für das adressierte Problem in Form einer Anleitung für den Anwender des Lösungsmusters.

Auf Basis dieser Kategorien und dem Prinzip der systematischen Dokumentation von Problem-Lösungspaaren folgend, erweitert CLOUTIER die Kategorien für eine Beschreibung von Lösungsmustern für das SE [Clo06, S. 42]. Eine Übersicht dieser Beschreibung ist in Tabelle 4-1 dargestellt.

Ausgehend von diesen Arbeiten beschreibt ANACKER eine einheitliche Strukturierung von Lösungsmustern. Bild 4-23 zeigt eine vereinfachte Darstellung. ANACKER gibt an, dass der **Name** eines Lösungsmusters eindeutig zu benennen ist, wobei der Name so zu wählen ist, dass dieser bei der Suche nach geeigneten Lösungsmustern unterstützen kann. Die Beschreibung des **Problems** ist neben dem Musternamen die wesentliche Information für die Auswahl eines Musters und umfasst die Merkmale bzw. Funktionen des zu realisierenden Systems. Zur Ausgestaltung einer **Lösung** schlägt ANACKER die Beschreibung der Wirkstruktur und des Verhaltens vor, was bspw. über geeignete Partialmodelle der Spezifikationstechnik CONSENS [GFD+09] möglich ist. Über die Beschreibung des **Kontextes** sollen erfolgreiche Anwendungen des Musters beschrieben werden, die ggf. durch textuelle Beschreibungen und Skizzen zu ergänzen sind [Ana15, S. 106 f.].

ANACKER argumentiert weitergehend, dass die einheitlich strukturierten Lösungsmuster innerhalb des SE in unterschiedlichen Bereichen eine Relevanz haben und sich die An-

Name der Kategorie	Beschreibung
Mustername	Der Name des Musters sollte beschreibend sein, damit der Nutzer des Musters die Verwendung verstehen kann.
Aliasnamen	Andere Namen, unter denen das Muster bekannt sein kann.
Schlüsselwörter	Schlüsselwörter, die bei der Suche nach geeigneten Mustern in einem Repository helfen.
Problemkontext	Kurze Erörterung der Arten von Situationen, in denen das Problem auftreten kann. Sie sollte so weit gefasst sein, dass eine beliebige Anzahl von Situationen, in denen das Problem auftreten kann, berücksichtigt werden kann.
Problembeschreibung	Was ist das Problem, das mit diesem Muster gelöst werden kann?
Kräfte	Herausforderungen, die bei dem vom Muster behandelten Problem bestehen, und die Probleme bei der Anwendung des Musters. Einschränkungen die das Muster bei seiner Anwendung mit sich bringen kann. Beschreibung aus unterschiedlichen Blickwinkeln auf das Muster.
Lösung	Diskussion darüber, wie das Muster das jeweilige Problem löst.
Diagramme	Dies können ein oder mehrere Diagramme sein, die zur Darstellung des Musters erforderlich sind. Dies kann jede gewünschte Notationsmethode sein.
Schnittstellen	Diskussion der kritischen Schnittstellen oder Informationsflüsse, die für die Implementierung des Musters erforderlich sind - welche Parameter der Schnittstelle können geändert werden und welche nicht. Welche Schnittstellenabhängigkeiten gibt es, wenn überhaupt?
Resultierender Kontext	Welche ungelösten Probleme gibt es bei der Anwendung/-Verwendung des Musters?
Beispiel	Ein Beispiel, wie das Muster angewendet werden kann.
Begründung für das Muster	Warum das Muster funktioniert.
Bekannte Anwendungen	Wo wird das Muster noch an anderen Stellen oder in anderen Anwendungen verwendet?
Verwandte Muster	Andere Muster, die in Verbindung oder in Assoziation mit diesem Muster verwendet werden können.
Referenzen	Weitere Informationen, die für das Verständnis oder die Anwendung des Musters nützlich sein können.
Autoren	Wer hat das Muster dokumentiert?

Tabelle 4-1: SE Muster nach CLOUTIER *[Clo06, S. 42]*

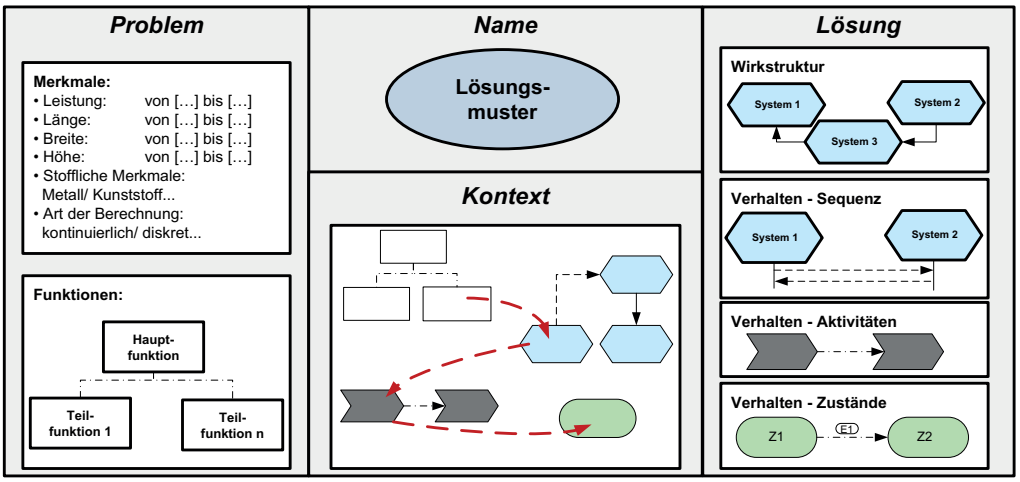


Bild 4-23: Einheitliche Strukturierung von Lösungsmustern für den Systementwurf nach ANACKER *[Ana15, S. 106]*

wendung somit nicht auf einzelne Bereiche beschränkt [Ana15, S. 49 f.]. Demzufolge veranschaulicht die in Bild 4-24 dargestellte Musterhierachie nach CLOUTIER [CV06] eine Übersicht von Anwendungsbereichen für Lösungsmuster innerhalb des SE. Die Übersicht adressiert auf der obersten Ebene nicht technische Bereiche (Organisationsmuster, Geschäftsmuster, Leitbildmuster). Darunter sind Systemarchitekturmuster aufgeführt, die weiter in Struktur-, Rollen-, Anforderungs-, Aktivitäts-, und Prozessmuster zergliedert sind. Dabei verdeutlicht die Darstellung die direkte Beziehung der Anforderungsmuster mit den Systemtestmustern, die beide im Kontext dieser Arbeit relevant sind.

4.5.2 Muster im Bereich Anforderungsspezifikation

Für die Entwicklung von Anforderungsmustern existieren unterschiedliche Techniken. Für diese Arbeit relevante Techniken sind im folgenden beschrieben.

UML basierte Anforderungsmuster nach KONRAD & CHENG

Eine bereits in Kapitel 3.4.2 beschriebene Technik sind die von KONRAD & CHENG ausgearbeiteten Anforderungsmuster [CA09]. In weiteren Arbeiten der Autoren sind u.a. Muster für die Spezifikation von Echtzeitanforderungen beschrieben [KC05]. Zudem ist ein Ansatz für die mustergetriebene Modellierung innerhalb der Anforderungsanalysephase beschrieben [KCC05]. Dieser Ansatz zeigt u.a. die Integration der mustergetriebenen Modellierung in bestehende Entwicklungsprozesse, was nach FRANCH [FPQ20] wichtig für die erfolgreiche Wiederverwendung von Lösungswissen ist.

MSD basierte Anforderungsmuster nach FOCKEL ET AL.

Motiviert durch KONRAD & CHENG zeigen FOCKEL et al. einen weitergehenden Ansatz, der auf den in Kapitel 4.2.2 eingeführten MSDs basiert. Die Autoren argumentieren, dass es zunehmend schwieriger wird, qualitativ hochwertige Anforderungsspezifikationen zu erstellen, wenn komplexe nachrichtenbasierte Interaktionen in verteilten softwareintensiven

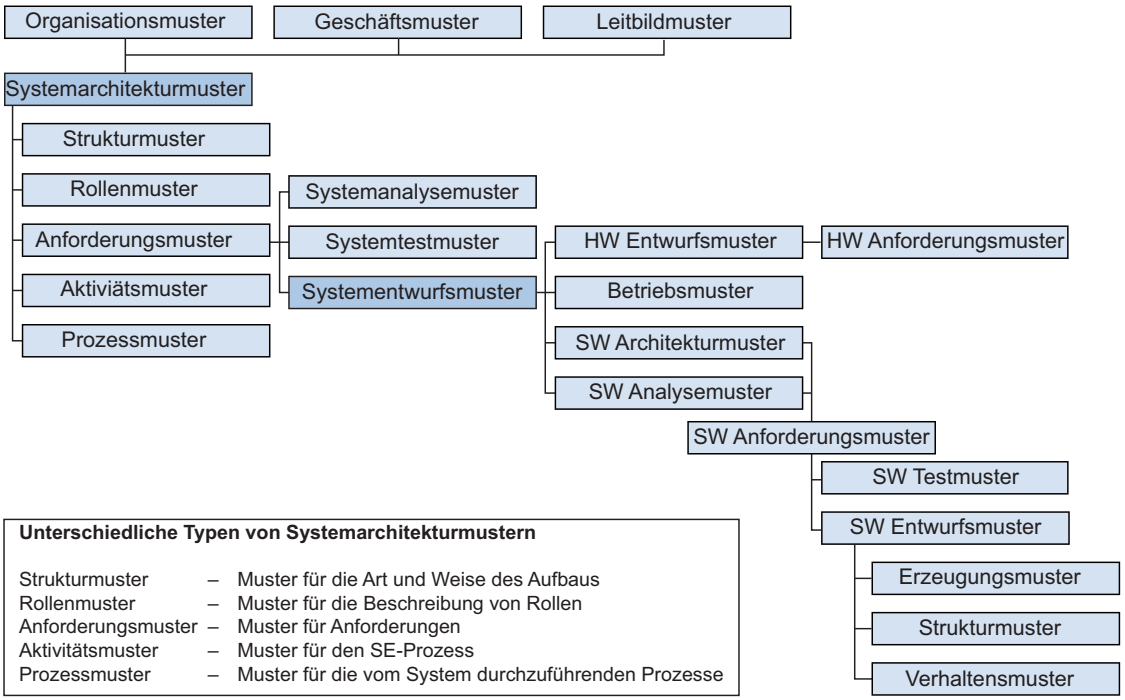


Bild 4-24: Musterhierarchie nach CLOUTIER *[Ana15, S. 50] [CV06]*

Systemen zu berücksichtigen sind. Um Entwicklungsingenieure hierbei zu unterstützen, können automatisierte Analysetechniken eingesetzt werden. Da für die notwendige formale Spezifikation jedoch zusätzliches Expertenwissen notwendig ist, schlagen FOCKEL et al. einen Musterkatalog vor, der den Entwicklungsingenieuren die Anwendung der formalen Modellierung erleichtern soll [FHK+16]. Der Fokus liegt hierbei auf Echtzeit- und Sicherheitsanforderungen, die innerhalb der Automobilindustrie von großer Bedeutung sind. Insgesamt sind 86 Muster beschrieben, wobei diese aus verwandten Arbeiten abgeleitet wurden (s. KONRAD & CHENG [KC05], DWYER [DAC99] und BITSCH [Bit01])

Der Aufbau des Musterkatalogs ist über ein Beispiel aus der Automobilindustrie anhand von drei Anforderungen veranschaulicht [FHK+16]. Diese Anforderungen beschreiben die Interaktion zwischen Steuergeräten zur sicheren Verrieglung der Fahrzeugtüren. Dabei stellt die *Electronic Stability Control* (ESC) Informationen an das *RearDoorSystem* bereit, das wiederum mit der *RearDoorMechanics* interagiert. Die Anforderungen sind wie folgt definiert [FHK+16]:

Anforderung 1: „*Once the ESC informs the RearDoorSystem that the vehicle starts moving, the RearDoorSystem shall eventually lock the RearDoorMechanics before the ESC informs the RearDoorSystem that the vehicle stopped moving.*“

Anforderung 2: „*Once the ESC informs the Rear- DoorSystem that the vehicle starts moving, the RearDoorSystem shall lock the RearDoorMechanics after at most 3s.* “

Anforderung 3: „*Once the ESC informs the RearDoorSystem that the vehicle starts moving, the RearDoorSystem may not unlock the RearDoorMechanics before the ESC informs the*

RearDoorSystem that the vehicle stopped moving.“

Der Beschreibung in Kapitel 4.2.2 folgend, können diese Anforderungen über das in Bild 4-25 dargestellte MSD formalisiert werden, wobei hier zusätzlich der zeitliche Verlauf berücksichtigt wird.

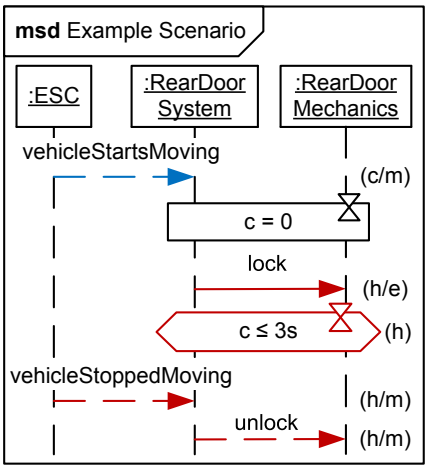


Bild 4-25: Formalisierte Darstellung der Anforderungen innerhalb eines MSDs. [FHK+16]

Ein exemplarisches Muster ist das *Absence Pattern*, das bei der Modellierung einzelner Anforderungen unterstützen kann. Das Muster wird durch das MSD in Bild 4-26 spezifiziert. Hier ist festgelegt, dass das Event *p* nicht vor dem Event *r* auftreten darf, wenn das Event *q* ausgeführt wurde. Über die *false*-Bedingung wird somit eine bestimmte Reihenfolge für das Auftreten von Events untersagt. Wird dieses Muster auf die zuvor beschriebene

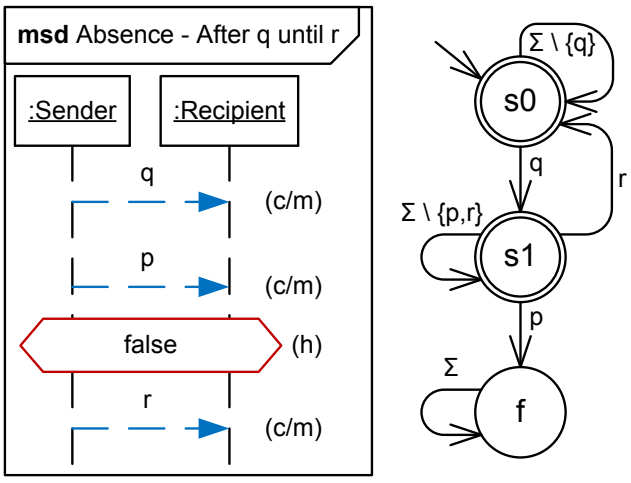


Bild 4-26: Formalisierte Darstellung der Anforderungen innerhalb eines MSDs. [FHK+16]

Anforderung 3 angewendet, führt dies zu der Darstellung in Abbildung 4-27. Auf diese Weise ist festgelegt, dass das Event *unlock* nicht ausgeführt werden darf, wenn nicht

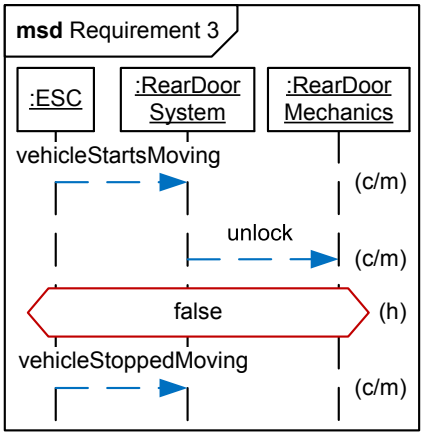


Bild 4-27: Formalisierte Darstellung der Anforderungen innerhalb eines MSDs. [FHK+16]

vorher dass Event *vehicleStoppedMoving* ausgeführt wurde. Da in diesem Beispiel die Nachrichten als *cold* definiert sind, wird keine bestimmte Reihenfolge des Auftretens verlangt, es ist lediglich eine bestimmte Reihenfolge durch die als *hot* definierte Bedingung verboten (vgl. Kapitel 4.2.2) [FHK+16].

4.5.3 Muster im Bereich Verifikation und Validierung

Wie oben beschrieben, sind für diese Arbeit innerhalb des SE Musteransätze relevant, die bei der Anforderungsspezifikation und beim Entwurf und der Spezifikation von Tests angewendet werden können. Geeignete Ansätze im Kontext der V&V sind im folgenden aufgeführt.

Verifikationsmuster nach SALADO & KANNAN

SALADO & KANNAN beschreiben elementare Muster für die Entwicklung von Verifikationsstrategien. Verifikationsstrategien sind als eine Zusammensetzung von Verifikationsaktivitäten v beschrieben, deren Beziehungen über gerichtete azyklische Graphen definiert sind [SK18]. Verifikationsaktivitäten werden hier verstanden als Aktivitäten zur Sammlung von Informationen über Eigenschaften des zu entwickelnden Systems. Diese Eigenschaften werden als Systemparameter θ zusammengefasst. Der Entwurf von Verifikationsstrategien ist motiviert durch die Unsicherheit, die mit dem entsprechendem Systemparameter θ verbunden ist. Der Ansatz verfolgt das Ziel, über eine formale Modellierung die Beschreibung von elementaren Verifikationsstrategien herzuleiten. Als Ergebnis wurden 7 elementare Muster herausgearbeitet, die als Bausteine für die Beschreibung komplexer Verifikationsstrategien betrachtet werden können [SK18; SK19].

Die 7 identifizierten Verifikationsstrategien (Pattern) sind in Bild 4-28 veranschaulicht und im folgenden beschrieben.

Pattern 1: Dieses Muster beschreibt den elementarsten Fall. Über die Verifikationsaktivität

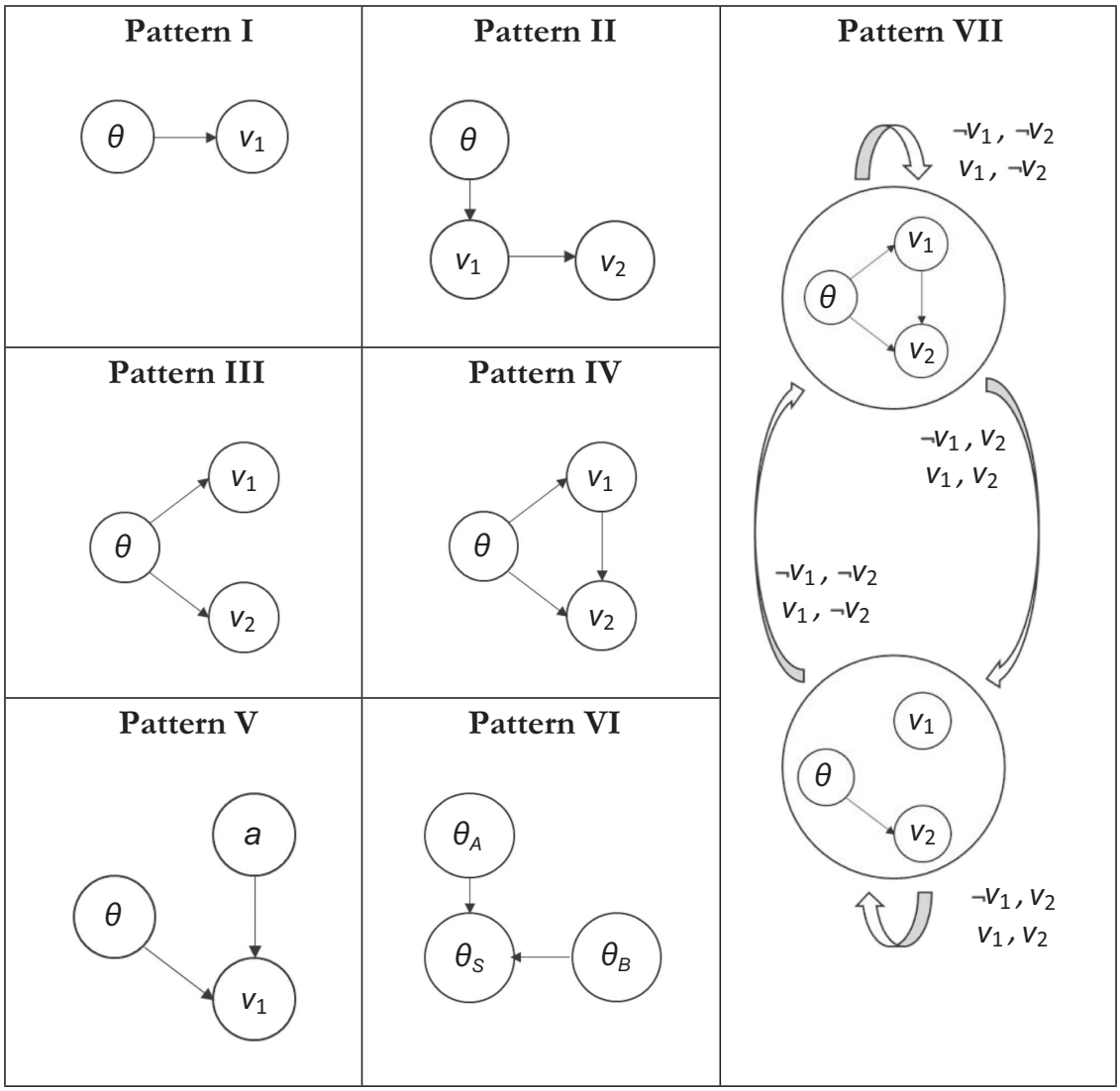


Bild 4-28: Elementare Muster zur Beschreibung von Verifikationsstrategien nach Salado & Kannan [SK19]

v_1 sollen Informationen über die Systemeigenschaft θ erfasst werden. Bei der Entwicklung einer neuen Funktion kann bspw. die Antwortzeit des Systems auf eine Benutzereingabe als Systemeigenschaft θ beschrieben werden, die über die Verifikationsaktivität v_1 ermittelt werden kann.

Pattern 2: Dieses Muster beschreibt, dass eine Verifikationsaktivität v_2 auf Basis einer vorangegangenen Verifikationsaktivität v_1 ausgeführt werden kann, wobei v_2 keine neuen Informationen über die Systemeigenschaft θ liefert. Die Autoren geben an, dass dieses Muster der Vollständigkeit halber aufgeführt wird, es in optimalen Verifikationsstrategien jedoch nicht enthalten sein sollte, da v_2 keine zusätzlichen Informationen über θ bereitstellt, sobald v_1 ausgeführt wurde. Ein mögliches Beispiel für die Anwendung dieses Musters ist, dass mit v_1 die Ausführungszeit einer Funktion bestimmt wird und dass auf Basis dieser Messung mit v_2 die Messgenauigkeit des Validierungssystems ermittelt wird.

Pattern 3: Dieses Muster beschreibt zwei voneinander unabhängige Verifikationsaktivitäten v_1 und v_2, die zur Ermittlung von Informationen über die gleiche Systemeigenschaft θ verwendet werden.

Pattern 4: Dieses Muster entspricht dem vorherigen Muster, hier sind die Verifikationsaktivitäten jedoch voneinander abhängig. Das bedeutet, dass das Ergebnis von v_1 die Verifikationsaktivität v_2 beeinflusst. Wird bspw. die Kühlmitteltemperatur eines Systems unter elektronischer Belastung an zwei unterschiedlichen Arbeitspunkten gemessen, lässt sich das Ergebnis der ersten Messung v_1 zu dem erwarteten Ergebnis der zweiten Messung v_2 extrapolieren.

Pattern 5: Dieses Muster entspricht dem ersten Muster, mit dem Unterschied, dass die Verifikationsaktivität v_1 von einem externen Parameter a beeinflusst wird. Wenn bspw. die Antwortzeit des Systems ermittelt wird, kann über a vorgegeben werden, ob der Messwert vom Validierungssystem als Sekunden oder Millisekunden interpretiert werden soll.

Pattern 6: Dieses Muster beschreibt Abhängigkeiten zwischen Systemeigenschaften. Wird bspw. die Ausführungszeit von zwei Subsystemen mit θ_A und θ_B beschrieben, beeinflussen diese Eigenschaften die mögliche Ausführungszeit des Gesamtsystems θ_S.

Pattern 7: Dieses Muster veranschaulicht die dynamische Zusammensetzung von Mustern basierend auf den verfügbaren Verifikationsnachweisen. Wenn bspw. die Ausführungszeit eines Systems die zu bestimmende Systemeigenschaft θ ist, ist dies durch eine Simulation möglich (Verifikationsaktivität v_1), die nachgelagert auf dem realen System gemessen werden kann (Verifikationsaktivität v_2). Das Ergebnis von v_1 beeinflusst somit das Vertrauen in das Messergebnis von v_2. Liegt jedoch das Messergebnis vor, ist die Bestimmung der Ausführungszeit über die Simulation nicht mehr notwendig.

Testmuster nach MCGREGOR

MCGREGOR kombiniert *Design Pattern* aus der Softwaretechnik [GHJ+94] mit Testmustern [McG99]. Aus der Sicht des Autors zielt die Anwendung von Mustern in beiden Bereichen darauf ab, den Entwurf von Software zu unterstützen, wobei in dem einen Fall der Entwurf des Softwaresystems adressiert wird und im anderen Fall der Entwurf der Testsoftware, um das Softwaresystem zu testen. MCGREGOR argumentiert, dass durch die integrative Betrachtung die Wiederverwendung von Lösungswissen in beiden Bereichen gefördert wird.

> *„(...) I can save testing time and effort by recognizing that the test of a segment of code that was designed using a design pattern actually follows a pattern itself. That test pattern can be reused wherever similar designs are used; however, each instance of use will be tailored to the exact needs of the current situation. [McG99].“*

Eine Gegenüberstellung für die Anwendung von *Design Pattern* und Testmuster ist nach MCGREGOR in Tabelle 4-2 dargestellt. Der Autor verdeutlicht die Anwendung von Test-

Abschnitt	**GOF Design Pattern Anwendung**	**Testmuster Anwendung**
Absicht	Kurze Beschreibung der Lösung	Beschreibung der Aspekte des zu testenden Entwurfs
Motivation	Eine Beschreibung des zu lösenden Problems und der Art und Weise, wie die Lösung das Problem adressiert	Eine Beschreibung des Entwurfsmusters und warum dieses Testmuster ein guter Weg ist, um es zu testen
Anwendbarkeit	Beschreibung, wann das Muster angewendet werden kann	Wird verwendet, wenn das Testmuster nicht immer für alle Instanzen des Entwurfsmusters verwendet wird
Struktur	Ein UML-Klassendiagramm, das die grundlegenden Beziehungen zwischen Klassen darstellt	Spezifikation der Testklassen (einschließlich der spezifischen Testfälle) und ihrer Interaktionen mit dem Cluster under Test (CUT)
Auswirkungen	Was ist an dem fertigen Gesamtentwurf anders, weil dieses Muster verwendet wurde?	Beschreibt, wie gut das CUT durch die Verwendung dieses Musters abgedeckt wird
Implementierung	Beschreibung der grundlegenden Durchführungsentscheidungen, die getroffen werden müssen	Identisch - umfasst den Zugang zu den Attributen und Verhaltensweisen in der Produktionssoftware
Kollaborationen	Eine Beschreibung der Interaktionen zwischen den Klassen, die an dem Muster beteiligt sind.	Wird insbesondere zur Beschreibung der Interaktionen zwischen dem CUT und der Prüfsoftware verwendet
Beispielcode	Codesegmente zur Veranschaulichung einer bestimmten Anwendung des Musters	Enthält Einzelheiten zur Integration in die Testumgebung
Bekannte Verwendungen	Beispiele für die Verwendung des Musters	Nicht verwendet, da Testsoftware nur selten veröffentlicht wird
Verwandte Muster	Andere Muster, die für die Bestimmung der Details dieses Musters nützlich sein können	Andere Muster, Design und Test, die für die Implementierung dieses Musters nützlich sind

Tabelle 4-2: Gegenüberstellung von Design Pattern und Testmustern [McG99].

mustern anhand des *Observer Patterns* aus der Softwaretechnik (vgl. [GHJ+94]). Über dieses Muster kann die Beziehung zwischen einem zentralen *Subject* und mehreren davon angängigen *Observern* beschrieben werden. Ändert sich der Zustand des *Subjects*, werden die abhängigen *Observer* benachrichtigt, damit diese sich entsprechend über Abfragen beim *Subject* selbst aktualisieren können [McG99] (vgl. [GHJ+94, S. 293]).

Ausgehend von diesem *Design Pattern* beschreibt MCGREGOR ein zugehöriges Testmuster, das die Vollständigkeit der vom *Subjekt* an den *Observer* gesendeten Benachrichtigungen überprüft. Ebenso beschreibt das Testmuster, wie die Korrektheit der vom *Observer* gesendeten Abfragen sowie die Konsistenz der Daten des *Subjects* und der *Observer* überprüft werden kann.

4.5.4 Methodische Ansätze

Eine Methode für die systematische Dokumentation und Anwendung von Lösungsmustern

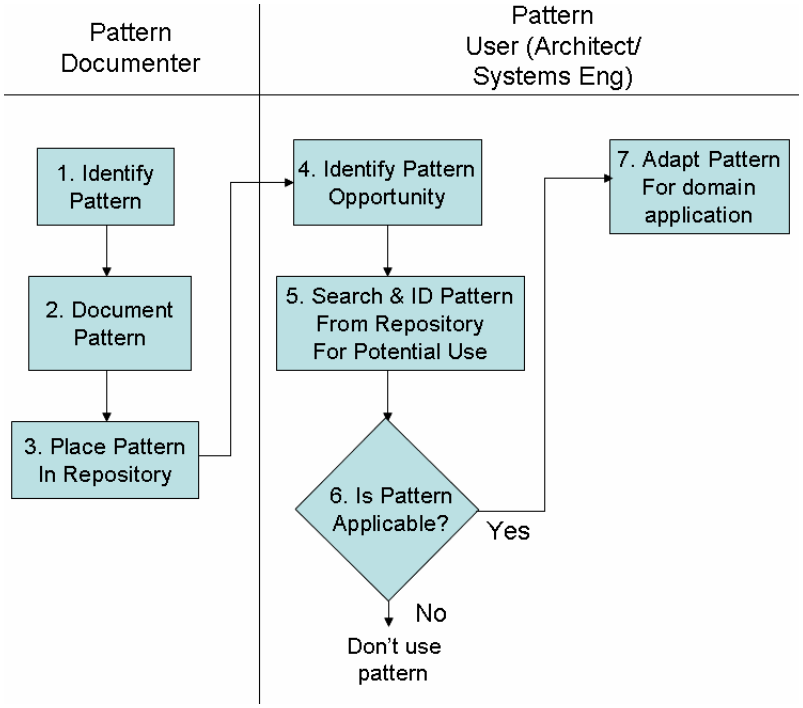


Bild 4-29: Methode zur Erstellung und Anwendung von SE Pattern [Clo06, S. 48].

ist ebenfalls von CLOUTIER beschrieben [Clo06, S. 48 f.]. Wie in Bild 4-29 dargestellt, unterscheidet der Autor zwischen der Rolle *Pattern Documenter* und *Pattern User*, wobei der *Pattern Documenter* für die Identifikation und Dokumentation von Mustern verantwortlich ist, die dann über die Verwendung eines geeigneten Templates in einem Repository zu hinterlegen sind. Der *Pattern User* beginnt mit der Identifikation möglicher Einsatzmöglichkeiten vorhandener Muster. Wenn eine Einsatzmöglichkeit identifiziert ist, kann das Repository auf potentiell geeignete Muster durchsucht werden. Ist ein geeignetes Muster anwendbar, kann dieses für den jeweiligen Anwendungsbereich angepasst werden.

Neben diesem linearen Pfad von der Identifikation eines Musters bis zu dessen Anwendung beschreibt ALEXANDER, dass Muster nicht als statisch anzusehen sind. Demnach ist bei methodischen Ansätzen zu berücksichtigen, dass Erkenntnisse durch die Anwendung der Muster zur Überarbeitung von existierenden Mustern genutzt werden (vgl. [Ale79]).

4.5.5 Bewertung der Ansätze

Die Problemanalyse (s. Kapitel 3.4.4) kommt zu dem Ergebnis, dass *Copy-Paste-Tailoring* Ansätze überwiegen, um vorhandene Spezifikationen und somit das darin implizit enthaltene Wissen wiederzuverwenden. Im Gegensatz dazu sind die zuvor beschriebenen Ansätze potentiell geeignet, um Lösungswissen systematisch externalisieren und wiederverwenden zu können. Da für den erfolgreichen Einsatz von Lösungsmustern die aktive Verwendung und Weiterentwicklung innerhalb der Entwicklungsprozesse entscheidend ist (vgl. [Clo06]), sind hierfür geeignete methodische Ansätze erforderlich. Die einheitliche Strukturierung von Lösungsmuster nach ANACKER in Kombination mit den in Kapitel 4.1 beschriebenen Ansätzen zur Systemmodellierung, sowie den zuvor beschriebenen methodischen Ansätzen nach CLOUTIER, bieten hierfür eine geeignete Grundlage.

FOCKEL et al. [FHK+16] beschreiben musterbasierte Ansätze, die für die Anforderungsspezifikation eine Relevanz haben. Die beschriebenen Anforderungsmuster zielen jedoch darauf ab, bei der Modellierungstätigkeit zu unterstützen. Die Aufbereitung und Wiederverwendung von Lösungswissen wird hier nicht adressiert. Hierfür wäre eine weitergehende Vernetzung der Anforderungsmuster mit den Elementen des Systemmodells notwendig.

Die Verifikationmuster nach SALADO & KANNAN [SK19] zeigen elementare Zusammenhänge zwischen Eigenschaften des zu entwickelnden Systems und Verifikationsaktivitäten. Dieser Ansatz kann als Ausgangspunkt für die Wiederverwendung von Lösungswissen für den Entwurf und die Spezifikation von Tests gesehen werden. Der Ansatz beschreibt die Zusammenhänge jedoch zu abstrakt. Es sind detailliertere Beschreibungen der Systemeigenschaften und der Verifikationsaktivitäten notwendig. Ebenso fehlt hier die Beschreibung einer geeigneten Methode für die Anwendung der identifizierten Muster.

Der Ansatz nach MCGREGOR [McG99] zeigt einen für diese Arbeit relevanten Ansatz, da er Entwurfs- und Testmuster integrativ betrachtet. Durch diese direkte Beziehung der jeweiligen Muster ist die Anwendung der Testmuster jedoch nur möglich, wenn der Entwurf auch musterbasiert erfolgt. Es erscheint sinnvoll für die Wiederverwendung von Lösungswissen Artefakte des RE und der V&V zu vernetzen, der Ansatz nach MCGREGOR aus der Softwaretechnik wäre hierfür jedoch so zu verallgemeinern, dass eine flexiblere Anwendung möglich wird.

4.6 Handlungsbedarf

Das Ergebnis der Problemanalyse sind die in Kapitel 3.6 aufgelisteten Anforderungen an die Arbeit. Der zuvor beschriebene Stand der Technik fasst potentiell geeignete Ansätze

zusammen. Eine Gegenüberstellung der Anforderungen mit dem Stand der Technik ist im folgenden aufgelistet:

A1) Verständlichkeit: Die untersuchten szenariobasierten Ansätze adressieren eine intuitive Modellierung. Jedoch ist für die Modellierung weiterhin Expertenwissen notwendig. In den existierenden Ansätzen werden Anwender ohne dieses Expertenwissen nicht ausreichend unterstützt.

A2) Ganzheitliche Betrachtung des Systemkontextes: Die untersuchten Ansätze betrachten Systeme in ihrem Umfeld. Die Interaktion mit weiteren Systemen, die in unabhängigen Entwicklungsprozessen entwickelt werden, wird in bestehenden Ansätzen nicht ausreichend adressiert.

A3) Orientierung an etablierten Standards in der Steuergeräteentwicklung: Die Ansätze zur Testfallgenerierung oder zur Anforderungsmodellierung können zwar teilweise im Kontext der Steuergeräteentwicklung angewendet werden, die Entwicklung von Artefakten, wie sie in den Referenzprozessen für die Steuergeräteentwicklung gefordert werden, unterstützen die untersuchten Ansätze nicht.

A4) Unterstützung von funktionalen Anforderungen in natürlicher Sprache: Die bestehenden Ansätze zeigen, wie aus Anforderungen in unstrukturierter natürlicher Sprache Testfälle generiert werden können. Ebenso wird die Modellierung von Anforderungen über Anforderungsmuster unterstützt. Es fehlt jedoch ein systematisches Vorgehen und eine anwendungsorientierte methodische Unterstützung.

A5) Unterstützung einer iterativen Spezifikation von Anforderungen: Die untersuchten Techniken im Bereich der System- und Anforderungsmodellierung ermöglichen bereits eine iterative Spezifikation von Anforderungen. Auch hier fehlt jedoch die Beschreibung eines systematischen Vorgehens und eine geeignete methodische Unterstützung.

A6) Unterstützung einer frühen und kontinuierlichen Validierung von Anforderungen: Die untersuchten Ansätze zur Systemmodellierung unterstützen durch die Modellierung des Validierungssystems eine frühe und kontinuierliche Validierung. Hier steht jedoch der Entwurf der dafür notwendigen Testumgebungen und die Analyse der Testergebnisse im Vordergrund. Techniken, die die eigentlichen Testaktivitäten unterstützen, werden hier nicht berücksichtigt.

A7) Unterstützung einer iterativen Spezifikation von Testfällen: Die untersuchten Ansätze zur Systemmodellierung unterstützen ein iteratives Vorgehen. Es besteht jedoch ein Bedarf, Testfälle kontinuierlich mit übergeordneten Validierungszielen abzustimmen, um so die iterative Testfallspezifikation in einem *Advanced Systems* Kontext zu ermöglichen.

A8) Wiederverwenden von Lösungswissen: Für die Wiederverwendung von Lösungswissen bestehen vielversprechende Ansätze. Für die Nutzung dieser Potentiale fehlt jedoch eine geeignete Aufbereitung des relevanten Lösungswissens, das in der Anforderungsanalyse und beim Testentwurf eingesetzt werden kann. Methodische Ansätze sind nur

rudimentär beschrieben.

A9) Werkzeugunterstützung: Die untersuchen Ansätze zur Systemmodellierung können in Model-Based Systems Engineering (MBSE) Werkzeugen realisiert werden. Ebenso existieren für die Anforderungsmodellierung und Testfallgenerierung bereits Werkzeuge. Es fehlt eine geeignete Integration der Werkzeuge, um eine musterbasierte Modellierung innerhalb der Anforderungsanalysephase zu unterstützen.

Die Bewertungen der einzelnen Ansätze und die zuvor dargestellte Gegenüberstellung kommt zu dem Ergebnis, dass keiner der Ansätze alle identifizierten Anforderungen erfüllt. Somit besteht ein Handlungsbedarf für die *Systematik zur integrativen Anforderungsanalyse und Testspezifikation für die Entwicklung von Systemen im Automobilbereich.*

5 Systematik zur integrativen Anforderungsanalyse und Testspezifikation

Dieses Kapitel ist der Kern dieser Arbeit und beschreibt die *Systematik zur integrativen Anforderungsanalyse und Testspezifikation für die Entwicklung von Systemen im Automobilbereich*. Entsprechend der in Kapitel 2 beschriebenen Vorgehensweise wurde die Lösung iterativ erarbeitet. Für eine bessere Verständlichkeit stellt dieses Kapitel die Zwischenergebnisse aus den einzelnen DSR Iterationen zusammenhängend dar. Kapitel 5.1 zeigt zur Einführung eine Übersicht aller Bestandteile der Systematik. Das Fundament der Systematik ist eine Referenzarchitektur für die Systementwicklung, die im Rahmen des MoSyS Projekts entwickelt wurde. Die für diese Arbeit relevanten Aspekte der Referenzarchitektur werden in Kapitel 5.2 vorgestellt. Kapitel 5.3 beschreibt das Vorgehensmodell zur Anwendung der Systematik. Zentrale Elemente der Lösung sind die Ansätze zur integrativen Anforderungsanalyse und Testspezifikation (Kapitel 5.4), die systematische Externalisierung und Wiederverwendung von Lösungswissen (Kapitel 5.5), sowie die Werkzeugunterstützung (Kapitel 5.6).

5.1 Aufbau der Systematik

Der Aufbau der Systematik ist in Bild 5-1 dargestellt. Die Adressaten der Systematik sind Systemarchitekten, die Anforderungen an ein technisches System spezifizieren und analysieren, sowie Test-Designer, die anhand einer definierten Testbasis Testspezifikationen erstellen. Um Systemarchitekten und Test-Designer bei den notwendigen Aktivitäten zu

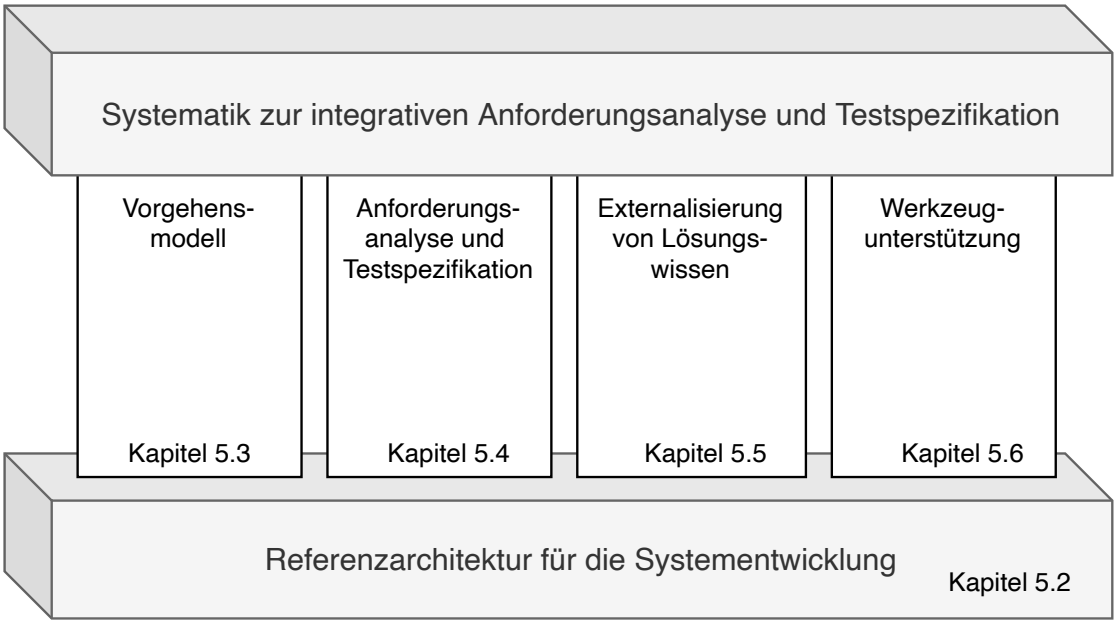


Bild 5-1: Bestandteile der Systematik zur integrativen Anforderungsanalyse und Testspezifikation für die Entwicklung von Systemen im Automobilbereich.

unterstützten, beinhaltet die Systematik vier Säulen, die über die Elemente der Referenzarchitektur miteinander in Beziehung stehen.

Die **Referenzarchitektur** umfasst eine Ontologie, worüber die für die Systematik relevanten Elemente und Konzepte veranschaulicht und miteinander in Beziehung gesetzt werden, sowie aufgabenspezifische Sichten, die eine fokussierte und kollaborative Modellierung durch Systemarchitekten und Test-Designer ermöglichen.

Ein idealisiertes Vorgehen ist durch ein **Vorgehensmodell** beschrieben, das das Vorgehen auf einer Makro-Ebene in Phasen/Meilensteine, Aktivitäten und Resultate gliedert, um eine Integration der Systematik in bestehende Entwicklungsprozesse zu unterstützen. Als Ergänzung zum Vorgehensmodell auf der Makro-Ebene ist für den konkreten Problemlösungsprozess eine Methode zur integrativen Produkt- und Validierungssystemmodellierung beschrieben.

Die Säule **Anforderungsanalyse und Testspezifikation** beschreibt die Technik zur testgetriebenen Modellierung und automatisierten Analyse von funktionalen Anforderungen. Diese Technik ist die Basis für die integrative Produkt- und Validierungssystemmodellierung.

Die **Externalisierung von Lösungswissen** beschreibt die systematische Sicherung von Lösungswissen über ein zentrales Systemmodell mit Hilfe eines digitalen Musterkatalogs, der Systemarchitekten und Test-Designer bei der integrativen Anforderungsanalyse und Testspezifikation unterstützt.

Die Säule **Werkzeugunterstützung** beinhaltet Werkzeuge für die einzelnen Teilbereiche und ermöglicht die Anwendung der Systematik.

5.2 Referenzarchitektur für die Systementwicklung

Der Begriff *Architektur* umschreibt

> *„(...) fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution [ISOd, S.2].“*

Dieser Definition folgend basiert die *integrative Anforderungsanalyse und Testspezifikation* auf einer ganzheitlichen Systemarchitektur, die sowohl das zu entwickelnde Produktsystem als auch die zur Entwicklungszeit benötigten Validierungssysteme umfasst. Die Referenzarchitektur beschreibt die relevanten Elemente der Produkt- und Validierungssysteme und definiert deren Beziehungen. Ausgehend von den Elementen der Referenzarchitektur unterstützen aufgabenspezifische Sichten aus unterschiedlichen Standpunkten den Entwurf und die kontinuierliche Weiterentwicklung projektspezifischer Systemarchitekturen. Dabei sind Anforderungs- und Testspezifikationen die Basis für den Entwurf und die Weiterentwicklung der Systemarchitekturen und zentrale Resultate der Systematik. Daraus folgt, dass die Elemente für eine vollständige und konsistente Spezifikation von Systemanforde-

rungen und Tests als Ausgangspunkt für den Entwurf der Systeme zu verstehen sind. Der eigentliche Systementwurf ist jedoch nicht Teil der Systematik.

5.2.1 Aufbau der Referenzarchitektur

Zur Veranschaulichung der grundsätzlichen Zusammenhänge zeigt Bild 5-2 die unterschiedlichen Dimensionen der Referenzarchitektur. Die erste Dimension betrachtet die Systemtypen. Im Kontext dieser Arbeit sind die zwei Typen Produkt- und Validierungssystem relevant. Die zweite Dimension bildet die Systemhierarchie über unterschiedliche Systemlevel ab. Auf diese Weise ist sowohl die Beschreibung von interagierenden Systemen in einem SoS Kontext als auch die Beschreibung einzelner Komponenten innerhalb eines Systems möglich. Die dritte Dimension betrachtet die Entwicklungszeit, wobei der zeitliche Verlauf über Produkt- und Validierungssystemgenerationen strukturiert ist (vgl. [MGM+22]).

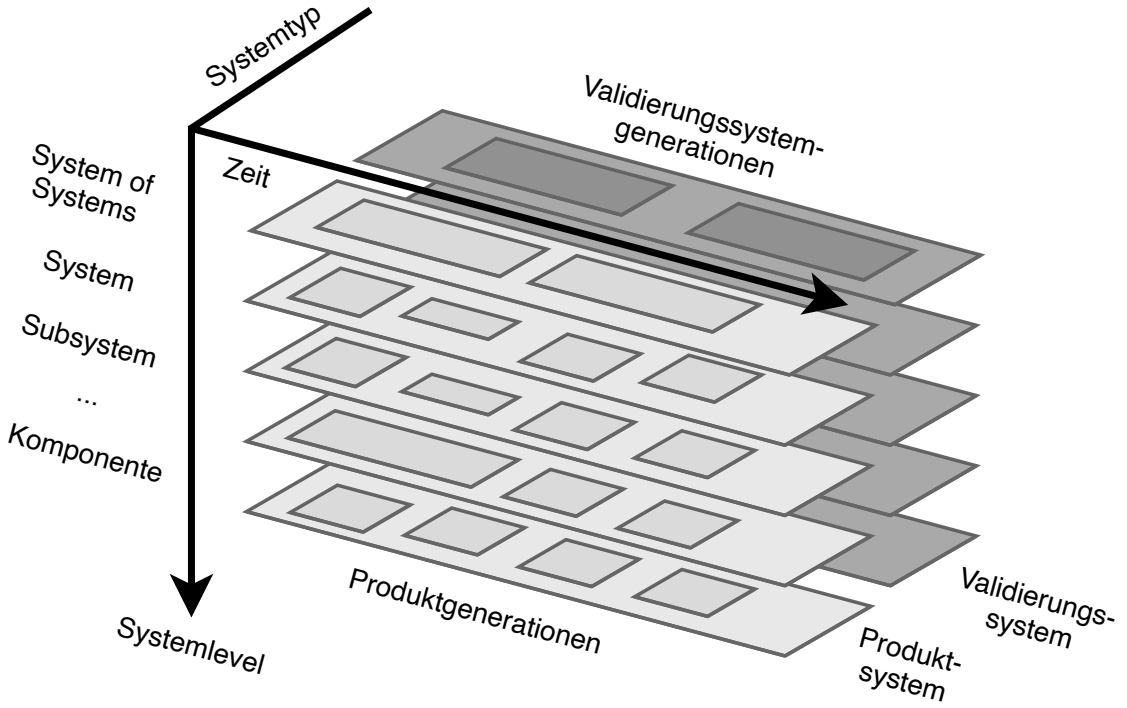


Bild 5-2: Unterschiedliche Dimensionen zur Beschreibung der Referenzarchitektur nach [MGM+22].

Ergänzend zu den drei Dimensionen Systemtyp, Systemlevel und Zeit existiert für jede Instanz eines Produkt- und Validierungssystems eine Kombination aus Problemraum, Anforderungen, Funktionen, sowie einer logischen Systembeschreibung. Der Aufbau des Produktsystems mit einer Trennung von Problem- und Lösungsraum basiert auf etablierten MBSE-Ansätzen (vgl. [PBD+16]). Ausgehend von dem Aufbau des Produktsystems basiert die im Bild 5-3 dargestellte integrative Betrachtung von Produkt- und Validierungssystem

auf den Arbeiten von MANDEL et al. [MBB+21; MGM+22] und ist im Rahmen des MoSyS Projekts entstanden (vgl. Kapitel 4.1.3).

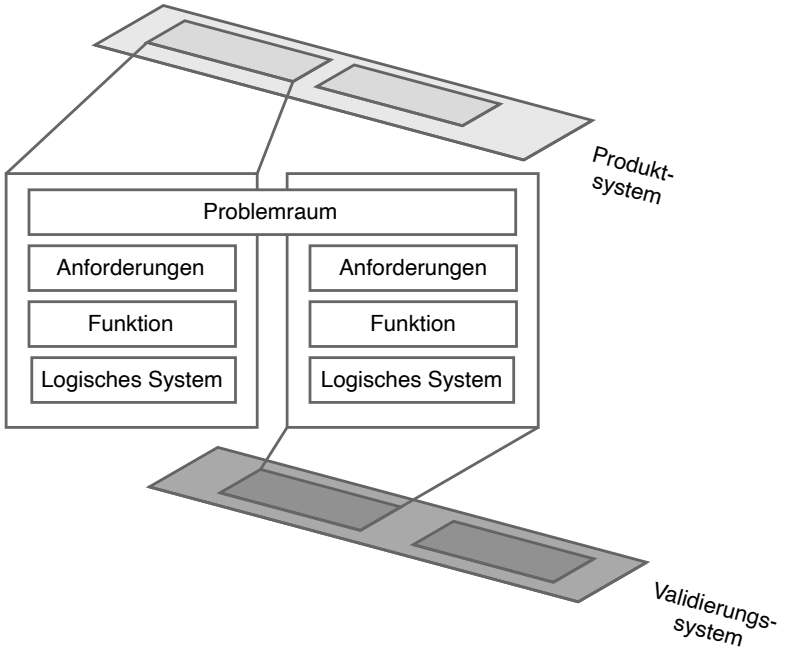


Bild 5-3: Integrative Betrachtung des Produkt- und Validierungssystems.

Wie im Bild 5-3 dargestellt, sind die Elemente des Problemraums sowohl der Ausgangspunkt für die Ausgestaltung des Produkt- als auch des Validierungssystems, wodurch das Produkt- und Validierungssystem über den Problemraum miteinander in Beziehung stehen.

Analog zum Aufbau des Produktsystems ist die Beschreibung des Validierungssystems strukturiert in *Anforderungen* und *Funktionen*, die durch ein *logisches System* realisiert werden. Hierbei werden Anforderungen an das Validierungssystem durch *Validierungsbedarfe und Validierungsziele* beschrieben. Die Funktion des Validierungssystems ist definiert durch *Testfälle und Testszenarien*. Der logische Aufbau resultiert aus den logischen Elementen der *Testumgebungen*.

Die Elemente und Beziehungen innerhalb der Referenzarchitektur sind durch die im folgenden beschriebene Ontologie konkretisiert. Diese Ontologie wird nach HOLT & PERRY als eine domänenspezifische Sprache für die zielgerichtete Kommunikation und integrative Systemmodellierung verstanden (s. Kapitel 4.1.1).

5.2.2 Ontologie und Standpunkte für die integrative Modellierung von Produkt- und Validierungssystem

Ausgehend von der zuvor gezeigten Darstellung in Bild 5-3 orientiert sich der Aufbau der Ontologie in Problemraum, Produkt- und Validierungssystem[1]. Eine Aufteilung der Ontologie erfolgt durch die drei im folgenden beschriebenen Standpunkte, die die Perspektiven der Stakeholder, des Systemarchitekten und des Test-Designers repräsentieren. Diese Standpunkte ermöglichen die Definition von Sichten auf das Systemmodell, die wiederum Basis für die Durchführung von Modellierungsaktivitäten sind. Dabei orientieren sich die Standpunkte zwar an den Perspektiven eines Stakeholders, Systemarchitekten oder Test-Designers, wodurch jedoch keine Zuordnung der Standpunkte zu den jeweiligen Rollen getroffen wird. Die aus dem Standpunkt des Stakeholders resultierenden Sichten sind bspw. sowohl für den Test-Designer als auch den Systemarchitekten relevant, um gemeinschaftlich Validierungsbedarfe zu definieren.

Problemraum (Standpunkt Stakeholder): Der verbindende Standpunkt des Stakeholders ist in Bild 5-4 dargestellt. Entsprechend den Rahmenbedingungen für die Entwicklung von Systemen im Automobilbereich sind die *Stakeholder-Anforderungen* ein zentrales Element innerhalb des Problemraums, um Systemeigenschaften auf Basis von Geschäftszielen zu definieren (vgl. Kapitel 3.1.2). Im Kontext dieser Arbeit liegt der Fokus auf funktionalen Stakeholder-Anforderungen, wobei die Stakeholder-Anforderungen im allgemeinen die Basis für den Entwicklungsauftrag an ein Tier1 Zulieferunternehmen darstellen (vgl. Kapitel 3.2). Da die Stakeholder-Anforderungen von unterschiedlichen externen als auch internen Stakeholdern definiert werden können, beinhaltet die Ontologie das Element *Stakeholder* zur expliziten Modellierung und Differenzierung unterschiedlicher Stakeholder. Jeder Stakeholder kann *Stakeholderbedarfe* definieren, die die Basis für die Beschreibung von *Anwendungsfällen* sind. Die Anwendungsfälle können wiederum durch *Anwendungsszenarien* konkretisiert und den Stakeholder-Anforderungen zugeordnet werden, um diese zu strukturieren. Zur Unterstützung einer Kontextmodellierung ist die Zuweisung von *interagierenden Systemen* zu dem jeweiligen Anwendungsfall möglich. Ein wesentlicher Aspekt des Stakeholder-Standpunkts ist die Vernetzung der Elemente des Problemraums mit den *Validierungsbedarfen* des Validierungssystems. Hierdurch entsteht die Möglichkeit, Validierungsbedarfe auf Basis der Stakeholderbedarfe festzulegen, noch bevor mit der Ausgestaltung des Systems begonnen wurde. Die Validierungsbedarfe sind somit ein Schnittstellenelement, das den Standpunkt des Stakeholders mit dem Standpunkt des Test-Designers in Beziehung setzt. Die gestrichelten Linien in Bild 5-4 definieren die Schnittstellenelemente. In gleicher Weise sind die Stakeholder-Anforderungen als ein Schnittstellenelement zu verstehen, da diese Anforderung der Ausgangspunkt zur Beschreibung des Produktsystems sind.

Produktsystem (Standpunkt Systemarchitekt): Wie oben bereits angedeutet, basieren die in Bild 5-5 dargestellten Ontologie-Elemente des Produkts auf etablierten MBSE Ansätzen

[1] Eine Übersicht der Ontologie ist im Anhang A1 dargestellt.

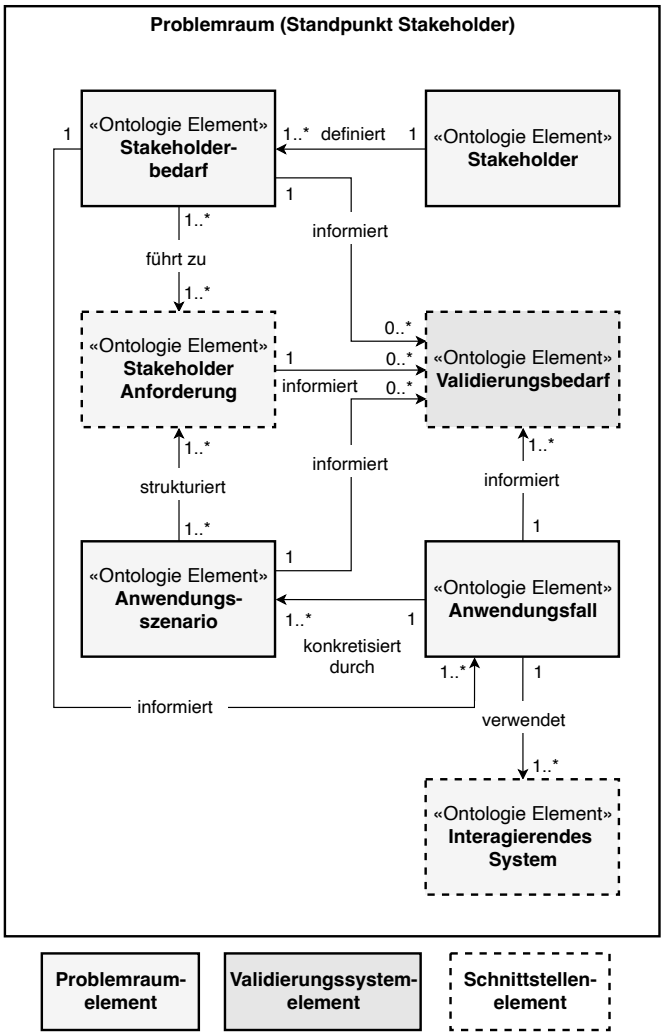


Bild 5-4: Ontologie-Elemente zur Beschreibung des Stakeholder-Standpunkts.

(vgl. [Est+07; Wei16; PBD+16]) und ermöglichen die Modellierung von *Systemanforderungen*, *Funktionen* und des *logischen Systems*. Über die Stakeholder-Anforderungen besteht eine Schnittstelle zum Problemraum, und über die Vernetzung mit Validierungsbedarfen und durch den Aufbau des logischen Systems besteht eine Schnittstelle zum Validierungssystem.

Validierungssystem (Standpunkt Test-Designer): Der in Bild 5-6 dargestellte Standpunkt des Test-Designers ist die Basis für die Modellierung des Validierungssystems. Ein zentrales Element sind hier *Validierungsbedarfe*, die aus dem Problemraum und den Elementen des Produktsystems abgeleitet werden. Die Validierungsbedarfe können über konkrete *Validierungsziele* operationalisiert werden. Zur Beschreibung von konkreten Testaktivitäten werden *Testfälle* verwendet. Über das Element *Test* erfolgt eine Zuordnung von Testfällen zu Testumgebungen, die für die Umsetzung der Testfälle benötigt werden. Der Aufbau der Testumgebungen kann durch konkrete *Testmodelle* beschrieben werden, die wiederum aus den interagierenden Systemen aus dem Problemraum oder bereits vorhandenen Elementen des logischen Systems des Produktsystems abgeleitet werden. Auf diese Weise ist es

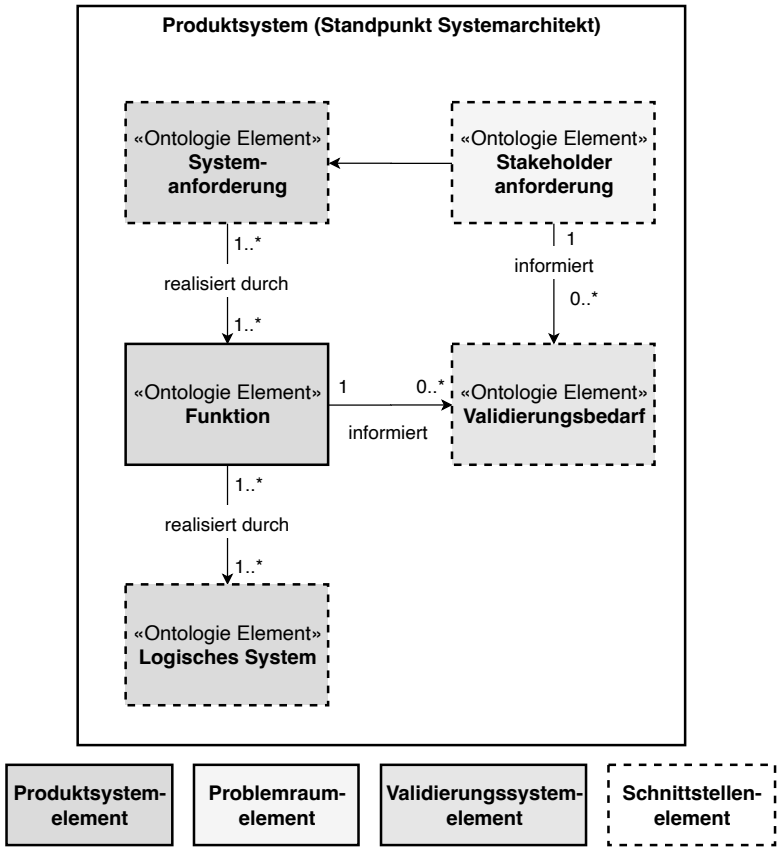


Bild 5-5: Ontologie-Elemente zur Beschreibung des Standpunkts des Systemarchitekten.

möglich, die Testumgebungen entsprechend dem aktuellen Entwicklungsfortschritt zu modellieren. Ist bspw. ein Teilsystem aus einer vorherigen Entwicklungsiteration realisiert, setzt sich die Testumgebung zusammen aus dem bereits vorhandenen Teilsystem sowie Modellen von noch zu entwickelnden Systemelementen und den interagierenden Systemen. Dabei resultieren die interagierenden Systeme aus den zu unterstützenden Anwendungsfällen (Standpunkt Stakeholder). Dabei sind die interagierenden Systeme nicht Teil des zu entwickelnden Systems, können jedoch eine Funktion erfüllen, die für die Realisierung und Validierung des Anwendungsfalls von Bedeutung ist.

Neben der expliziten Modellierung von Testumgebungen unterstützt der in Bild 5-6 dargestellte Standpunkt eine Unterscheidung in einen Validierungs- und Verifikationspfad. Für die **Verifikation** von Stakeholder Anforderungen erfolgt eine direkte Zuordnung von Stakeholder-Anforderungen zu dem Ontologie-Element Test. Dem Test können wiederum ein oder mehrere Testfälle zugeordnet werden. Auf diese Weise entsteht einerseits eine bidirektionale Traceablity zwischen den Stakeholder-Anforderungen und Testfällen und andererseits zwischen den Testfällen und der Testumgebung bis hin zu den Elementen des logischen Produktsystems. Die **Validierung** basiert auf den Validierungsbedarfen und den daraus abgeleiteten Validierungszielen. Zur Verlinkung von Validierungszielen mit konkreten Testfällen werden *Testszenarien* verwendet. Die Ausführung der Testfälle als Teil eines Testszenarios oder eines Tests auf Basis von Stakeholder-Anforderungen führt zu

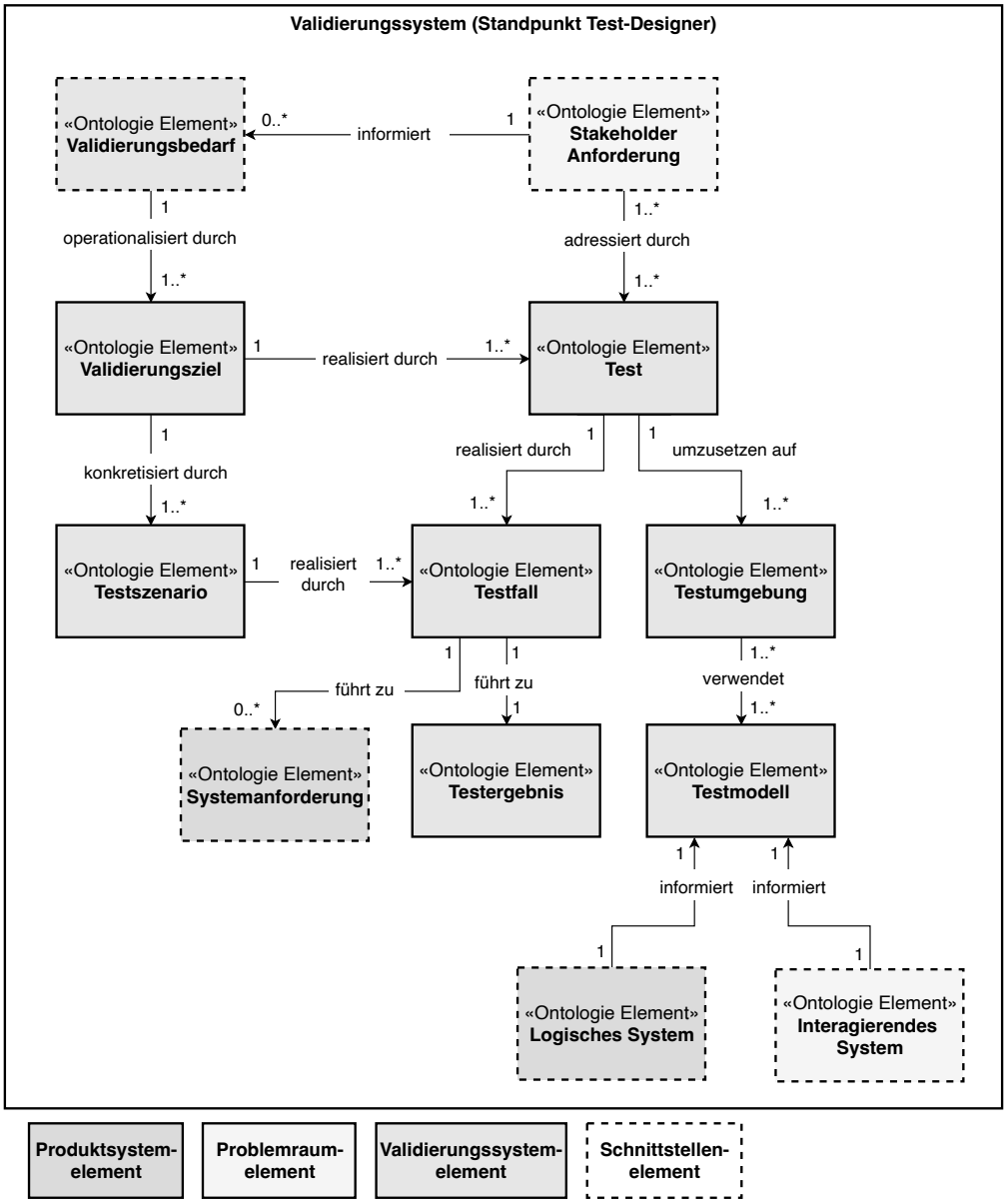


Bild 5-6: Ontologie-Elemente zur Beschreibung des Standpunkts des Test-Designers

Testergebnissen. Zusätzlich resultiert die Ausführung von Testfällen, dem testgetriebenen Entwicklungsansatz folgend, zu Systemanforderungen (s. Methode und Modellierungsansatz in Kapitel 5.3 und 5.4).

Eine Übersicht der in der Ontologie verwendeten Begriffe ist im Folgenden dargestellt:

- **Stakeholder**: Person oder Organisation, die ein Recht, einen Anteil, einen Anspruch oder ein Interesse an einem System oder an dessen Eigenschaften hat, die ihren Bedürfnissen und Erwartungen entsprechen [ISOe; DRF+15].
- **Stakeholderbedarf**: Der Stakeholderbedarf wird durch die Kommunikation mit externen oder internen Stakeholdern ermittelt, um deren Erwartungen, Bedürfnisse, Anforderungen, Werte, Probleme, Fragen und wahrgenommene Risiken und Chancen

zu verstehen [DRF+15].

- **Stakeholder-Anforderung**: Anforderungen verschiedener Stakeholder, die das Projekt bestimmen, einschließlich der erforderlichen Systemfähigkeiten, -funktionen und/oder -dienste, Qualitätsstandards, Systembeschränkungen sowie Kosten- und Terminbeschränkungen. Die Anforderungen der Stakeholder können in der Stakeholder Anforderungsspezifikation festgehalten werden [DRF+15].
- **Anwendungsfall**: Ein Anwendungsfall beschreibt die Interaktion zwischen dem Anwender eines Systems und/oder die Interaktion zwischen Systemen und unterstützt damit die Beschreibung von Anforderungen an das Systemverhalten (vgl. [ISOc]).
- **Anwendungsszenario**: Ein Anwendungsszenario beschreibt eine spezifische Interaktion zwischen Anwender und System und/oder zwischen Systemen und ermöglicht so die Konkretisierung von Anwendungsfällen. Ein Anwendungsfall kann über mehrere Anwendungsszenarien konkretisiert werden (vgl. [Sut03]).
- **Interagierendes System**: Systeme, die mit dem zu entwickelnden System interagieren, um eine integrierte Gesamtfunktionalität zu realisieren (vgl. [ABK+16]). Interagierende Systeme dienen zur Beschreibung von Anwendungsfällen und Anwendungsszenarien.
- **Systemanforderung**: Eine eindeutige und verifizierbare Aussage, die eine Systemeigenschaft oder -einschränkung identifiziert und den Entwurf des Systems leitet (nach [VDA; DRF+15]).
- **Funktion**: Die Funktion entspricht dem beabsichtigten Verhalten und beschreibt eine voraussichtliche Wirkung auf die Umwelt (vgl. [PBS+13]).
- **Logisches System**: Beschreibt über die Verknüpfung von logischen Elementen den Aufbau des Systems, wobei logische Elemente eine oder mehrere Funktionen realisieren können (vgl. [Kai13, S. 68]).
- **Validierungsbedarf**: Beschreibt eine für die Validierung relevante Eigenschaft des zu entwickelnden Systems, die sich aus den Erwartungen der Stakeholder ableiten (vgl. [MBB+21]).
- **Validierungsziel**: Ein Validierungsziel operationalisiert ein oder mehrere Validierungsbedarfe und definiert den Umfang der Validierung für eine spezifische Entwicklungsiteration.
- **Test**: Ein Test ermittelt Systemeigenschaften eines (zu untersuchenden) Systems und liefert Erkenntnisse über das System. Ein Test umfasst stets einen Testfall und eine Testumgebung (vgl. [Ebe15, S. 136]).
- **Testfall**: Ein Testfall beschreibt die für die Ausführung notwendigen Randbedingungen, Eingangsgrößen und das erwartete Systemverhalten eines Systems (vgl. [Ebe15, S. 136] und [SLS11, S. 10]).

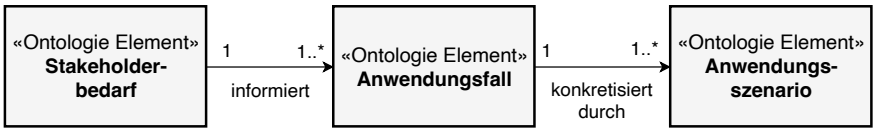
- **Testszenario**: Kombination mehrerer Testfälle mit dem Ziel, eine Systemeigenschaft zu validieren (vgl. [Kan03]).
- **Testergebnis**: Ergebnis der Ausführung eines Testfalls, sowohl auf Basis eines Tests und der verbundenen Stakeholder Anforderung, als auch als Teil eines Testszenarios.
- **Testumgebung**: Eine Testumgebung beinhaltet die Gesamtheit aller physischen und virtuellen Modelle bzw. Originale, die notwendig sind, um einen oder mehrere Testfälle durchzuführen und das erwartete Systemverhalten zu erfassen [Ebe15, S. 136].
- **Testmodell**: Repräsentiert die einzelnen Bestandteile der Testumgebung. Bestandteile sind hier die physischen und virtuellen Modelle als auch Originale auf Basis des logischen Systems und der identifizierten interagierenden Systeme.

5.2.3 Sichten auf das Modell

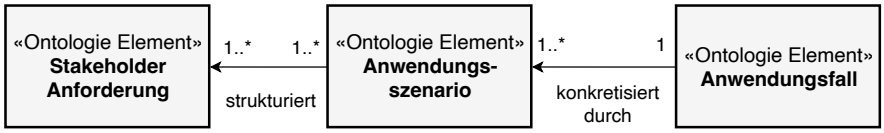
Ausgehend von den drei zuvor beschriebenen Standpunkten sind im folgenden Sichten (vgl. Kapitel 4.1.1) zur Unterstützung einer aufgabenspezifischen Modellierung durch den Systemarchitekten und Test-Designer definiert. Die Definition der Sichten basiert auf dem Ansatz der integrativen Anforderungsanalyse und Testspezifikation. Somit steht insbesondere die Modellierung und Kontextualisierung von Stakeholder-Anforderungen als Basis für die Entwicklung, sowie die Modellierung des notwendigen Validierungssystems und der Systemanforderungen des Produktsystems, im Vordergrund. Bild 5-7 strukturiert die Sichten in insgesamt 7 Ebenen, wobei die obere und untere Ebene die Modellierung des Problemraums bzw. des Produktsystems unterstützten. Die 5 mittleren Ebenen die adressieren die Modellierung des Validierungssystems.

Die einzelnen Sichten sind im folgenden beschrieben:

- **Systemkontext und Anwendungsfälle (Problemraum)**: Diese Ebene beinhaltet drei Sichten. Die erste Sicht (Anwendungsfälle und Anwendungsszenarien) unterstützt die Modellierung von Anwendungsfällen aus der Perspektive des Stakeholders, wobei jeder Anwendungsfall auf einem spezifischen Stakeholderbedarf basiert.



  Die zweite Sicht (Stakeholder-Anforderungen) unterstützt die Strukturierung der Stakeholder-Anforderungen durch die Vernetzung mit den zuvor definierten Anwendungsfällen und -szenarien.



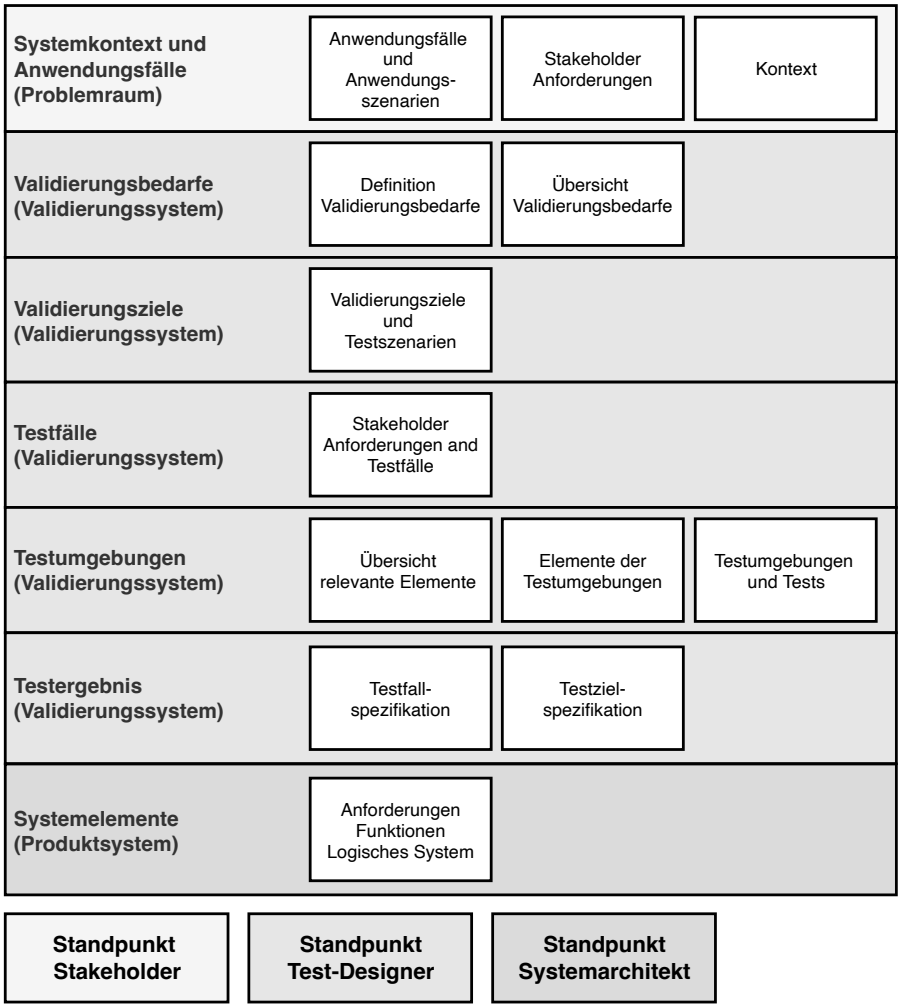
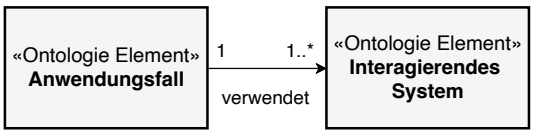
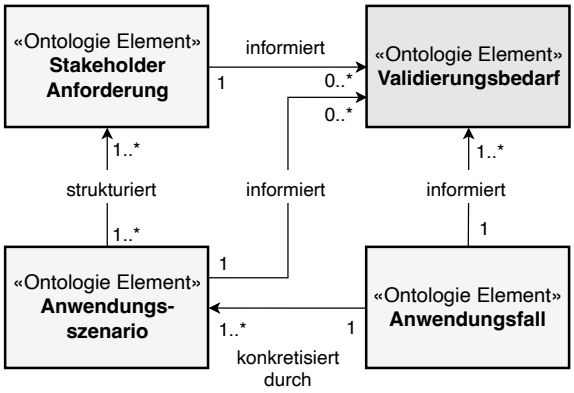


Bild 5-7: Sichten auf das Systemmodell.

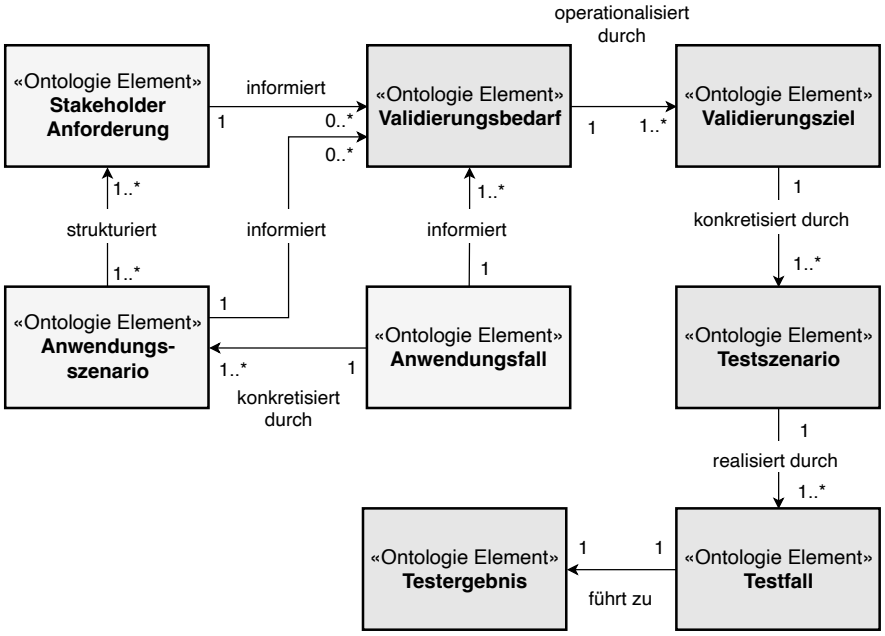
Die dritte Sicht (Kontext) unterstützt die Kontextmodellierung über die Vernetzung von Anwendungsfällen mit den interagierenden Systemen.



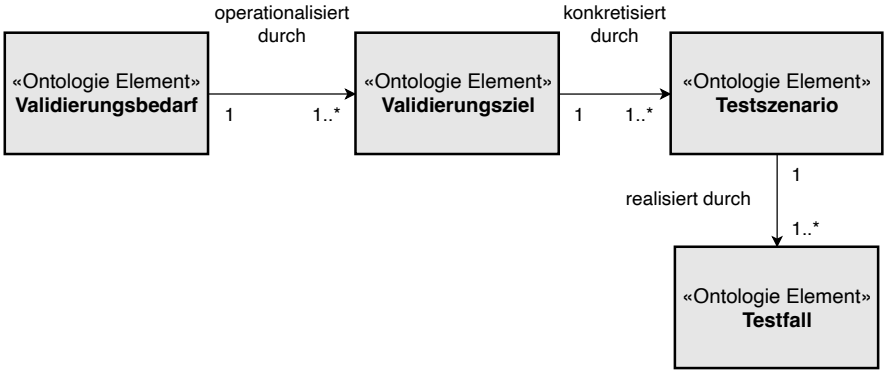
- **Validierungsbedarfe (Validierungssystem)**: Diese Ebene beinhaltete zwei Sichten. Die erste Sicht (Definition Validierungsbedarfe) ermöglicht die Definition von Validierungsbedarfen auf Basis der Elemente des Problemraums.



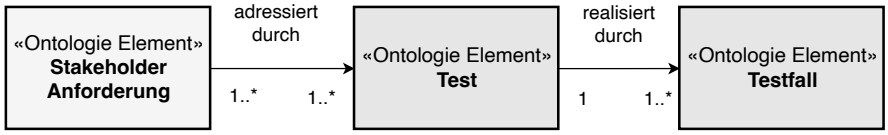
Die zweite Sicht (Übersicht Validierungsbedarfe) stellt eine Übersicht aller bereits angelegten Validierungsbedarfe bereit und beinhaltet die zugehörigen Testergebnisse aus vorangegangenen Entwicklungsiterationen. Die Testergebnisse stehen über das Validierungsziel und den damit vernetzten Testszenarien und Testfällen mit den Validierungsbedarfen in Beziehung.



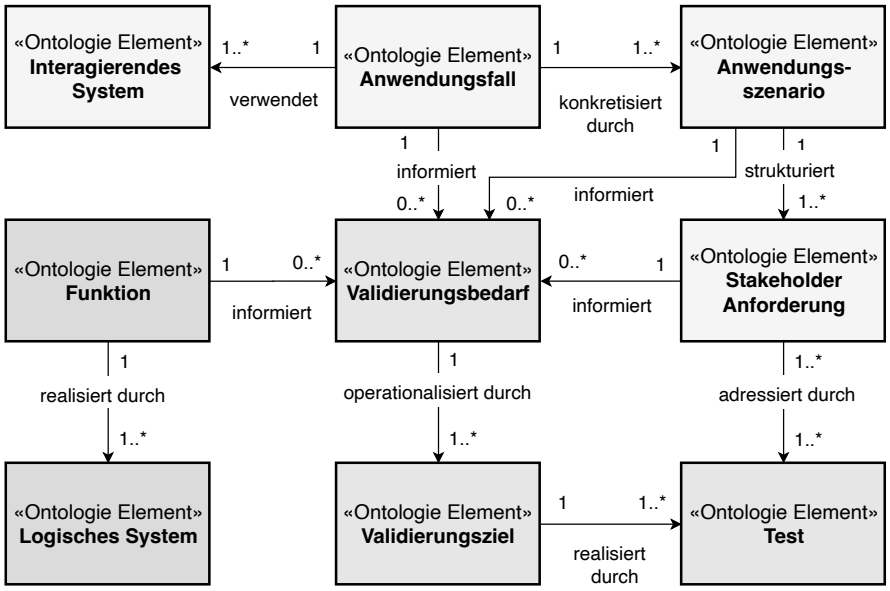
- **Validierungsziele (Validierungssystem)**: Diese Ebene beinhaltet eine Sicht (Validierungsziele und Testszenarien) für die Definition von Validierungszielen, ausgehend von den zuvor festgelegten Validierungsbedarfen. Neben der Definition der Validierungsziele unterstützt die Sicht das Anlegen und Vernetzen von Testszenarien und den zur Ausführung benötigten Testfällen.



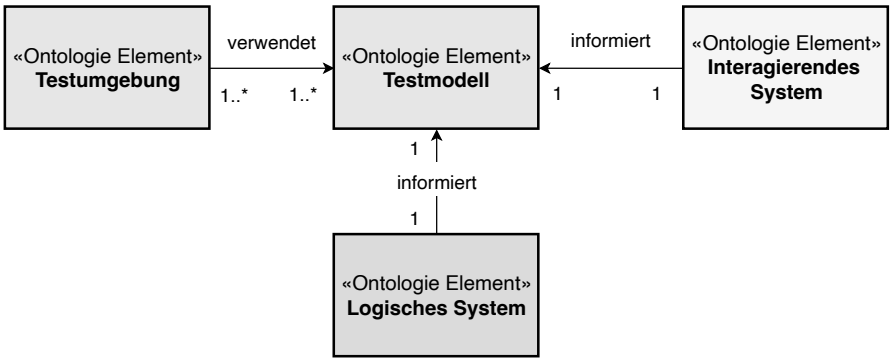
- **Testfälle (Validierungssystem)**: Diese Ebene enthält eine Sicht (Stakeholder-Anforderungen und Testfälle), um das Anlegen neuer Tests und zugehörigen Testfällen auf Basis der im Problemraum identifizierten Stakeholder-Anforderungen zu unterstützen.



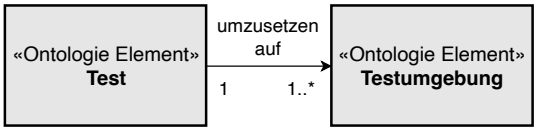
- **Testumgebungen (Validierungssystem)**: Diese Ebene enthält drei Sichten. Die erste Sicht (Übersicht relevante Elemente) ermöglicht die Analyse aller Elemente, die ausgehend von einem Validierungsziel für die Testumgebung von Bedeutung sein können. Mit dem Validierungsziel sind die für die Realisierung notwendigen Tests vernetzt. Ebenso besteht eine Beziehung zu dem Validierungsbedarf, der wiederum mit allen für die Validierung relevanten Elementen vernetzt ist.



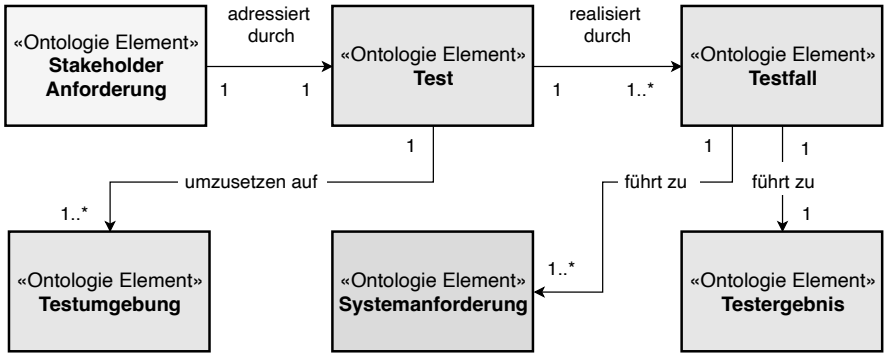
Die zweite Sicht (Elemente der Testumgebungen) dient zum Vernetzen der Testumgebungen mit den jeweils relevanten Testmodellen, die sich aus den im Problemraum definierten interagierenden Systemen, bzw. den logischen Elementen des Produktsystems ableiten lassen.



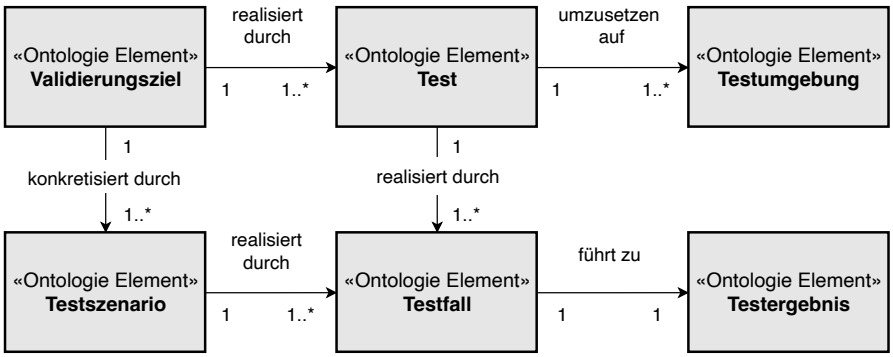
Die dritte Sicht (Testumgebungen und Tests) ermöglicht die Vernetzung der Tests mit den Testumgebungen.



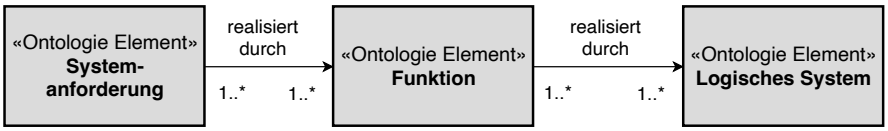
- **Testergebnisse (Validierungssystem)**: Diese Ebene beinhaltet zwei Sichten. Die erste Sicht (Testfallspezifikation) dient als Basis für die Verifikationsaktivität. Ausgehend von einer Stakeholder-Anforderung kann über diese Sicht die Ausführung eines Testfalls über das dazugehörige Testergebnis dokumentiert werden. Die Ausführung des Testfalls auf einer Testumgebung führt dabei zu einer Systemanforderung.



Ergänzend zur Testfallspezifikation unterstützt die zweite Sicht (Testzielspezifikation) die Validierungsaktivität und ermöglicht die Dokumentation, ob die definierten Validierungsziele durch die spezifizierten Systemanforderungen eingehalten werden.



- **Systemelemente (Produkt)**: Diese Ebene enthält eine Sicht (Anforderungen, Funktionen, Logisches System) und dient zum Vernetzen von Anforderungen, Funktionen und logischen Systemelementen.



5.3 Vorgehensmodell

Das Vorgehensmodell unterstützt Systemarchitekten und Test-Designer in der kollaborativen Systemmodellierung und der Analyse und Spezifikation von Anforderungen und Tests. Dabei wird das Ziel verfolgt, vollständige und konsistente Spezifikationen als Basis für das nachgelagerte Systemdesign und die Systemvalidierung zu erstellen. Das Vorgehensmodell orientiert sich an etablierten Standards für die Entwicklung von Systemen im Automobilbereich und unterstützt die Integration der Systematik in bestehende Entwicklungsprozesse. Wie in Bild 5-8 dargestellt ist das Vorgehensmodell hierfür unterteilt in *Phasen/Meilensteine*, *Aktivitäten* und *Resultate*, die im folgenden näher beschrieben sind.

5.3.1 Phase 1: Erfassen und Kontextualisieren von Stakeholder-Anforderungen

Die erste Phase umfasst drei Aktivitäten und beginnt mit dem Entwicklungsauftrag durch einen OEM. Grundlage für den Entwicklungsauftrag sind Stakeholder-Anforderungen, die durch den OEM an den Zulieferer übergeben werden. Diese Anforderungen spezifizieren die Rahmenbedingungen für die Systementwicklung und ermöglichen die spätere Integration des zu entwickelnden Systems mit weiteren interagierenden Systemen.

Die erste Aktivität in dieser Phase ist die Modellierung aller für den Entwicklungsauftrag relevanten Stakeholder und deren Bedarfe einschließlich der identifizierten Anwendungsfälle und konkreter Anwendungsszenarien. Die oben beschriebene Sicht *Anwendungsfälle und Anwendungsszenarien* unterstützt diese Aktivität, die gemeinschaftlich vom Systemarchitekten und Test-Designer auszuführen ist. Das Ziel dieser Aktivität ist die Modellierung

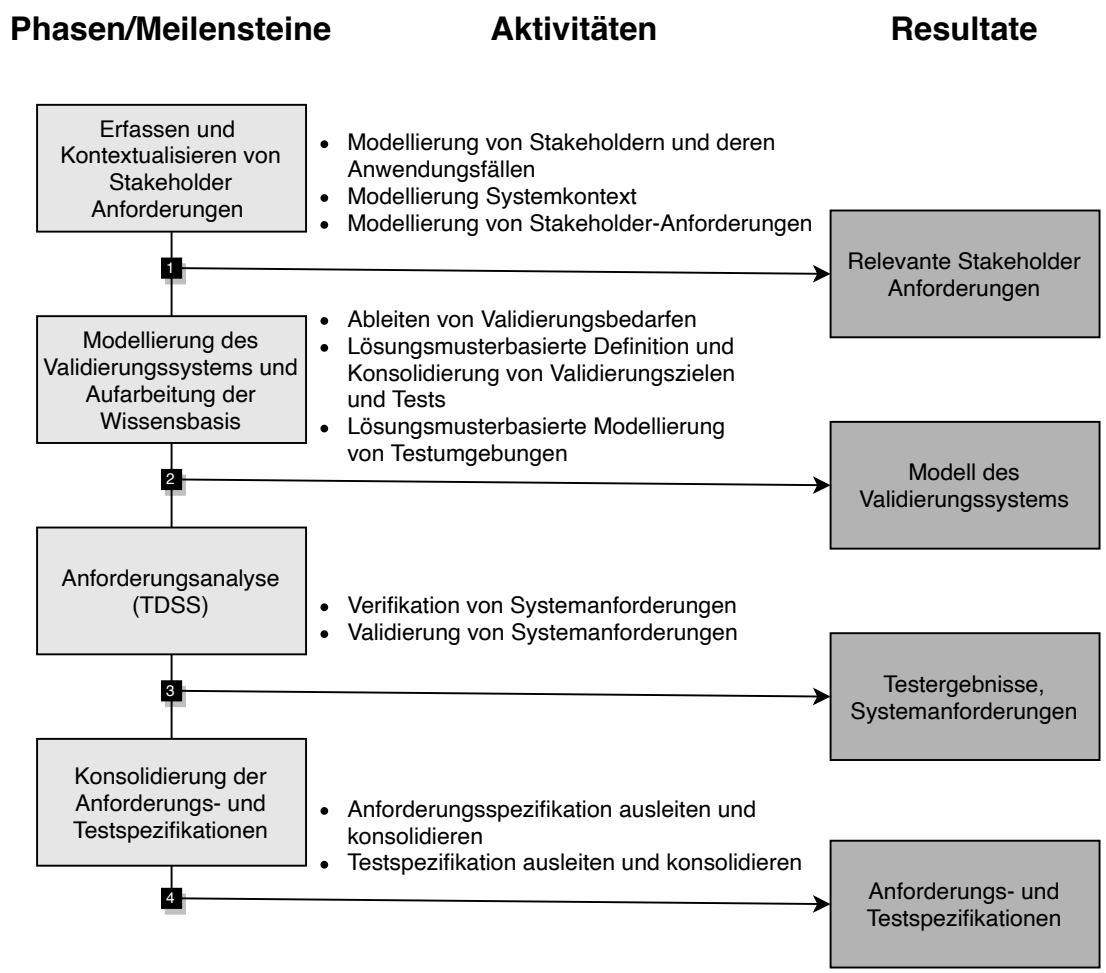


Bild 5-8: Vorgehensmodell für die integrative und iterative Analyse von Anforderungen und Spezifikation von Tests.

der erwarteten Systemeigenschaften aus der Perspektive der Stakeholder.

Die zweite Aktivität dient der Kontextmodellierung, um den gesamten für den Entwicklungsauftrag relevanten Kontext zu erfassen. Dabei kann der Kontext sowohl interagierende Systeme innerhalb als auch außerhalb des Fahrzeugs umfassen. Das Zusammenspiel des zu entwickelnden Systems mit den interagierenden Systemen innerhalb und außerhalb des Fahrzeugs realisiert eine durch den Endkunden wahrnehmbare Gesamtfunktionalität, wodurch die umfassende Kontextmodellierung insbesondere für die Validierung und die dafür notwendigen Testumgebungen von große Bedeutung ist. Die zweite Aktivität wird unterstützt durch die Sicht *Kontext* und ist durch den Systemarchitekten auszuführen.

Die dritte Aktivität dient zur Modellierung aller Stakeholder-Anforderungen auf Basis der zuvor durch den Systemarchitekten und den Test-Designer festgelegten Anwendungsfälle und -szenarien. Die Sicht *Stakeholder-Anforderungen* unterstützt diese Aktivität, die durch den Systemarchitekten auszuführen ist.

Das Resultat der ersten Phase ist eine strukturierte Spezifikation der relevanten Stakeholder-

Anforderungen.

5.3.2 Phase 2: Modellierung des Validierungssystems und Aufarbeitung der Wissensbasis

Die zweite Phase umfasst ebenfalls drei Aktivitäten, um auf Basis der Informationen aus der ersten Phase das für die Entwicklung des Produktsystems notwendige Validierungssystem zu modellieren. Ebenso ist die Aufbereitung der Wissensbasis durch die Anwendung und Aktualisierung des digitalen Musterkatalogs Teil dieser Phase.

In der ersten Aktivität erfolgt die Ableitung von Validierungsbedarfen. Die Sicht *Definition Validierungsbedarfe* unterstützt diese Aktivität, die von dem Systemarchitekten und dem Test-Designer gemeinschaftlich durchzuführen ist. Das Ergebnis der Aktivität sind Validierungsbedarfe, die mit den Stakeholder-Anforderungen, Anwendungsfällen oder Anwendungsszenarien vernetzt sind.

Auf Basis der zuvor abgeleiteten Validierungsbedarfe erfolgt in der zweiten Aktivität eine Definition und Konsolidierung von Validierungszielen und Tests. Diese Aktivität wird unterstützt durch den digitalen Lösungsmusterkatalog (s. Kapitel 5.5). Dieser Lösungsmusterkatalog stellt Lösungsmuster bereit, um den Test-Designer bei der Modellierung von Testszenarien zu unterstützen, die für das Erreichen der Validierungsziele notwendig sind. Ebenso wird der Test-Designer durch den Lösungsmusterkatalog bei der Modellierung von Tests und dazugehörigen Testfällen für die Verifikation von Stakeholder-Anforderungen unterstützt. Zur Durchführung der Aktivität stehen die Sichten *Validierungsziele und Testszenarien*, sowie *Stakeholder-Anforderungen und Testfälle*, zur Verfügung. Ergänzend zu den Sichten und dem Lösungsmusterkatalog wird der Test-Designer in dieser Phase durch ein Werkzeug für die Testfallgenerierung unterstützt (s. Kapitel 5.6.2). Dieses Werkzeug generiert aus den vorhandenen Stakeholder-Anforderungen Testfälle, die durch den Test-Designer über die Sicht *Stakeholder-Anforderungen und Testfälle* zum Modell hinzugefügt und vernetzt werden können.

Die dritte Aktivität umfasst die Modellierung von Testumgebungen, die für die Verifikation- und Validierungsaktivität benötigt werden. Für die Modellierung der Testumgebungen wird der Test-Designer durch die Sicht *Übersicht relevante Elemente* unterstützt. Diese Sicht zeigt die mit dem Validierungsziel vernetzten Elemente, die von dem Test-Designer zu sichten und hinsichtlich der Relevanz für die Testumgebung zu bewerten sind. Ausgehend von dieser Bewertung wird der Test-Designer über die Sicht *Elemente der Testumgebungen* beim Anlegen und Vernetzen neuer Testumgebungen unterstützt. Zudem kommt auch hier der Lösungsmusterkatalog zum Einsatz. Das Vernetzen der neu angelegten Testumgebungen mit den zuvor definierten Tests und generierten Testfällen schließt die dritte Aktivität ab und wird durch die Sicht *Testumgebungen und Tests* unterstützt.

Resultat der zweiten Phase ist ein Modell des Validierungssystems als Basis für die Verifikation und Validierung.

5.3.3 Phase 3: Anforderungsanalyse

Ausgehend von dem zuvor modellierten Validierungssystem umfasst die dritte Phase die Aktivitäten zur Verifikation und Validierung der Anforderungsspezifikation. Die Grundlage für die Aktivitäten in dieser Phase ist dabei die Methode zur testgetriebenen Spezifikation von Systemanforderungen (Test-Driven Scenario Specification (TDSS) - s. Kapitel 5.4.1). In Kombination mit einem Werkzeug (s. Kapitel 5.6.2) unterstützt diese Methode eine formalisierte Spezifikation und automatisierte Analyse der Systemanforderungen, getrieben über die zuvor generierten und innerhalb des Validierungssystems vernetzten Testfälle.

Die TDSS Methode wird sowohl für die Verifikation als auch für die Validierung verwendet. Für die Verifikation dient die Sicht *Testfallspezifikation*. Einerseits stellt diese Sicht die generierten Testfälle als Eingangsinformationen für die TDSS Methode bereit, andererseits werden die Testergebnisse und Systemanforderungen als Ergebnis der TDSS Methode über diese Sicht zum Systemmodell hinzugefügt und vernetzt. Da die Systemanforderungen als Teil des Produktsystems durch die Anwendung der TDSS Methode entstehen, entsteht über diese Phase eine enge Verzahnung zwischen dem Validierungs- und Produktsystem.

In gleicher Weise ermöglicht die *Testzielspezifikation* eine Validierung der Anforderungsspezifikation. Anstatt einzelner Testfälle sind hier die oben definierten Testszenarien über eine geeignete Kombination der Testfälle die Eingangsinformation für die TDSS Methode. Analog zur Testfallspezifikation ermöglicht die Testzielspezifikation das Hinzufügen der Validierungsergebnisse zum Systemmodell.

Die Resultate der dritten Phase sind verifizierte und validierte Systemanforderungen einschließlich der vernetzten Testergebnisse.

5.3.4 Phase 4: Konsolidierung der Anforderungs- und Testspezifikationen

Nach Abschluss der Anforderungsanalyse besteht die letzte Phase in der Konsolidierung der Anforderungs- und Testspezifikationen durch den Test-Designer und Systemarchitekten. Die Ergebnisse der vorherigen Phase ermöglichen dabei die Bewertung, ob alle relevanten Stakeholderanforderungen durch verifizierte Systemanforderungen adressiert werden. Ebenso wird deutlich, ob alle definierten Validierungsziele eingehalten werden.

Die Resultate der vierten Phase sind die geforderten Anforderungs- und Testspezifikationen, die automatisiert aus dem Systemmodell ausgeleitet werden.

5.3.5 Methode für die iterative Systemmodellierung

Das zuvor beschriebene Vorgehensmodell beinhaltet vier Phasen mit den zugehörigen Aktivitäten, sowie den Resultaten, die im Rahmen der Anforderungsanalyse und Testspezifikation im Automobilbereich benötigt werden (s. Kapitel 3.2 und 3.3). Da jedoch in der Praxis neben einem Phasen/Meilenstein orientierten Vorgehen auf der Makro-Ebene konkrete Aktivitäten als Teil eines Problemlösungsprozesses durchzuführen sind (vgl.

[ABK+16; VDI]), ist in diesem Kapitel eine Methode beschrieben, die Systemarchitekten und Test-Designer bei der iterativen Systemmodellierung unterstützt, um schließlich zu den jeweiligen Meilensteinen die notwendigen Ergebnisse entsprechend dem Vorgehensmodell aus dem Systemmodell ausleiten zu können.

Wie in Bild 5-9 dargestellt, ist die Methode auf Basis der Ontologie in die Blöcke *Problemraum*, *Validierungssystem* und *Produkt* strukturiert. Die Kreise I1, I2 und I3 deuten an,

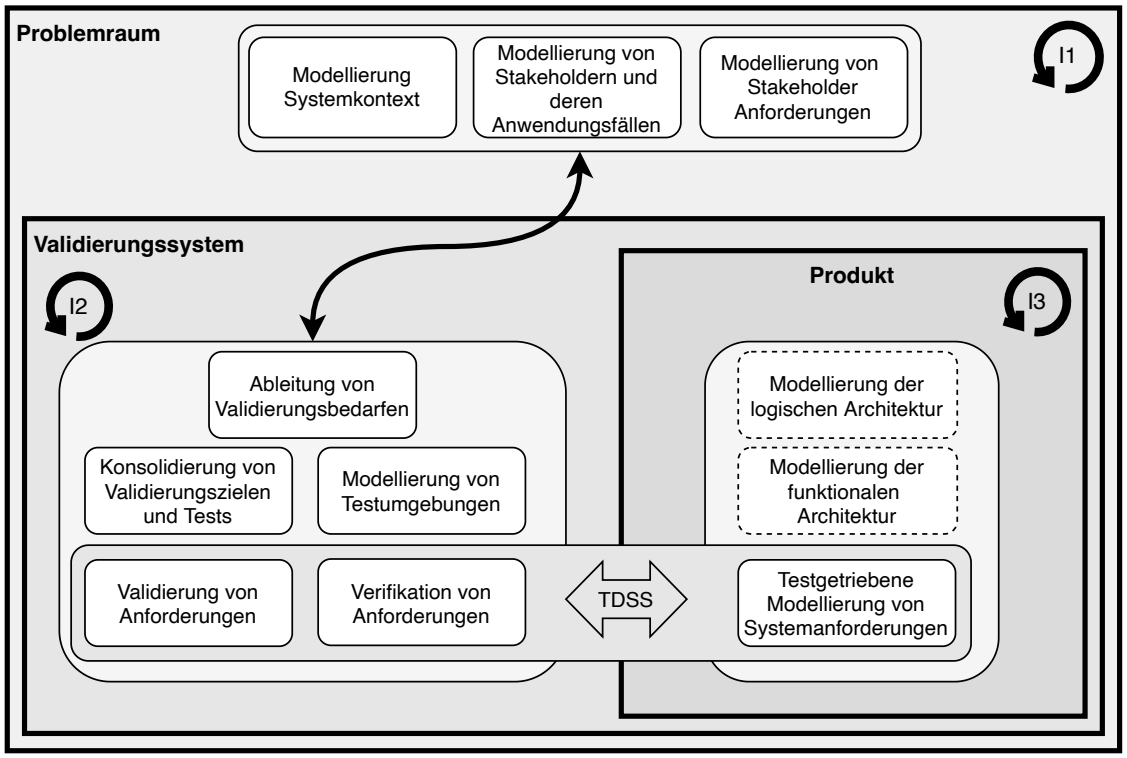


Bild 5-9: Vorgehensmodell für die integrative und iterative Analyse von Anforderungen und Spezifikation von Tests.

dass jeder Block entsprechend der aktuellen Entwicklungssituation und den verfügbaren Informationen im Entwicklungsprojekt separat angestoßen und durchlaufen werden kann. Der Aufbau der Methode ist dabei durch zwei Aspekte motiviert: Zum einen werden die Aktivitäten zur Testspezifikation und Anforderungsanalyse durch die Resultate der Modellierung innerhalb des Problemraums angestoßen, um einen starken Bezug zu den Stakeholderbedarfen herzustellen. Zum anderen basiert die Methode auf dem Ansatz der testgetriebenen Modellierung (vgl. Kapitel 4.4.1). Folglich werden die Informationen innerhalb des Problemraums (I1) dazu verwendet, Validierungsbedarfe zu definieren (I2), die wiederum die Basis zur testgetriebenen Modellierung und Analyse der Systemanforderung bilden (I3). Demnach wird die Modellierung des Validierungssystems vorangestellt und ermöglicht eine testgetriebene Modellierung der Systemanforderungen. Neben dem starken Bezug zu den Stakeholderbedarfen entsteht so eine enge Verzahnung von Produkt- und Validierungssystem.

Die einzelnen Modellierungsaktivitäten sind so definiert, dass diese ebenfalls entsprechend

der aktuellen Entwicklungssituation ausgewählt und durchgeführt werden können. Auf diese Weise wird ein stark iteratives und problemorientiertes Vorgehen unterstützt. Auf eine sequentielle Darstellung wurde hier verzichtet, um eine flexibleres Vorgehen zu ermöglichen und die jeweiligen Aktivitäten entsprechend dem zu lösenden Problem auswählen zu können. Falls bspw. für der Aktivität zur *Modellierung von Testumgebungen* innerhalb des Validierungssystems weitere Kontextinformationen notwendig sind, können diese zunächst in der Aktivität *Modellierung Systemkontext* innerhalb des Problemraums erfasst werden. Dabei entsteht durch die in der Ontologie definierten Beziehungen und durch das iterative Durchlaufen aller Aktivitäten ein konsistentes Systemmodell. Gleichzeitig wird durch die Zuordnung der zuvor definierten Sichten zu den jeweiligen Aktivitäten eine zielorientierte Modellierung unterstützt.

5.4 Integrative Anforderungsanalyse und Testspezifikation

Die integrative Anforderungsanalyse und Testspezifikation ist ein zentraler Bestandteil der Systematik. Der Ansatz basiert auf der zuvor beschriebenen Referenzarchitektur (Kapitel 5.2) und ist in das Vorgehensmodell (Kapitel 5.3) und in die Methode zur iterativen Systemmodellierung (Kapitel 5.3.5) integriert. Der Ansatz basiert auf der im folgenden beschriebenen TDSS-Methode. Wie zuvor beschrieben und in Bild 5-9 veranschaulicht, verbindet die TDSS-Methode die Produkt- und Validierungssystemmodellierung. Dafür sind die Ein- und Ausgangsdaten der Methode durch Ausführen der Aktivitäten innerhalb der Phase 3 des Vorgehensmodells (Kapitel 5.3.3) mit der Systemmodellierung verzahnt.

5.4.1 Methode (TDSS)

Ausgehend von der Validierungssystemmodellierung in Phase 2 des Vorgehensmodells, bzw. I2 in Bild 5-9, stellen die *Testfallspezifikation* und die *Testzielspezifikation* die Eingangsdaten für die TDSS Methode bereit. Um von den dort spezifizierten Testfällen und Testszenarien zu analysierten Systemanforderungen zu gelangen, sind die in Bild 5-10 dargestellten und im folgenden beschriebenen Schritte auszuführen.

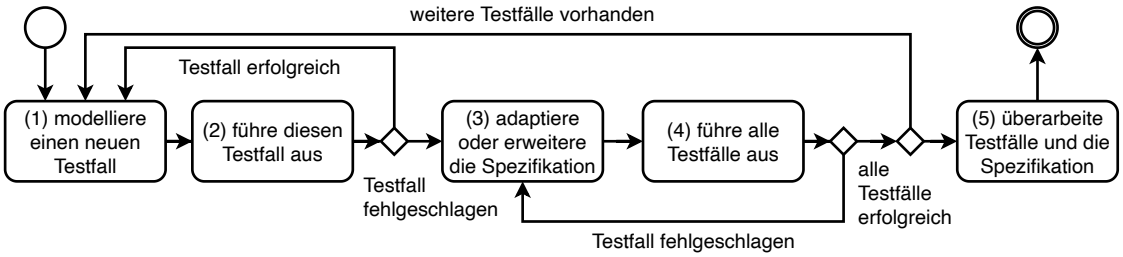


Bild 5-10: Methode zur testgetriebenen Anforderungsmodellierung (Test-Driven Scenario Specification - TDSS [WGK19])

Schritt 1: Modelliere einen neuen Testfall oder ein neues Testszenario: Ist die Aktivität *Verifikation von Anforderungen* der Ausgangspunkt, erfolgt in Schritt 1 die Modellierung

eines neues Testfalls aus der *Testfallspezifikation*. Ist die *Validierung von Anforderungen* der Ausgangspunkt, erfolgt hier die Modellierung eines neuen Testszenarios aus der *Testzielspezifikation*, das mehrere Testfälle miteinander kombiniert.

Schritt 2: Führe den Test aus: In Schritt 2 folgt die Ausführung des zuvor modellierten Testfalls oder Testszenarios. Dabei wird entsprechend dem TDD Paradigma (s. Kapitel 4.4.1) erwartet, dass die Ausführung fehlschlägt, da die erwarteten Systemanforderungen noch nicht modelliert wurden. Ist die Ausführung des Testfalls/Testszenarios in diesem Schritt erfolgreich, wurde die erwartete Systemeigenschaft bereits in vorangegangenen Iterationen umgesetzt, und es kann mit der Modellierung eines neues Testfalls/Testszenarios (Schritt 1) fortgefahren werden. Bei einem negativen Ergebnis ist Schritt 3 auszuführen.

Schritt 3: Adaptiere oder erweitere die Spezifikation: In diesem Schritt erfolgt die Modellierung der Systemanforderung über die Anwendung der SMLK Modellierungssprache.

Schritt 4: Führe alle Tests aus: Nachdem die Systemanforderungen in Schritt drei umgesetzt sind, folgt die Ausführung aller bisher modellierten Testfälle und Testszenarien. Auf diese Weise kann sichergestellt werden, dass jeder Testfall durch eine Systemanforderung adressiert wird und die modellierten Systemanforderungen die Validierungsziele erreichen. Da der Fokus hier nicht mehr wie zuvor in Schritt 2 auf einzelnen Testfällen/Testszenarien liegt, ermöglicht Schritt 4 die automatisierte Identifikation von Seiteneffekten zwischen Systemanforderungen. Werden bspw. widersprüchliche Systemanforderungen identifiziert, erfordert dies die Anpassung oder Erweiterung der Spezifikation in Schritt 3.

Schritt 5: Überarbeite Testfälle/Testszenarien und die Spezifikation: Falls keine weiteren Testfälle/Testszenarien zu modellieren sind, schließt Schritt 5 die TDSS Methode ab. Da in den Schritten 1 bis 4 der Fokus auf der erfolgreichen Durchführung des Tests liegt, ermöglicht Schritt 5 die Überarbeitung der Testfälle und der Systemanforderungen, um bspw. die Nachvollziehbarkeit des Modells zu erhöhen.

Durch die Anwendung dieser Methode werden der Systemarchitekt und Test-Designer bei der Modellierung der Systemanforderungen unterstützt. Die Ausführung der Testfälle und Testszenarien ermöglicht dabei eine automatisierte Analyse der Systemanforderungen. Da die Modellierung testgetrieben in kleinen Iterationen erfolgt, erhält der Anwender ein direktes Ergebnis der Testdurchführung, wodurch die Anwendbarkeit forciert wird. Testergebnisse können so unmittelbar auf unzureichende Systemanforderungen, fehlerhafte oder nicht mehr relevante Testfälle und Testszenarien zurückgeführt werden. Da die Ergebnisse der TDSS Methode gleichzeitig innerhalb des Systemmodells vernetzt sind, wird eine weitergehende Analyse bis zu den Elementen des Problemraums möglich. So kann bspw. ein Ergebnis der TDSS Methode auf ein fehlerhaftes Testszenario hindeuten, was wiederum mit den modellierten Anwendungsfällen- und szenarien und somit mit dem Standpunkt des Stakeholders (Phase 1 und I1 in Bild 5-9) abzugleichen ist.

5.4.2 Formalisierung und Ausführung von Systemanforderungen

Um die TDSS Methode anwenden zu können, wird für die Systematik die in Kapitel 4.2.4 eingeführte textuelle Modellierungssprache SMLK verwendet. Ausgehend von den zugrunde liegenden Konzepten zur szenariobasierten Modellierung (s. Kapitel 4.2), ermöglicht SMLK eine iterative Spezifikation von Anforderungen und ist somit für die Anwendung einer testgetriebenen Entwicklungsmethode geeignet.

Der Ausgangspunkt für die Modellierung und Ausführung der Systemanforderungen ist durch Bild 5-11 veranschaulicht. Auf der einen Seite sind einzelne Testfälle oder zu einem Testszenario kombinierte Testfälle zu modellieren, auf der anderen Seite erfolgt die testgetriebene Modellierung der Systemanforderungen. Sowohl die Testfälle als auch die Systemanforderungen werden dabei in jeweils einer SMLK *Szenariospezifikation* modelliert. Hierdurch wird die Ausführung der Testfälle und der Systemanforderungen über jeweils ein *Szenarioprogramm* möglich. Diese Szenarioprogramme sind über *Event-Knoten* miteinander verbunden und interagieren über den Austausch von *SMLK-Events*.

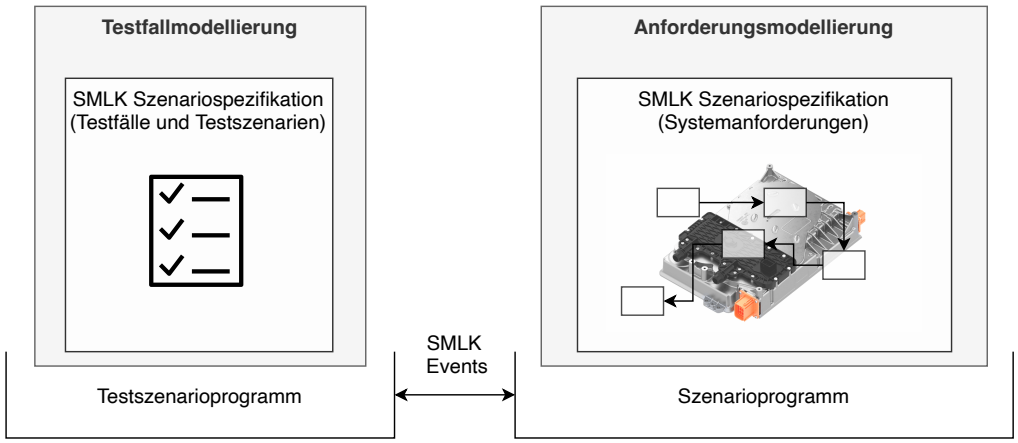


Bild 5-11: SMLK Szenariospezifikationen für die Testfall- und Anforderungsmodellierung.

Dem Blackbox-Testverfahren folgend (s. Kapitel 4.3.1), können auf diese Weise Testfälle spezifiziert werden, die die Ausführung des Anforderungsmodells über definierte Schnittstellen anstoßen und die resultierende Systemreaktion auswerten. Wie bereits in Bild 4-18 veranschaulicht, werden für den dynamischen Test funktionaler Systemeigenschaften die Testeingabedaten über PoCs bereitgestellt. Für die Definition dieser PoCs ist innerhalb des Szenarioprogramms festzulegen, welche externen Events das Systemverhalten beeinflussen können. Die Definition dieser Events ist am Beispiel der Funktion *Plug Interlock* in Listing 5.1 dargestellt. In dem dargestellten Fall ist der On-Board Charger (OBC) das zu entwickelnde System. Die Information, dass die Verriegelung eines Ladesteckers angestoßen werden soll, wird von einem zweiten Steuergerät (Application ECU) bereitgestellt. Für die testgetriebene Modellierung der Systemanforderungen des OBCs ist diese Interaktion über einen Testfall zu beschreiben, und das Event ist entsprechend über die Ausführung des Testszenarioprogramms zur Verfügung zu stellen. Somit ist, wie in Zeile 2 in Listing

5.1 dargestellt, das zugehörige Event in die Liste der externen Events hinzuzufügen.

```
1 scenarioProgram.addEnvironmentMessageType(
2 obc::lockRequest
3 )
```

Listing 5.1: Definition von externen Events innerhalb der Szenariospezifikation.

Der Testfall als Ausgangspunkt für die Modellierung der Systemanforderung ist in Listing 5.2 dargestellt. Über den Aufruf der Methode *runTest*(Scenarioprogramm) werden die zwei Szenarioprogramme miteinander verbunden, wobei das Szenarioprogramm zur Ausführung der Systemanforderungen als Parameter übergeben wird und der Inhalt des Testszenarioprogramms im Rumpf der Methode (Zeile 3 und 4) spezifiziert ist.

```
1 @Test
2 fun ‘Verriegelung des Ladesteckers‘() = runTest(Scenarioprogram){
3  request(applicationECU sends obc.lockRequest("active request"))
4  waitFor(chargingSocket sends obc.setStatus("locked"))
5 }
```

Listing 5.2: Testfall für die Verriegelung des Ladesteckers.

Bei der Ausführung des Testfalls wird in Zeile 3 zunächst das Event angefordert, das die oben beschriebene Interaktion zwischen der applicationECU und dem obc beschreibt. Wird dieses Event durch das Testszenarioprogramm für die Ausführung ausgewählt und schließlich ausgeführt, ist es innerhalb des verbundenen Szenarioprogramms sichtbar. Da dieses Event in Listing 5.1 in die Liste der relevanten externen Events aufgenommen wurde, kann es das Systemverhalten beeinflussen. In Listing 5.3 ist bspw. ein Szenario dargestellt, das dieses Event als *Trigger-Event* verwendet, wodurch die Ausführung des Szenarios durch das im Testszenarioprogramm ausgewählte Event angestoßen wird.

```
1 scenario(applicationECU sends obc.lockRequest("active request")){
2  request(obc.evaluateRequest())
3  request(obc sends chargingSocket.actuateMotor("close"))
4  request(chargingSocket sends obc.setStatus("locked"))
5 }
```

Listing 5.3: Spezifikation eines Szenarios zur Verrigelung des Ladesteckers

Im Rumpf des Szenarios ist das erwartete Systemverhalten beschrieben, was schließlich in Zeile 4 zu der Rückmeldung führt, dass der Ladestecker verriegelt wurde. Ebenso wie das Trigger-Event steht das in Zeile 4 angeforderte Event bei seiner Ausführung innerhalb des verbundenen Testszenarioprogramms zur Verfügung. Da in dem Testfall (Zeile 4 in Listing 5.2) auf dieses Event gewartet wird, ist so eine erfolgreiche Ausführung des Testfalls möglich.

5.4.3 Testfallgenerierung

Der zuvor beschriebene Ansatz zeigt, wie über die TDSS-Methode Systemanforderungen systematisch formalisiert und über die Ausführung der Modelle automatisiert analysiert werden können. Da die Modellierung der Anforderungen durch Testfälle getrieben wird, ist die Qualität der Testfälle von besonderer Bedeutung. Gleichzeitig ist der Aufwand für

die Testfallerstellung möglichst gering zu halten, um eine Anwendbarkeit der Systematik in realen Entwicklungsprojekten zu ermöglichen.

Vor diesem Hintergrund verwendet die Systematik die in Kapitel 4.3.3 beschriebene Technik zur Testfallgenerierung und kombiniert diese mit der TDSS-Methode und der Systemmodellierung. Somit kann eine optimale Menge an Testfällen aus einzelnen Anforderungen generiert werden (s. [FFV+23]). Diese können wiederum innerhalb der Validierungssystemmodellierung (Phase 2) durch Ausführung der Aktivität *Konsolidierung von Validierungszielen und Tests* mit Testszenarien und Validierungszielen vernetzt werden, um so strukturierte Eingangsdaten für die TDSS-Methode bereitzustellen.

Exemplarisch zeigt Bild 5-12, wie durch die Anwendung des Ansatzes nach FISCHBACH [Fis22] zunächst die kausalen Abhängigkeiten in der Anforderung identifiziert werden, um

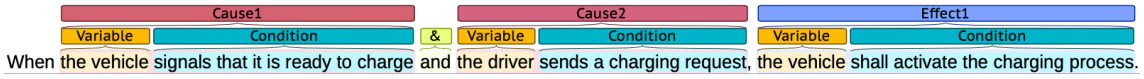


Bild 5-12: Anwendung der Testfallgenerierung als Basis für die Anwendung der TDSS.

diese in einem Zwischenschritt in einen CEG zu transformieren und schließlich die in Bild 5-13 dargestellten Testfälle zu generieren [2]. Somit kann der Aufwand für den manuellen

ID	Input		Expected Result	E
	the vehicle	the driver	the vehicle	
1	signals that it is ready to charge	sends a charging request	shall activate the charging process	
2	not signals that it is ready to charge	sends a charging request	not shall activate the charging process	
3	signals that it is ready to charge	not sends a charging request	not shall activate the charging process	

Bild 5-13: Anwendung der Testfallgenerierung als Basis für die Anwendung der TDSS-Methode.

Testentwurf reduziert werden. Die Überführung der generierten Testfälle in SMLK-Code ist in Listing 5.4 dargestellt.

```
1  @Test
2  fun ‘vehicle ready and driver request‘() = runTest(Scenarioprogram){
3   request(vehicle sends driver.readyToCharge())
4   request(driver sends vehicle.chargingRequest())
5   waitFor(obc.chargingStatus("active"))
6  }
7  @Test
8  fun ‘vehicle not ready  and driver request‘() = runTest(Scenarioprogram){
9   request(driver sends vehicle.chargingRequest())
10  waitFor(obc.chargingStatus("not active"))
11 }
12 @Test
13 fun ‘vehicle ready and no driver request‘() = runTest(Scenarioprogram){
14  request(vehicle sends driver.readyToCharge())
15  waitFor(obc.chargingStatus("not active"))
16 }
```

Listing 5.4: Modellierung der generierten Testfälle

[2] http://www.cira.bth.se, zuletzt abgerufen am 3.2.2023

5.4.4 Inter- und Intrasystemverhalten

Ein wichtiger Aspekt für eine umfassende Analyse von Anforderungen ist die Betrachtung des für die jeweilige Systementwicklung relevanten Kontextes. Da für die Realisierung von *Advanced Systems* zunehmend externe Systeme mit dem Fahrzeug interagieren, um eine integrierte Funktionalität bereitzustellen, sind diese interagierenden Systeme für die testgetriebene Anforderungsmodellierung zu berücksichtigen. Hierfür ist die Kontextmodellierung in Phase 1 des Vorgehensmodells (Kapitel 5.3) der Ausgangspunkt, wobei die Informationen über die Modellierung des Validierungssystems aufbereitet und über die zur Verfügung stehenden Sichten (Kapitel 5.2.3) für die Anforderungsanalyse verwendet werden können.

Der für die Anforderungsanalyse innerhalb eines ASE-Kontextes entwickelte Ansatz ist in Bild 5-14 anhand des zu Beginn eingeführten Beispiels (Kapitel 1.2) veranschaulicht. Der Aufbau entspricht hierbei der bereits eingeführten Unterteilung in eine Testfall- und

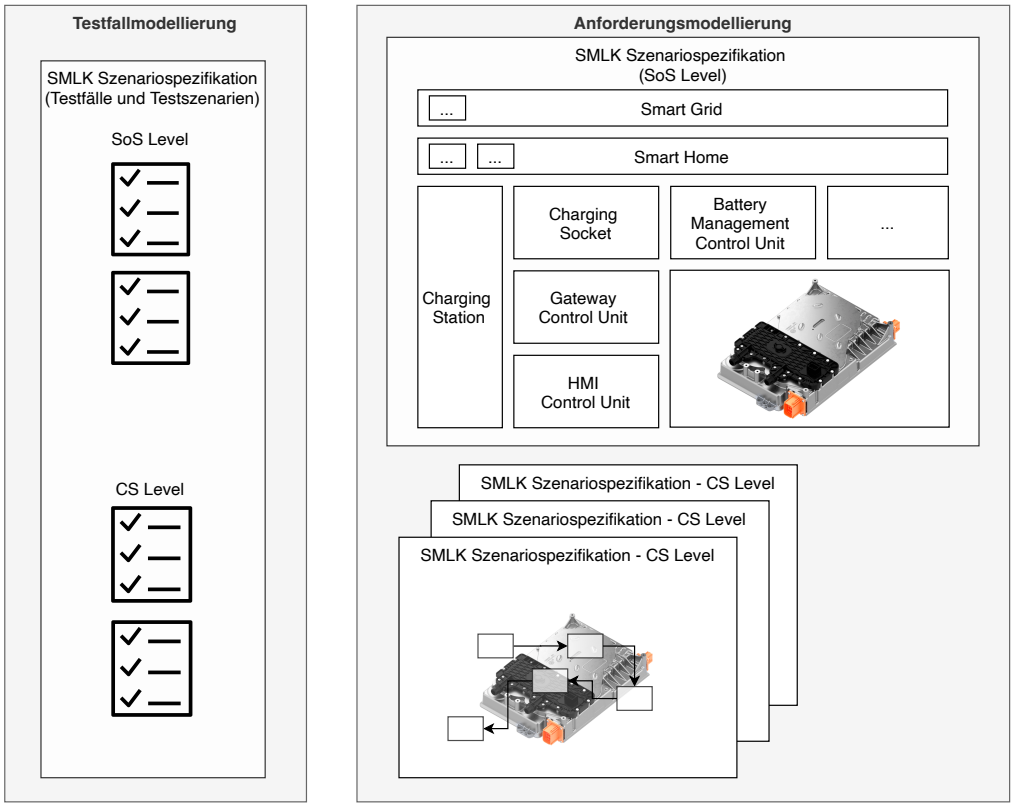


Bild 5-14: Ansatz für die testgetriebene Anforderungsmodellierung im ASE Kontext.

Anforderungsmodellierung. Hier wird jedoch zwischen Testfällen und Anforderungen auf SoS- und CS-Ebene unterschieden. Dabei liegt der Fokus bei der Modellierung auf der SoS-Ebene auf der Interaktion zwischen den Systemen ohne konkrete Details einzelner Systeme festzulegen. Die testgetriebene Ausführung der Systeminteraktionen auf SoS-Ebene unterstützt somit die Einordnung der in Phase 1 erfassten Anwendungsfälle, wodurch Verständnis über die Rolle des zu entwickelnden Systems innerhalb umfassenderen Kontextes aufgebaut werden kann. Auf der CS-Ebene erfolgt die Anforderungsmodellierung dem

oben beschriebenem Vorgehen. Die Testfälle treiben hier die Modellierung eines spezifischen Systems, wobei hier auch konkrete Daten für das jeweilige System berücksichtigt werden.

Ein Mehrwert für die Anforderungsanalyse entsteht hierbei durch die Kombination der beiden Ebenen. Einerseits können Systeminteraktionen abstrahiert beschrieben werden (SoS-Ebene) und andererseits ist eine detaillierte Anforderungsmodellierung des zu entwickelnden Systems möglich (CS-Ebene). Angelehnt an [HMM+21] werden zur Unterscheidung der beiden Ebenen innerhalb des Anforderungsmodells die Begriffe Inter- und Intrasystemverhalten verwendet. Die integrierte Ausführung des Inter- und Intrasystemverhaltens ermöglicht dabei die Identifikation von Widersprüchen in den Spezifikationen. So kann sichergestellt werden, dass bei einer kontinuierlichen Konkretisierung der Systemanforderungen des zu entwickelnden Systems die übergeordnete Funktionalität nicht beeinträchtigt wird. Weitergehend unterstützt der Ansatz die Bewertung des Systemverhaltens bei Kontextänderungen, indem vorhandene CS-Szenariospezifikationen mit angepassten SoS-Szenariospezifikationen ausgeführt werden.

Das methodische Vorgehen ist in Bild 5-15 dargestellt. Ausgehend von der der TDSS-Methode ist im oberen Bereich dargestellt, dass die Testfälle die Modellierung des Inter-Systemverhaltens treiben. Im unteren Bereich treiben Testfälle die Modellierung des Intra-Systemverhaltens. Beide Ebenen sind jedoch über die verwendete Modellierungssprache miteinander verzahnt, sodass die Ausführung und automatisierte Analyse des integrierten Systemverhaltens möglich wird.

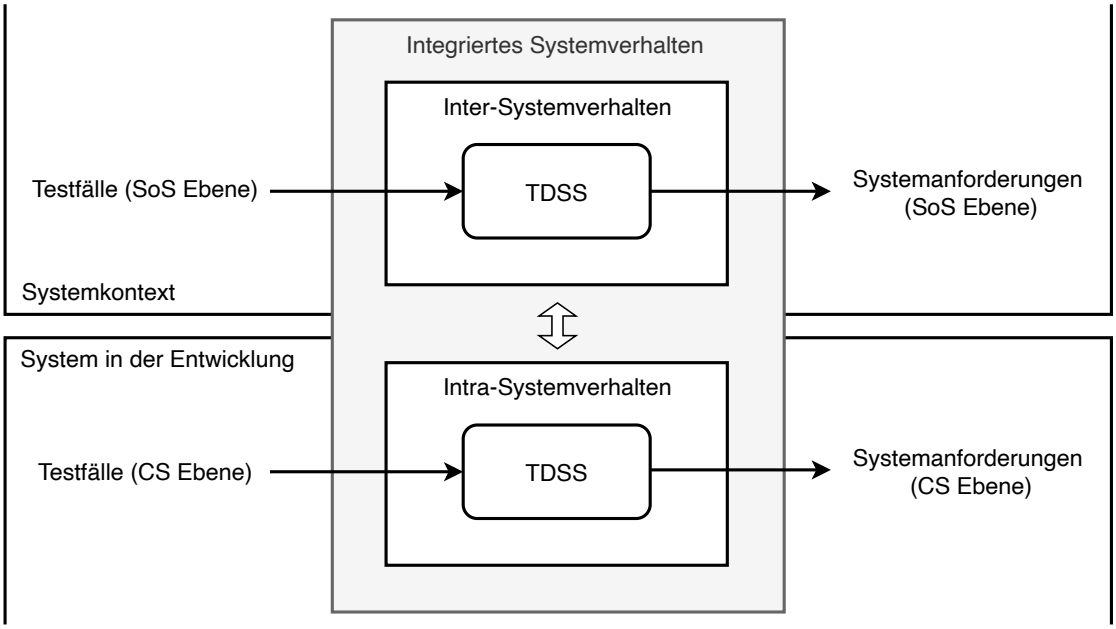


Bild 5-15: Methode für die testgetriebene Anforderungsmodellierung im ASE Kontext (vgl. [WGW+21])

5.4.5 Anwendungsbeispiel

Ausgehend von dem zu Beginn eingeführten Beispiel (Kapitel 1.2) zeigt Bild 5-16 eine konkrete Anwendung innerhalb eines Smart-Grid Kontextes (s. [WG21]). Im Bild ist die

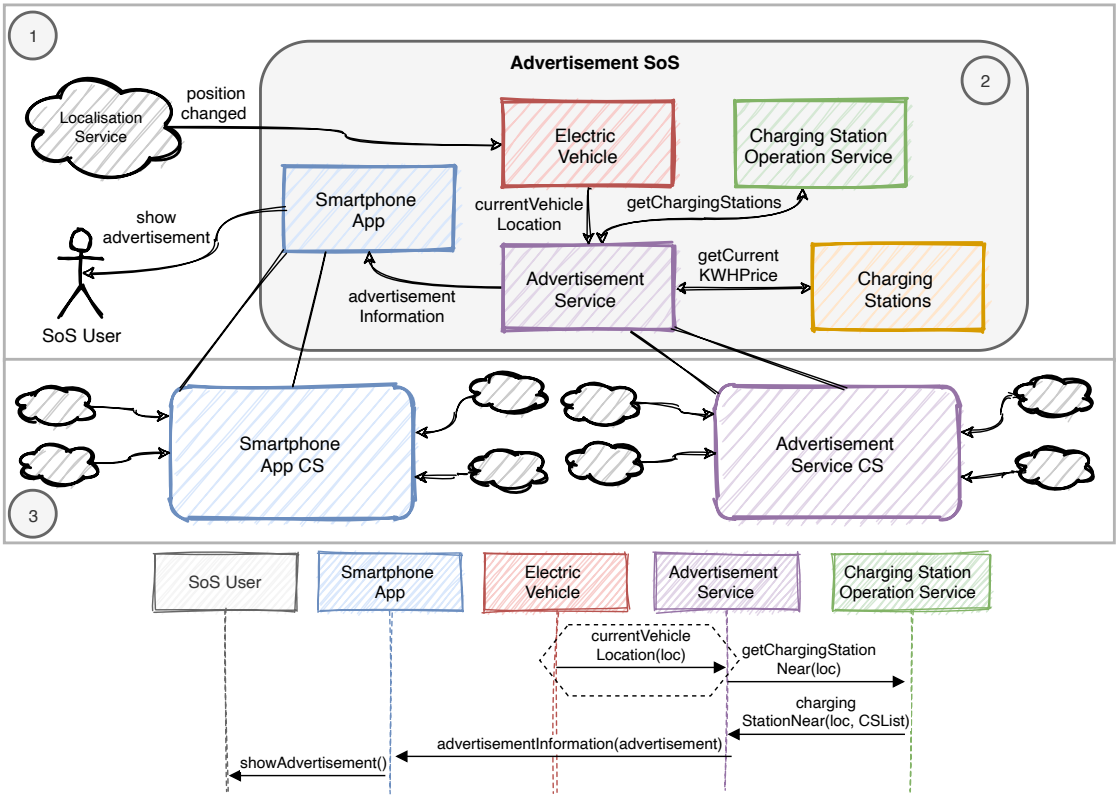


Bild 5-16: Anwendungsbeispiel für die Inter- und Intrasystemmodellierung in einem ASE-Kontext [WG21].

Interaktion zwischen mehreren Systemen abgebildet, die zusammen eine durch den *SoS-User* erlebbare Funktionalität realisieren. Das Ziel in diesem Beispiel ist die Realisierung einer Werbeanzeige, die dem *SoS-User* Informationen über Ladestationen in der aktuellen Umgebung eines Elektrofahrzeugs bereitstellt. Wie im Sequenzdiagramm in Bild 5-16 skizziert, ist der Ausgangspunkt des Szenarios eine Änderung der Fahrzeugposition. Wird eine Änderung erkannt, wird diese Information an einen *Advertisement Service* weitergegeben, der wiederum bei einem *Charging Station Operation Service* die in der Umgebung verfügbaren Ladestationen anfragt. Die verfügbaren Ladestationen werden dem *SoS-User* über eine *Smartphone App* angezeigt.

Der Methode in Bild 5-15 folgend ist das im Sequenzdiagramm skizzierte Inter-System-verhalten in Listing 5.5 formalisiert. Für die testgetriebene Formalisierung werden dabei unterschiedliche PoC verwendet. Zunächst wird das gesamte SoS als eine Blackbox betrachtet (① in Bild 5-16). In den Testfällen auf dieser Ebene wird erwartet, dass ein vollständiger Ablauf von der Änderung der Fahrzeugposition (Eingangsdaten am PoC) bis zur Anzeige der Ladestationen (Ausgangsdaten am PoO) spezifiziert ist. Hierzu ist in dem Szenario festzulegen, wie die einzelnen Systeme innerhalb des SoS Informationen

austauschen (② in Bild 5-16).

Das Szenario in Listing 5.5 beginnt mit der Definition eines *Trigger-Events*, was in diesem Fall die Änderung der Fahrzeugposition ist (Zeile 1). Durch das in Zeile 3 angeforderte Event wird die aktuelle Fahrzeugposition an den `advertisementService` übergeben. In Zeile 4 folgt die Anfrage der verfügbaren Ladestationen im Umfeld des Fahrzeugs durch den `advertisementService` an den `chargingStationOperationService`. Die Antwort des `chargingStationOperationService` an den `advertisementService` wird durch das Event in Zeile 9 modelliert, das als Parameter die Fahrzeugposition und die verfügbaren Ladestationen enthält. Ein wichtiger Aspekt an dieser Stelle ist, dass auf dieser Ebene lediglich relevant ist, welche Systeme beteiligt sind und welche Art der Information ausgetauscht wird. Somit werden in Zeile 6 und 7 prototypische Parameterwerte definiert, die bei einer weiteren Ausgestaltung der CSs durch konkrete Daten ersetzt werden. Damit die Ausführung des Intersystemverhaltens dennoch möglich ist und bei einer weiteren Ausgestaltung auf CS-Ebene möglich bleibt, wird das Event in Zeile 9 über das Schlüsselwort `requestParamValuesMightVary` angefordert, das die Verwendung von prototypischen und konkreten Parametern erlaubt.

```
1  scenario(localisationService sends electricVehicle receives ElectricVehicle::
       positionChanged){
2   val loc = it.parameters[0] as Location
3   request(electricVehicle sends advertisementService.currentVehicleLocation(loc))
4   request(advertisementService sends chargingStationOperationService.getChargingStationsNear
       (loc))
5
6   val dummyChargingStation1 = ChargingStation("EcoBigCharge")
7   val dummyChargingStation2 = ChargingStation("StarCharge")
8
9   val availableChargingStationsEvent = requestParamValuesMightVary(
       chargingStationOperationService
10  sends advertisementService.chargingStationsNear(loc, listOf(dummyChargingStation1,
       dummyChargingStation2)))
11
12  // To be refined: how does the ADService actually collect the price updates to forward to
       the user?
13
14  val advertisement = Advertisement(mapOf(dummyChargingStation1 to 295,
       dummyChargingStation2 to 289))
15  requestParamValuesMightVary(advertisementService sends smartphoneApp.
       advertismentInformation(advertisement))
16  request(smartphoneApp sends sosUser.showAdvertisement())
17 }
```

Listing 5.5: SoS Szenariospezifikation.

In gleicher Weise ist in Zeile 15 die Interaktion zwischen dem `advertisementService` und der `smartphoneApp` modelliert, wobei hier ein prototypisches `advertisement` Objekt als Parameter übergeben wird, über das den einzelnen Ladestationen Preisinformationen zugeordnet sind. Auf diese Weise ist es möglich eine vollständige Ende-zu-Ende Interaktion zu spezifizieren und testgetrieben auszuführen, Details einzelner CS werden auf dieser Ebene jedoch nicht spezifiziert. Es ist bspw. nicht beschrieben, wie der `advertisementService` die Preisinformationen erfasst, was auf der SoS-Ebene nicht relevant ist, aber für die Entwicklung notwendigen CS zu spezifizieren ist.

Unter der Annahme, dass der `advertisementService` ein zu entwickelndes System ist, das

in das SoS integriert werden muss, kann die testgetriebene Spezifikation dieses Systems in einer separaten Szenariospezifikation erfolgen, die unabhängig von der SoS Szenariospezifikation erstellt und ausgeführt werden kann. Eine mögliche CS Szenariospezifikation ist in Listing 5.6 dargestellt. In diesem Fall wird nicht das SoS als Blackbox betrachtet, sondern das zu entwickelnde CS (③ in Bild 5-16). Die vorherige Spezifikation des Intersystemverhaltens ist dabei der Ausgangspunkt für die Definition der PoCs und PoOs. Entsprechend ist bspw. die Änderung der Fahrzeugposition in der CS Szenariospezifikation als ein externes Event zu definieren, das über die Ausführung eines Testfalls die Ausführung eines Szenarios triggert (s. Zeile 1 in Listing 5.6).

```
1  scenario(electricVehicle sends AdvertisementService::currentVehicleLocation.symbolicEvent
       ()){
2   val loc = it.parameters[0] as Location
3   request(advertisementService sends chargingStationOperationService.
        getChargingStationsNear(loc))
4   val availableChargingStationsEvent = waitFor(chargingStationOperationService sends
        AdvertisementService::chargingStationsNear.symbolicEvent())
5   val availableChargingStations = availableChargingStationsEvent.parameters[1] as List<
        ChargingStation>
6   val chargingStationsCurrentPrice = mutableMapOf<ChargingStation, Int>()
7   scenario {
8    for(chargingStation in availableChargingStations){
9     request(advertisementService sends chargingStation.getCurrentKWHPrice())
10    val replyEvent = waitFor(chargingStation sends AdvertisementService::
          updateCurrentKWHPrice.symbolicEvent())
11    chargingStationsCurrentPrice.put(chargingStation, replyEvent.parameters[1] as Int)
12   }
13  } before (advertisementService sends smartphoneApp receives SmartphoneApp::
        advertismentInformation)
14  val advertisement = Advertisement(chargingStationsCurrentPrice)
15  request(advertisementService sends smartphoneApp.advertismentInformation(advertisement))
16 }
```

Listing 5.6: CS Szenariospezifikation.

Ein interessanter Aspekt in diesem Szenario ist Beschreibung, wie die Preisinformationen ermittelt wird. In Zeile 8-12 ist definiert, dass der aktuelle Preis bei jeder einzelnen Ladestation abgefragt wird. Die resultierende Informationen wird anschließend für die Erstellung des `advertisement` Objekts verwendet, das schließlich an die `smartphoneApp` gesendet wird. Im Gegensatz zu der Spezifikation auf SoS-Ebene werden innerhalb der CS Szenariospezifikation keine prototypischen Werte verwendet. Folglich wird die Ausführung von Events über `request` angefordert. Bei der integrierten Ausführung der SoS Szenariospezifikation und der CS Szenariospezifikation werden die prototypischen Werte mit den konkreten Werten der CS Szenariospezifikation überschrieben. Ein weiterer wichtiger Aspekt ist, wie die Szenarien auf den unterschiedlichen Ebenen miteinander synchronisiert werden können. Listing 5.6 beinhaltet hierzu bspw. das Konstrukt `scenario ... before<event>` (Zeile 7 und 13), wodurch festgelegt wird, dass zunächst die Preisinformationen zu ermitteln sind, bevor diese an die `smartphoneApp` gesendet werden.

5.5 Wiederverwendung von Lösungswissen

Wissen für die Analyse und Spezifikation von Anforderungen, sowie für den Entwurf und die Spezifikation von Tests ist in Unternehmen oft personengebunden (s. Kapitel 3.4.4).

Um das Wissen zu externalisieren und projektübergreifend wiederverwenden zu können, ist in diesem Kapitel ein lösungsmusterbasierter Ansatz beschrieben. Das Vorgehen basiert auf der Referenzarchitektur (Kapitel 5.2), der zuvor beschriebenen integrativen und formalen Anforderungsanalyse und Testspezifikation (Kapitel 5.4) und ist in das Vorgehensmodell (Kapitel 5.3) integriert.

5.5.1 Aufbau der Lösungsmuster

Zur Strukturierung sind die Lösungsmuster nach ALEXANDER durch die vier Kategorien *Name*, *Problem*, *Lösung* und *Kontext* beschrieben (vgl. Kapitel 4.5.1). Basis für die Beschreibung der Lösungsmuster sind die in Bild 5-17 dargestellten Beziehungen nach der in Kapitel 5.2.2 eingeführten Ontologie. Demnach ist das Problem als Teil eines

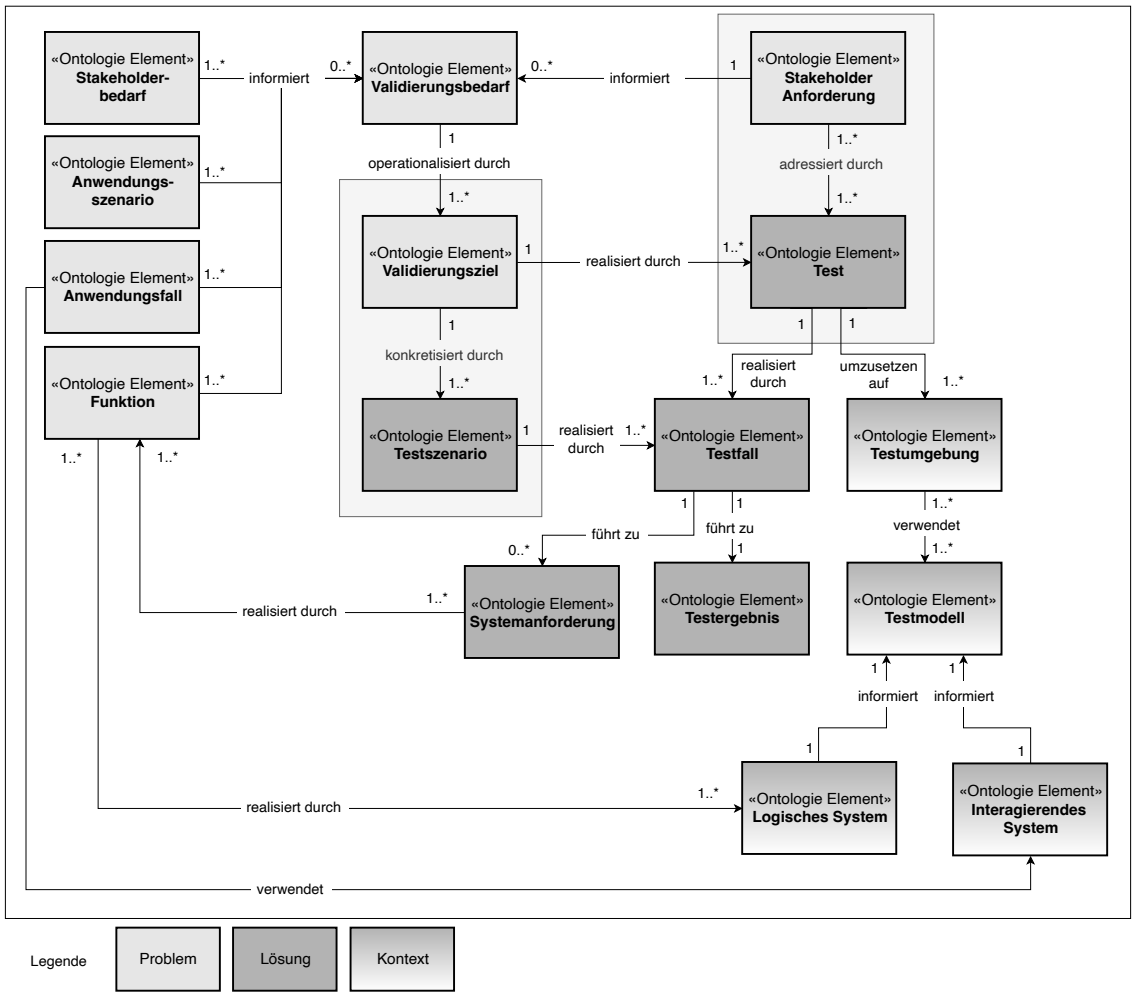


Bild 5-17: Ontologie-Elemente zur Beschreibung der Lösungsmuster

Lösungsmusters beschrieben durch Systemeigenschaften, die von Stakeholdern erwartet werden. Diese Erwartungen werden durch Stakeholderbedarfe, Anwendungsfälle, Anwendungsszenarien oder Funktionen definiert und entsprechend des Vorgehensmodells mit Validierungsbedarfen vernetzt. Hierdurch entsteht eine Übersicht aller Validierungsbedarfe, die entsprechend der jeweiligen Situation im Entwicklungsprojekt zur Definition von

konkreten Validierungszielen verwendet werden. Dabei wird über die Vernetzung von Validierungszielen zu den jeweiligen Validierungsbedarfen der Zusammenhang zwischen dem aktuellen Validierungsziel und den zugrunde liegenden erwarteten Systemeigenschaften deutlich.

Die gemeinschaftliche Aufgabe des Systemarchitekten und des Test-Designers besteht nun

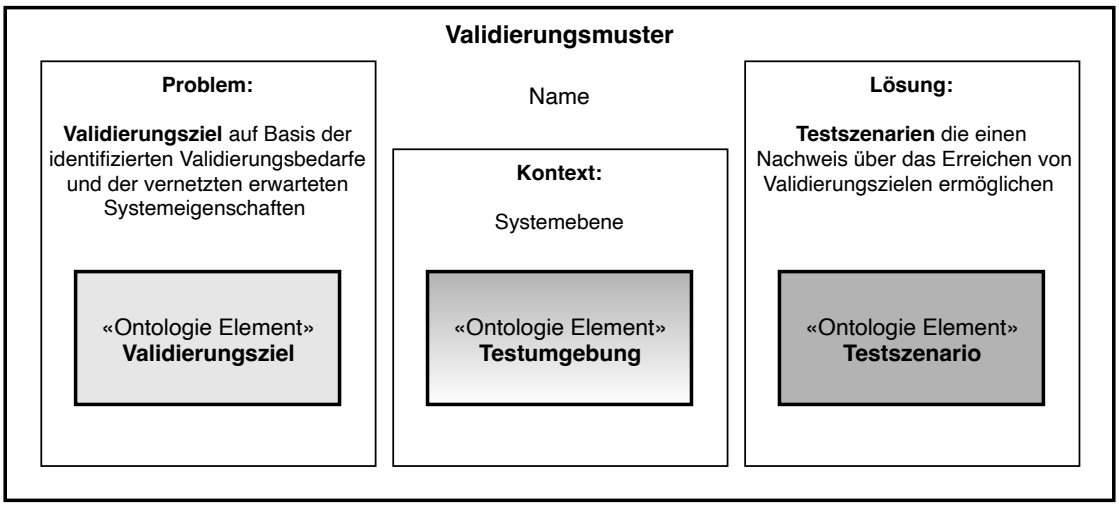


Bild 5-18: Grundlegende Ontologie-Elemente zur Beschreibung von Validierungsmustern.

in der Definition und der Modellierung, wie das **Validierungsziel auf Basis der erwarteten Systemeigenschaften** (Problem) über **Testszenarien** (Lösung) erreicht werden kann. So entsteht durch den Zusammenhang von Validierungsziel und Testszenario die Grundlage für die Beschreibung von **Validierungsmustern** in der in Bild 5-18 dargestellten Form.

Wie bereits in Bild 5-17 dargestellt, existiert neben dem Validierungspfad ein Verifika-

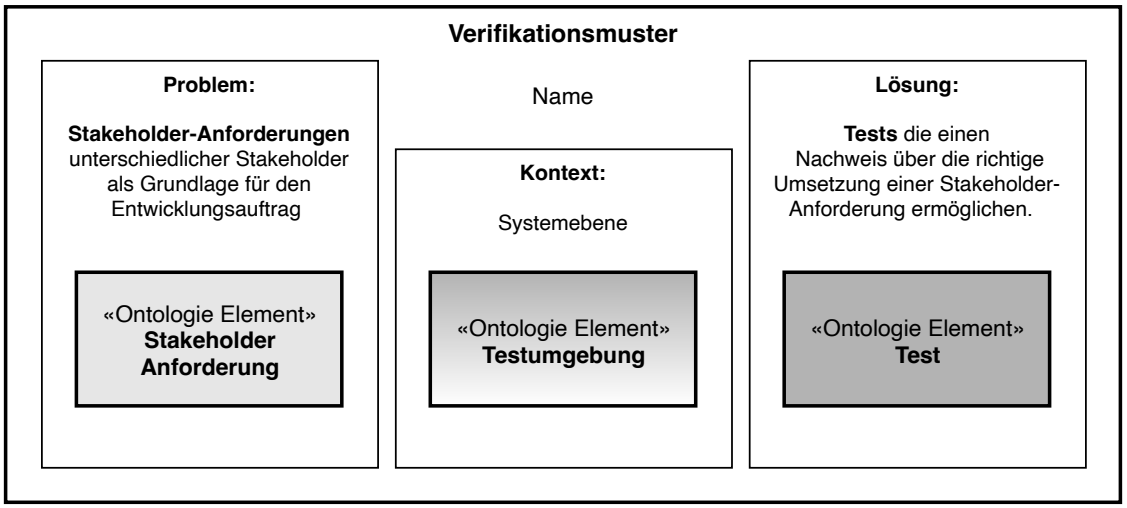


Bild 5-19: Grundlegende Ontologie-Elemente zur Beschreibung von Verifikationsmustern.

tionspfad. Hier besteht ein Zusammenhang zwischen den Stakeholder-Anforderungen

und den Tests, wobei durch den Test-Designer zu definieren ist, wie eine **Stakeholder-Anforderungen** (Problem) durch **Tests mit den vernetzten Elementen innerhalb des Validierungssystems** (Lösung) zu verifizieren ist. Diese Zusammenhänge ermöglichen die Beschreibung von **Verifikationsmustern** in der in Bild 5-19 dargestellten Form.

Neben den Problem-Lösungspaaren ist die Kontextbeschreibung ein wesentlicher Bestandteil für die Charakterisierung der Verifikations- und Validierungsmuster. Der **Kontext** wird hier beschrieben durch eine dem Test zugeordnete **Testumgebung**, die aus Testmodellen besteht, die wiederum aus dem Aufbau des logischen Produktsystems und den vernetzten interagierenden Systemen abgeleitet sind.

5.5.2 V&V Basismuster

Die zuvor beschriebenen Zusammenhänge sind die Grundlage für die Definition von V&V Basismustern nach den elementaren Mustern für die Entwicklung von Teststrategien nach SALADO & KANNAN (s. Kapitel 4.5.3). Im folgenden sind die Basismuster in Verifikations- und Validierungsmuster aufgeteilt. Dabei können die definierten Basismuster für die Verifikation und Validierung als Bausteine für die Erarbeitung von Verifikations- bzw. Validierungsstrategien angewendet werden (vgl. [SK19]).

Verifikationsmuster 1: Das Verifikationsmuster beschreibt den Fall, dass eine Stakeholder-Anforderung über *einen* Test verifiziert wird, wobei der Test durch einen Testfall oder durch mehrere Testfälle realisiert werden kann. Die Ausführung der mit dem Test verbundenen Testfälle auf einer Testumgebung führt zu Testergebnissen. Dabei sind die Testfälle mit Systemanforderungen vernetzt, da durch den testgetriebenen Entwicklungsansatz die formalisierte Systemanforderung als Teil des Anforderungsmodells in einer direkten Beziehung zum Testfall steht und somit Teil der Lösung ist. Auf diese Weise kann das Wissen darüber festgehalten werden, welche Systemanforderung/Systemanforderungen für die richtige Umsetzung der Stakeholder-Anforderung notwendig sind und welche Testfälle die erfolgreiche Verifikation ermöglichen.

Validierungsmuster 1: Entsprechend des vorherigen Verifikationsmusters beschreibt das Validierungsmuster, dass ein Validierungsziel durch *ein* Testszenario erreicht werden kann, das aus einem Testfall oder der Kombination von mehreren Testfällen besteht. Dabei ist der Kontext definiert durch die Summe der Testumgebungen, die über den Test dem Validierungsziel zugeordnet sind. Wie auch im Verifikationsmuster 1 sind in diesem Validierungsmuster über die Beziehung Testszenario, Testfall und formalisierte Systemanforderung die Systemanforderungen Teil der Lösung. Somit kann über die Anwendung des Validierungsmusters festgehalten werden, welche Systemanforderungen zur Erreichung des Validierungsziels notwendig sind.

Verifikationsmuster 2: Ebenso wie Verifikationsmuster 1 besteht bei diesem Muster eine eindeutige Beziehung zwischen der Stakeholder-Anforderung und einem Test. Hier steht der Test jedoch mit einem weiteren Test in Beziehung. Da der zweite Test keine Beziehung

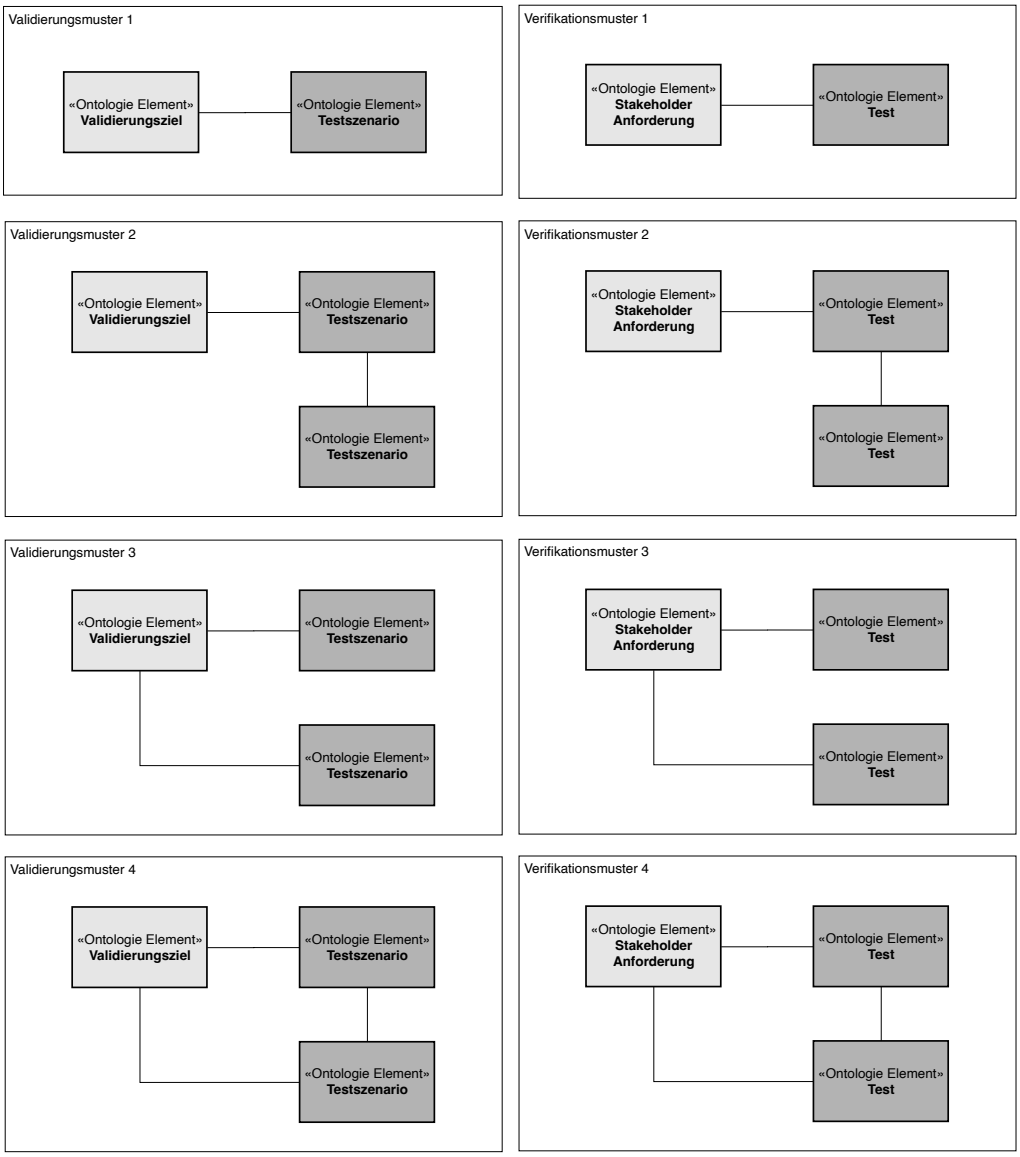


Bild 5-20: Übersicht der Basismuster auf Basis von [SK19]

zur Stakeholder-Anforderung besitzt, liefert dieser Test keine Informationen darüber, ob die Stakeholder-Anforderung richtig umgesetzt wurde. Die Beziehung zwischen den beiden Tests kann jedoch dazu verwendet werden, eine Eigenschaft der Testumgebung/Testumgebungen des ersten Tests zu ermitteln (bspw. Messgenauigkeit).

Validierungsmuster 2: Analog zum Verifikationsmuster 2 besteht eine eindeutige Beziehung zwischen dem Validierungsziel und einem Testszenario. Dieses Testszenario steht in Beziehung mit einem weiteren Testszenario, das jedoch keinen direkten Bezug zum ursprünglichen Validierungsziel besitzt. Diese Beziehung der beiden Testszenarien ermöglicht die Plausibilisierung der Ergebnisse der Testszenarien.

Verifikationsmuster 3: Im Unterschied zu Verifikationsmuster 1 werden für die Verifikation der Stakeholder-Anforderung mehrere Tests benötigt. Somit lässt sich die Stakeholder-Anforderung nur erfolgreich verifizieren, wenn die Testergebnisse aller mit den Tests

vernetzten Testfälle bekannt sind, wobei die Ausführung der Testfälle auf unterschiedlichen Testumgebungen erfolgen kann. In diesem Fall ist der Kontext definiert durch alle verwendeten Testumgebungen.

Validierungsmuster 3: Dieses Muster beschreibt die Beziehung von einem Validierungsziel zu mehreren Testszenarien. Somit ist der erfolgreiche Nachweis über das Erreichen eines Validierungsziels nur möglich, wenn alle Testszenarien ausgeführt werden und die Ergebnisse der vernetzten Testfälle vorliegen. Hier ist der Kontext definiert durch alle mit dem Validierungsziel in Beziehung stehenden Testumgebungen.

Verifikationsmuster 4: Das Verifikationsmuster 4 besteht ebenso wie das Verifikationsmuster 3 aus mehreren Tests zur Verifikation einer Stakeholder-Anforderung. Hier stehen die Tests jedoch miteinander in Beziehung. Für die erfolgreiche Verifikation der Stakeholder-Anforderung ist somit nicht nur die Umsetzung aller vernetzten Tests notwendig, sondern das Ergebnis des ersten Tests hat Einfluss auf die Interpretation des zweiten Tests.

Validierungsmuster 4: Analog zum Verifikationsmuster 4 besteht eine Beziehung zwischen einem Validierungsziel und mehreren Testszenarien, wobei die Ergebnisse der Testszenarien voneinander abhängig sind. Die Bewertung, ob das Validierungsziel erreicht wird, ist somit nur über die Bewertung aller Testergebnisse unter Berücksichtigung der Wechselwirkungen möglich.

5.5.3 Methodisches Vorgehen für die Externalisierung und Wiederverwendung von Lösungswissen

Die Aktivitäten *Konsolidierung von Validierungszielen und Tests* und *Modellierung von Testumgebungen* sind Teil des Vorgehensmodells für die iterative Modellierung des Validierungssystems (s. Kapitel 5.3.5). Die zuvor beschriebenen V&V Basismuster dienen zum einen zur Unterstützung der Modellierungstätigkeiten und zum anderen zur systematischen und kontinuierlichen Sicherung des Lösungswissens. Hierzu veranschaulicht Bild 5-21 die Anwendung der V&V Basismuster. Wie zuvor beschrieben, dienen die Basismuster als elementare Bausteine zur Entwicklung von V&V Strategien. Ausgehend von einem im Entwicklungsprojekt bestehenden Problem (Verifikation einer Stakeholder-Anforderung oder der Nachweis für das Erreichen von Validierungszielen), erfolgt zunächst eine Auswahl und Kombination der im Systemmodell hinterlegten Basismuster. Durch die Auswahl und Kombination der Basismuster entstehen V&V Strategiemuster, die ebenfalls im Systemmodell festgehalten und mit den Basismustern vernetzt werden. Die Beschreibung der Strategien adressiert dabei das konkrete projektspezifische Problem, beschreibt die jeweilige V&V Strategie jedoch auf einer logischen Ebene, d.h. mögliche Vorbedingungen, Testfallschritte und die erwarteten Ergebnisse werden abstrahiert festgehalten. Der letzte Schritt besteht in der projektspezifischen Anwendung der V&V Strategien. Hierbei wird die im Systemmodell festgehaltene V&V Strategie über den testgetriebenen Modellierungsansatz (Kapitel 5.4.2) in ein Anforderungsmodell überführt, und die Artefakte des Anforderungsmodells werden mit den Elementen des Systemmodells vernetzt. Auf diese Weise entsteht eine

durchgängige Traceability des ursprünglichen Basismusters über die erarbeitete Strategie bis hin zur Anwendung der Strategie. Der Zusammenhang von V&V Basismuster, V&V Strategie und Anwendung wird hier als **Digitaler Musterkatalog** bezeichnet. Dabei adressiert der Musterkatalog die Modellierung auf System- und Anforderungsebene, und durch die Vernetzung der Elemente des Systemmodells mit den formalisieren Testfällen, Testszenarien und Systemanforderungen wird die Externalisierung von Lösungswissen ermöglicht. Die Anwendung des Musterkatalogs verfolgt hierbei das Ziel, Systemarchitekten und Test-Designer bei Modellierung zu unterstützen, um schließlich vollständige und konsistente Spezifikationen aus dem Systemmodell auszuleiten. Diese Spezifikationen sind die Basis für die Realisierung des Produkts und der zugehörigen Validierungssysteme. Die Umsetzung dieser Systeme ist jedoch nicht Bestandteil des Musterkatalogs.

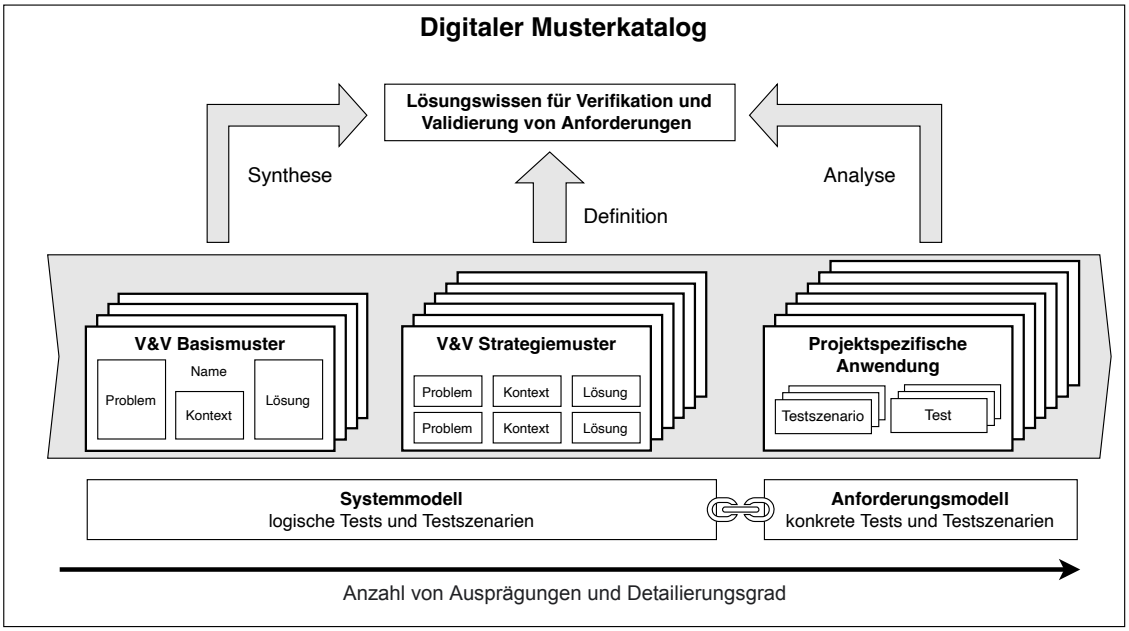


Bild 5-21: Digitaler Musterkatalog für die systematische Externalisierung und Wiederverwendung von Lösungswissen.

Wie zuvor beschrieben, ist die Anwendung des Musterkatalogs über die Aktivitäten *Konsolidierung von Validierungszielen und Tests* und *Modellierung von Testumgebungen* in das Vorgehensmodell integriert. Die Abhängigkeiten zwischen den Aktivitäten im übergeordneten Vorgehensmodell und den Aktivitäten zur Anwendung und Aufbereitung der Lösungsmuster sind in Bild 5-22 dargestellt, wobei die übergeordneten Aktivitäten grau hinterlegt sind. Demnach ist die Aktivität *Identifikation und Kombination von Basismustern* der erste Schritt zur Entwicklung von Strategiemustern auf Basis des aktuellen Problems und der aktuellen Situation innerhalb des Entwicklungsprojekts. Die Resultate dieser Aktivität zeigen, durch welche Basismuster das aktuelle Problem adressiert werden kann, womit diese Information die Basis für die Ausgestaltung und Konsolidierung von Validierungszielen und Tests ist. Gleichzeitig sind die Basismuster oder die Kombination von Basismustern Ausgangspunkt für die Modellierung von Testumgebungen. Somit adressiert der linke Pfad

die Definition von Problem-Lösungs-Paaren zur Unterstützung der Validierungsziel- und Testmodellierung, und der rechte Pfad adressiert die Kontextmodellierung.

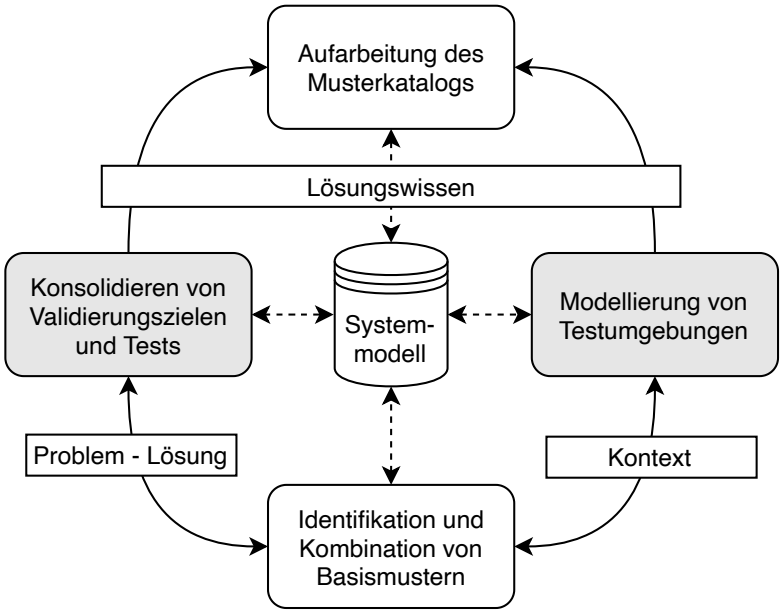


Bild 5-22: Methodisches Vorgehen für die Anwendung der Lösungsmuster für die Test- und Validierungszielmodellierung und zur Externalisierung von Lösungswissen über das zentrale Systemmodell.

Die Aktivität *Aufbereitung des Musterkatalogs* beinhaltet das Hinzufügen neuer Strategiemuster, oder das Anpassen bereits bestehender Strategiemuster. Ebenso erfolgt hier zum einen die Vernetzung der Strategiemuster mit den Basismustern und zum anderen die Vernetzung der Strategiemuster mit den projektspezifischen Anwendungen innerhalb des Anforderungsmodells. Über das Hinzufügen, Adaptieren und Vernetzen der Strategiemuster erfolgt somit die Sicherung des Lösungswissen innerhalb des zentralen Systemmodells.

5.5.4 Anwendungsbeispiel

Die Anwendung des digitalen Musterkatalogs ist über Bild 5-23 anhand der zu Beginn eingeführten Funktion *Timer-Charging* veranschaulicht (s. Kapitel 1.2). Dargestellt ist ein einfacher Zusammenhang eines Basismusters (Verifikationsmuster 4), das mit einer Verifikationsstrategie und einer projektspezifischen Anwendung vernetzt wurde. Ausgangspunkt ist hier eine Stakeholder-Anforderung der *Timer-Charging* Funktion, die festlegt, dass die vom Kunden vorgegebenen Daten vom OBC verarbeitet werden müssen und an eine Applikation zu übermitteln sind.

Bild 5-24 zeigt die Subaktivitäten für die Modellierung und Konsolidierung von Validierungszielen und Tests, wobei die Anwendung der im Musterkatalog beschriebenen Problem-Lösungspaare in der Subaktivität *Modellierung von Tests* (Verifikationsmuster) und *Modellierung von Validierungszielen* (Validierungsmuster) zum Einsatz kommen.

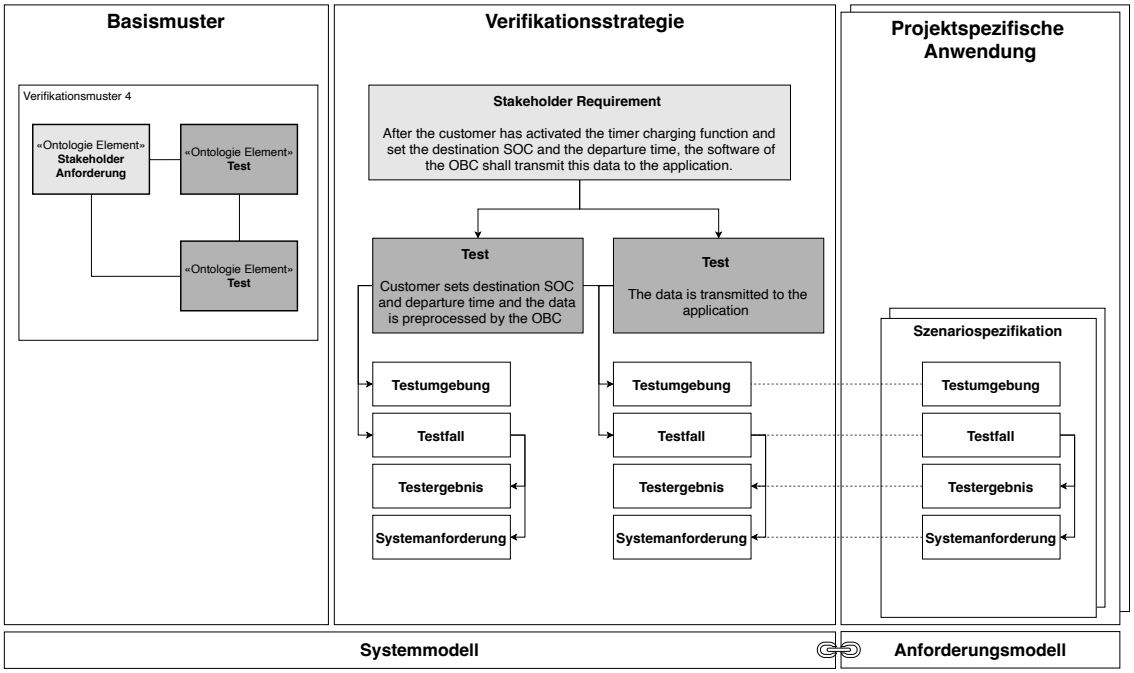


Bild 5-23: Beispiel für die Anwendung des Digitalen Musterkatalogs.

Da in diesem Beispiel eine Lösung für die Verifikation einer Stakeholder-Anforderung zu erarbeiten ist, sind die vernetzten Validierungsbedarfe zu analysieren und die zugehörigen Tests zu modellieren. Die wesentlichen Eingangsdaten sind hierfür die aus der Stakeholder-Anforderung generierten Testfälle (s. Kapitel 5.4.3), die über die zur Aktivität zugeordneten Sicht (s. Kapitel 5.2.3) dem Systemmodell hinzugefügt werden.

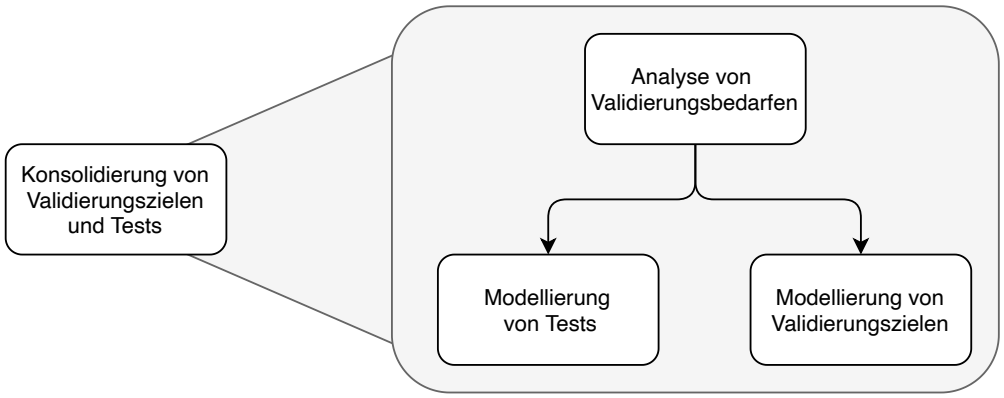


Bild 5-24: Anwendung der Strategiemuster zur Modellierung und Konsolidierung von Validierungszielen und Tests auf Basis der identifizierten Validierungsbedarfe.

Bild 5-25 zeigt die Subaktivitäten, in denen die in den Mustern enthaltenen Kontextinformationen zur Anwendung kommen. Dabei ist insbesondere das Expertenwissen über die zur Verfügung stehenden Testumgebungen und deren Eigenschaften von Bedeutung. Im dargestellten Beispiel sind für die vollständige Testdurchführung zwei Testumgebungen notwendig, woraus die dargestellte, dem Verifikationsmuster 4 entsprechende Auftei-

lung in die zwei voneinander abhängigen Tests resultiert. Ist das Lösungswissen für eine

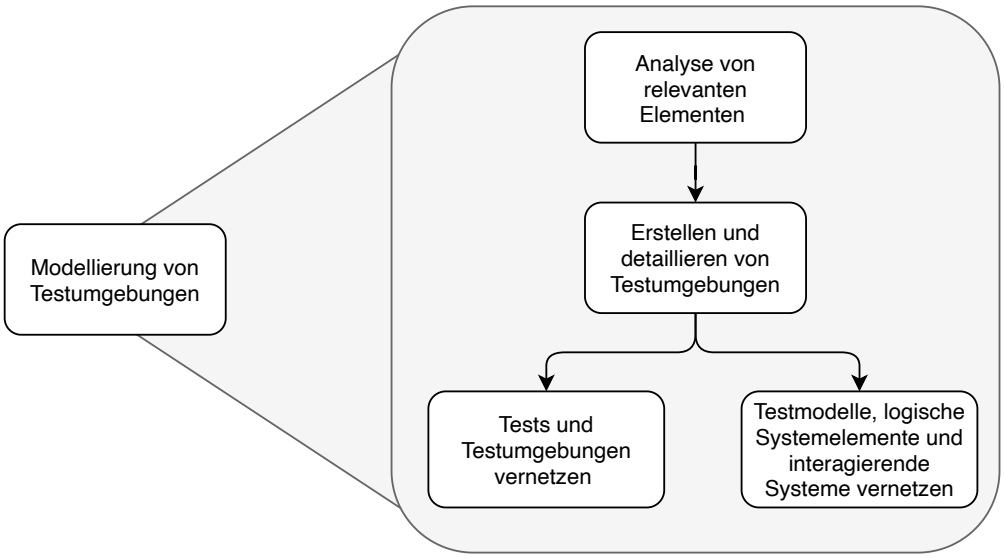


Bild 5-25: Anwendung der Kontextbeschreibung von Strategiemustern für die Modellierung von Testumgebungen.

geeignete Anforderungsverifikation innerhalb des Systemmodells erfasst, können diese Informationen mit dem Anforderungsmodell vernetzt werden, um über die testgetriebene Modellierung und Ausführung von Systemanforderungen eine weitergehende Analyse und Wissensaufbereitung zu unterstützen. Die Anforderungsmodellierung folgt dabei der in dem vorangegangenen Kapitel 5.4 beschriebenen Vorgehensweise.

5.6 Werkzeugunterstützung

Die Anwendung der Systematik wird durch die in diesem Kapitel beschriebenen Werkzeuge möglich. Die Beschreibung der Werkzeuge ist unterteilt in zwei Bereiche. Zum einen ist die Werkzeugunterstützung für die Systemmodellierung einschließlich der Aufbereitung und Sicherung von Lösungswissen dokumentiert (Kapitel 5.6.1)). Zum anderen ist ein integriertes Werkzeug für die Testfallgenerierung und automatisierte Anforderungsanalyse beschrieben (Kapitel 5.6.2)

5.6.1 Systemmodellierung

Das entwickelte MBSE Framework, bestehend aus der Ontologie (Kapitel 5.2.2) und den aufgabenspezifischen Sichten auf das Modell (Kapitel 5.2.3), wurde im Rahmen des MoSyS Projekts mit Hilfe von iQUAVIS[3] umgesetzt. Ausgehend von der Methode zur integrativen Produkt- und Validierungssystemmodellierung (s. Bild 5-9) zeigt Bild 5-26 den Einstiegspunkt für die Anwendung des Frameworks innerhalb von iQUAVIS. Auf

[3] https://www.two-pillars.de/iquavis/, zuletzt abgerufen am 3.11.2022

Basis des entwickelten Vorgehensmodells (Kapitel 5.3.3) sind die einzelnen Aktivitäten abgebildet. Ebenso ist hier die aus der Ontologie hervorgehende Aufteilung in Problemraum, Validierungssystem und Produkt dargestellt, um die Aktivitäten zu strukturieren. Dem

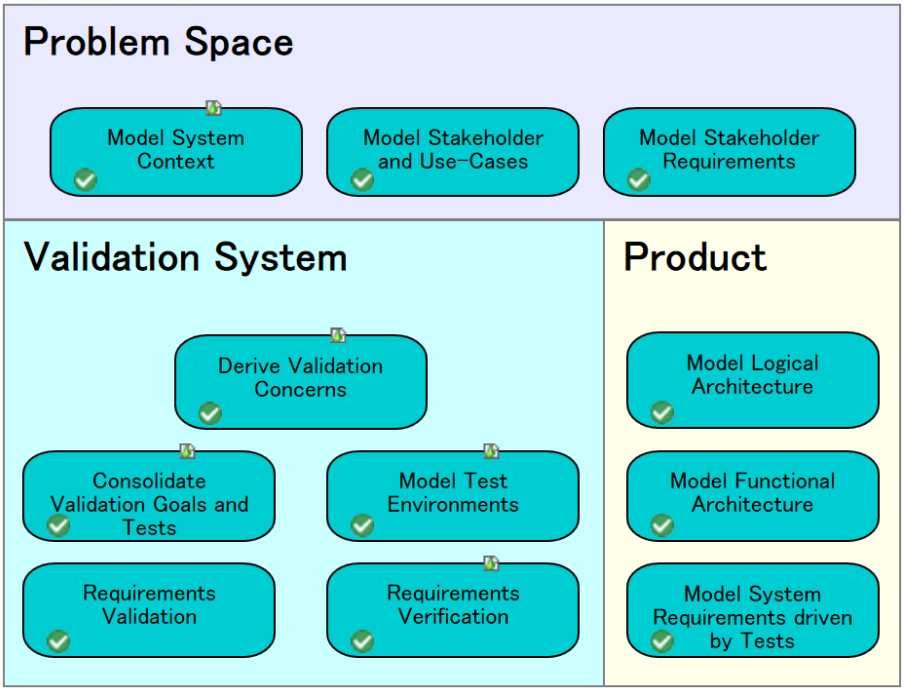


Bild 5-26: Aktivitäten zur Anwendung der Methode zur integrativen Anforderungsanalyse und Testspezifikation [WMG+24].

MBSE-Ansatz nach HOLT & PERRY [HP19] folgend umfasst das Systemmodell kohärente und konsistente Sichten. Motiviert durch den Ansatz nach MANDEL et al. [MBB+21] (vgl. Kapitel 4.1.1) wurden die beschriebenen Sichten (s. Kapitel 5.2.3) in iQUAVIS über geeignete Diagramme umgesetzt und den einzelnen Aktivitäten zugeordnet. Auf diese Weise kann durch den Systemarchitekten oder den Test-Designer die in der aktuellen Entwicklungssituation notwendige Aktivität ausgewählt und mit Hilfe des zugeordneten Diagramms durchgeführt werden. Die Anwendung ist als Teil der Evaluierung in Kapitel 6.3 beschrieben.

Innerhalb des Werkzeugs wurden überwiegend Baumdiagramme verwendet, um Systemelemente intuitiv hinzufügen und vernetzen zu können. Für das Ausleiten der resultierenden Test- und Anforderungsspezifikationen wurden Tabellen für eine übersichtliche Darstellung verwendet. Ebenso wird die Analyse von Systemelementen, die für die Validierung relevant sind, über eine tabellarische Darstellung ermöglicht. Die Kontextmodellierung sowie die Modellierung des logischen Aufbaus der Testumgebungen wird durch Blockdiagramme unterstützt.

Der Lösungsmusterkatalog kann ebenfalls über ein Baumdiagramm angewendet werden.

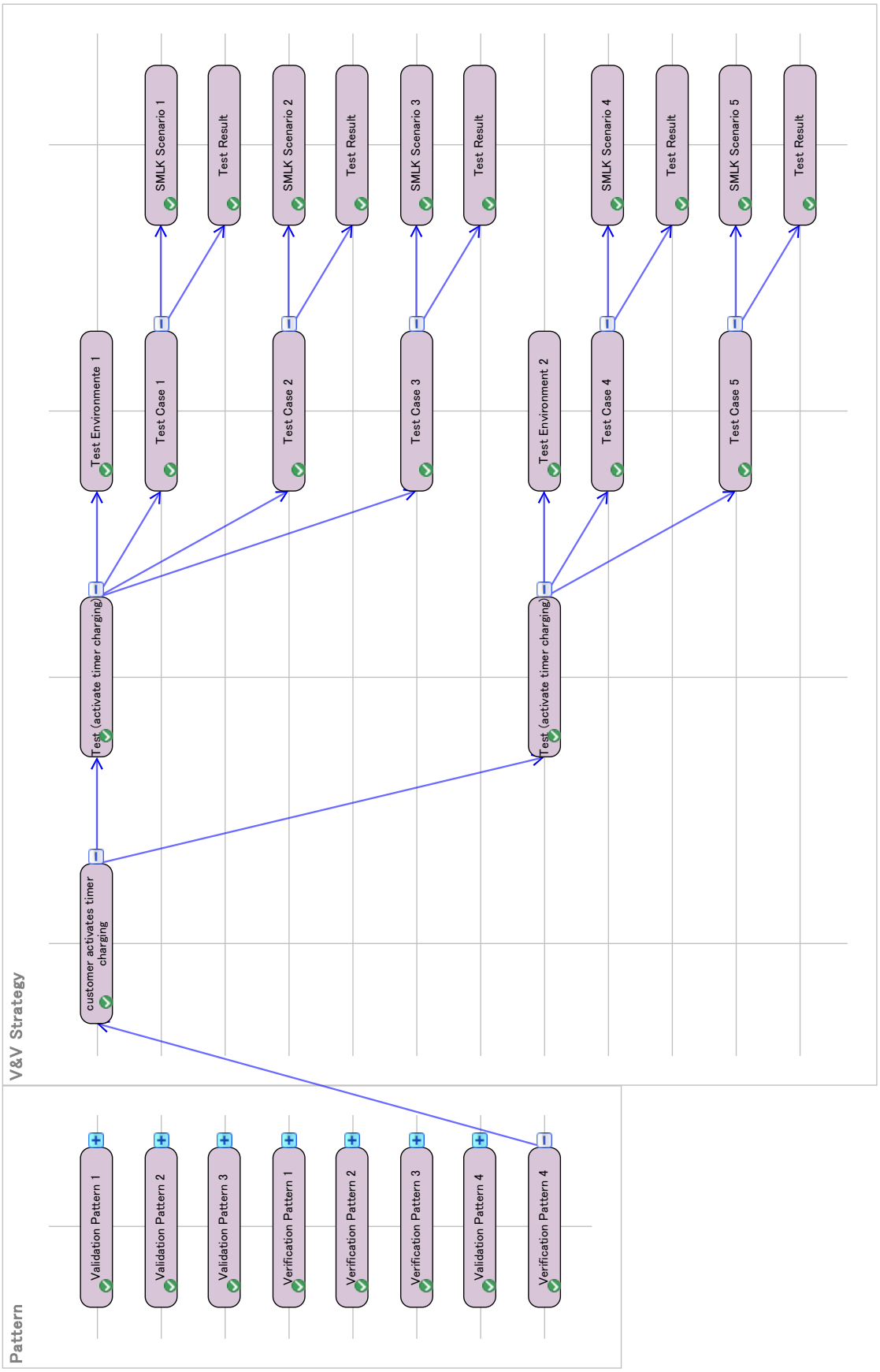


Bild 5-27: Implementierung des Musterkatalogs innerhalb des MBSE-Werkzeugs iQUAVIS.

Bild 5-27 zeigt die identifizierten Basismuster, die mit Strategiemustern vernetzt sind. Die mit den Strategiemustern vernetzten Elemente des Validierungssystems beschreiben die jeweilige Lösung für die entsprechende V&V Strategie. Hier wird ebenfalls deutlich, dass eine Traceability bis hin zu den mittels SMLK formalisierten Systemanforderungen besteht. Ebenso sind die bei Ausführung der TDSS-Methode entstehenden Testergebnisse Teil einer V&V Strategie. Dabei sind die im Systemmodell erfassten Testumgebungen die Grundlage für die Spezifikation der einzelnen Objekte innerhalb des Anforderungsmodells. Wie im vorherigen Kapitel in Bild 5-23 angedeutet, wird somit über die Systemelemente *Testumgebung*, *Testfall*, *Testergebnis* und *Systemanforderung* eine enge Verzahnung zwischen System- und Anforderungsmodell hergestellt. Die Umsetzung des Werkzeugs für die Anforderungsmodellierung ist im folgenden Kapitel beschrieben.

5.6.2 Anforderungsmodellierung

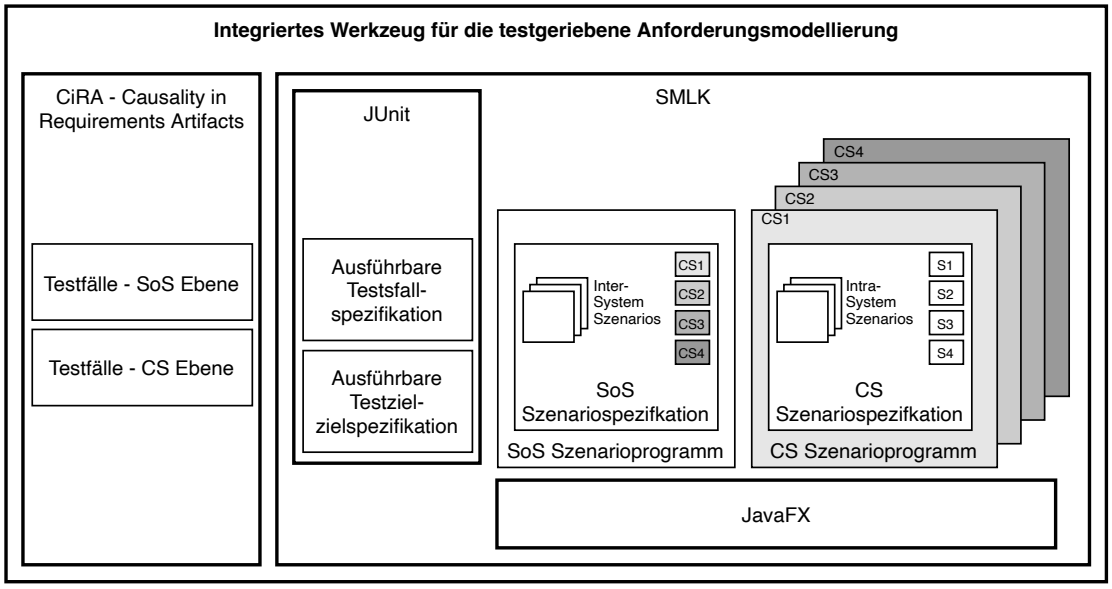


Bild 5-28: Integriertes Werkzeug für die testgetriebene, formale Modellierung, Simulation und Dokumentation von Systemanforderungen

Das Werkzeug für die Anforderungsmodellierung integriert das im Kontext dieser Arbeit entwickelte Werkzeug[4] [WG21] mit dem *Causality in Requirement Artifacts* (CiRA) Ansatz[5] für die automatisierte Generierung von Testfällen auf Basis von Anforderungen in unstrukturierter natürlicher Sprache [FFV+23]. Für eine weitergehende Unterstützung in der Anforderungsanalysephase ermöglicht das integrierte Werkzeug eine testgetriebene Simulation der Systemanforderungen und eine anschließende automatisierte Dokumen-

[4] https://bitbucket.org/crstnwchr/besos/ für die Modellierung auf SoS- und CS-Ebene, abgerufen am 1.3.2023

[5] http://www.cira.bth.se, abgerufen am 25.1.2023

tation in Form von UML Sequenzdiagrammen[6] [7]. Auf diese Weise können die Vorteile einer textuellen Anforderungsmodellierung und einer grafischen Darstellung ausgewählter Systemanforderungen miteinander kombiniert werden (s. [WFG+21; WMG+24]). Das für die Modellierung, Simulation und Dokumentation von Systemanforderungen verwendete Werkzeug ist von GREENYER beschrieben [Gre21] und ist über die *IntelliJ IDE*[8] anwendbar.

Der Aufbau des Werkzeugs für die Anforderungsmodellierung ist in Bild 5-28 skizziert. Auf der linken Seite ist die CiRA-Komponente dargestellt. Der Beschreibung in Kapitel 5.4.4 entsprechend können so Testfälle auf unterschiedlichen Abstraktionsebenen generiert werden. Zur Ausführung dieser Testfälle wird das JUnit[9] Framework verwendet, wobei das Framework in die SMLK Umgebung integriert ist. Somit ist es möglich, sowohl einzelne Testfälle der Testfallspezifikation, als auch umfangreiche Testszenarien der im Systemmodell definierten Testzielspezifikation zu modellieren und auszuführen.

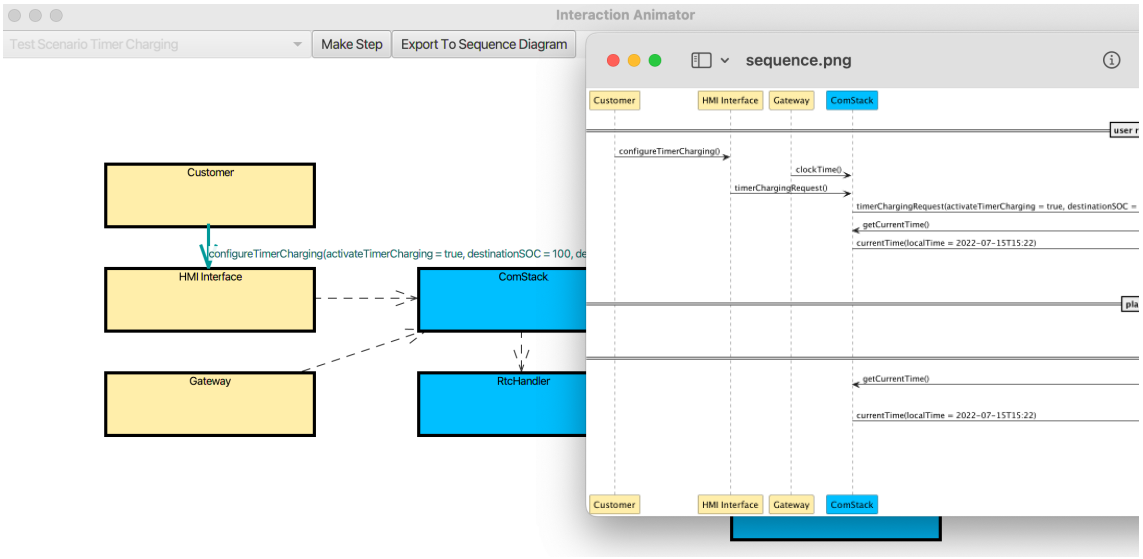


Bild 5-29: Visualisierung der modellierten Systeminteraktionen und Dokumentation der Systemanforderungen über UML Sequenzdiagramme.

Wie bereits in Kapitel 5.4.2 beschrieben, werden die Systemanforderungen mittels SMLK über Szenariospezifikationen formalisiert, die dann über Szenarioprogramme innerhalb der IntelliJ IDE ausführbar sind. Dabei wird für Analyse des integrierten Systemverhaltens eine separate Ausführung des Inter- und Intrasystemverhaltens unterstützt. Getrieben

[6] https://bitbucket.org/jgreenyer/smlk für die szenariobasierte textuelle Modellierung, abgerufen am 1.3.2023

[7] https://bitbucket.org/jgreenyer/smlk-animator/ für die Simulation und Dokumentation von Systemanforderungen, abgerufen am 1.3.2023

[8] https://www.jetbrains.com/de-de/idea/, abgerufen am 28.2.2023

[9] https://junit.org/, abgerufen am 1.3.2023

durch die Testfälle und Testszenarien ist es somit möglich, das Verhalten auf der SoS- und CS-Ebene unabhängig voneinander auszuführen und schrittweise zu konkretisieren, wobei gleichzeitig durch die Verbindungen der einzelnen Szenarioprogramme die automatisierte Analyse des integrierten Systemverhaltens möglich wird (s. [WG21]). Die Konsolidierung des modellierten Systemverhaltens wird über die Visualisierung unter Verwendung des JavaFX Frameworks[10] möglich. Die automatisierte Dokumentation des abgestimmten Systemverhaltens erfolgt über Sequenzdiagramme unter Verwendung von PlantUML[11]. Bild 5-29 zeigt exemplarisch eine Systeminteraktion auf der linken Seite und das resultierende Sequenzdiagramm auf der rechten Seite.

[10] https://openjfx.io, abgerufen am 1.3.2023

[11] https://plantuml.com/de/, abgerufen am 1.3.2023

6 Evaluierung der Systematik

Dieses Kapitel beschreibt die Evaluierung der zuvor beschriebenen Systematik. Grundlage für die Evaluierung ist die in Kapitel 2.2 beschriebene Vorgehensweise (s. Bild 2-1). Demnach wurden die veröffentlichten Zwischenergebnisse in drei DSR-Iterationen erarbeitet und evaluiert. Die erste Iteration adressiert die formale Anforderungsmodellierung und ist somit Teil des ersten Handlungsfelds (Spezifikation und Analyse von Anforderungen). Die zweite Iteration untersucht die integrative Anforderungsanalyse und Testspezifikation und schließt zusätzlich zum ersten Handlungsfeld das zweite Handlungsfeld (Entwurf und Spezifikation von Tests) mit ein. Die dritte Iteration umfasst die Systemmodellierung, liefert einen Beitrag zu den ersten beiden Handlungsfeldern und adressiert insbesondere das dritte Handlungsfeld (Externalisierung und Wiederverwendung von Lösungswissen). Die einzelnen der Ergebnisse sind im folgenden beschrieben.

6.1 Ansatz für die formale Anforderungsmodellierung (Iteration 1)

Die Ergebnisse der ersten Iteration wurden beim *Modeling in Automotive System and Software Engineering Workshop* (MASE19) im Rahmen der *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (MODELS'19) vorgestellt. Um die unten beschriebenen Ergebnisse besser nachvollziehbar und reproduzierbar zu machen, ist der entwickelte Programmcode[1] frei zugänglich.

[WGK19] WIECHER, C.; GREENYER, J.; KORTE, J.: Test-Driven Scenario Specification of Automotive Software Components. In: 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion, Munich, Germany, September 15-20, 2019, IEEE, 2019, S. 12–17

Wie in Bild 6-1 angedeutet, ist das **Problembewusstsein** der Ausgangspunkt der ersten DSR Iteration. Dieses Problembewusstsein ist über eine mehrjährige teilnehmende Beobachtung in der Steuergeräteentwicklung entstanden. Zusammen mit einer ersten Literaturanalyse führen diese Erkenntnisse zu den in Kapitel 3.2.4 beschriebenen Herausforderungen für die Spezifikation und Analyse von Anforderungen. Der erste **Lösungsvorschlag** basiert auf den identifizierten Ansätzen für die modellbasierte Spezifikation und Analyse von funktionalen Anforderungen (Kapitel 4.2). Das **Artefakt-Design** umfasst die Methode zur testgetriebenen Anforderungsmodellierung (TDSS) (Kapitel 5.4.1), die dafür verwendete Modellierungssprache (Kapitel 4.2.4) und des zugehörige Werkzeug (Kapitel 5.6.2). Zur **Demonstration** wurde zunächst ein Workshop innerhalb der Entwicklungsabteilung des beteiligten Unternehmens durchgeführt. Hierbei war das Ziel, Potentiale der modellba-

[1] https://bitbucket.org/jgreenyer/smlk/src/master/src/test/kotlin/org/scenariotools/smlk/examples/derating/Derating.kt

sierten Anforderungsanalyse zu identifizieren. Ergebnis des Workshops war die Auswahl einer Funktion mit zugehörigen Anforderungen, die im Rahmen eines Experiments mit dem entwickelten DSR Artefakt umgesetzt werden sollten. Um die Leistungsfähigkeit des Artefaktes zu bewerten wurden die Anforderungen zusätzlich und unabhängig voneinander mit den in der Entwicklungsabteilung etablierten Ansätzen umgesetzt. Die **Evaluierung** und die **Schlussfolgerung** sind im Folgenden beschrieben.

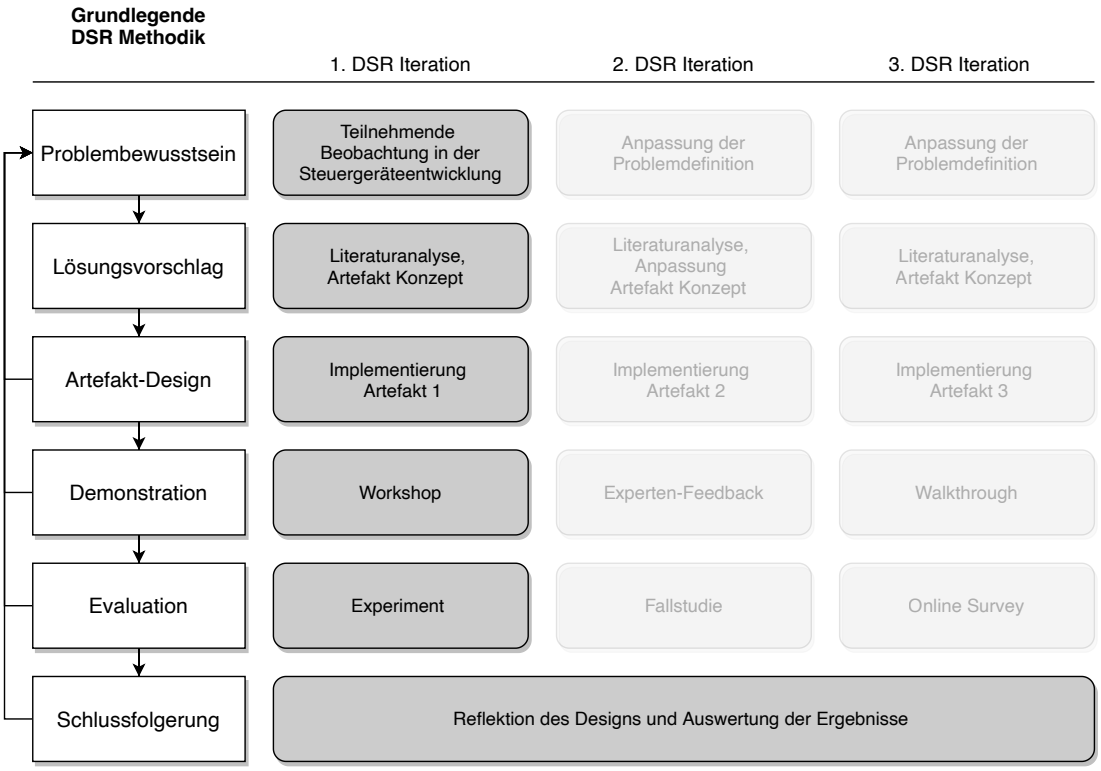


Bild 6-1: Vorgehensweise für die erste DSR Iteration.

6.1.1 Beschreibung der Funktion Derating

Die im Workshop ausgewählte Funktion war die *Regulierung der Ausgangsleistung* (Derating) des OBCs (s. Kapitel 1.2). Der OBC ist an das Hochspannungsnetz des Fahrzeugs sowie dessen Kühlungssystem angeschlossen. Bild 6-2 zeigt die hierfür notwendigen Hochspannungsanschlüsse auf der linken Seite und die Kühlmittelanschlüsse auf der rechten Seite. Die Funktion Derating überwacht die Temperatur des Kühlmittels und die Temperaturen der Leiterkarten innerhalb des Steuergeräts. In Abhängigkeit der gemessenen Temperaturen wird die Ausgangsleistung reduziert, um eine Überhitzung des Steuergeräts zu verhindern.

Für das Experiment wurden innerhalb des Workshops die in Tabelle 6-1 dargestellten Anforderungen der Funktion Derating ausgewählt. Diese Anforderungen beschreiben die grundlegende Derating Funktionalität: Wird durch die Temperatursensoren eine Temperatur des Kühlmittels im Bereich von $-40°C$ und $65°C$ gemessen, soll die Ausgangsleistung

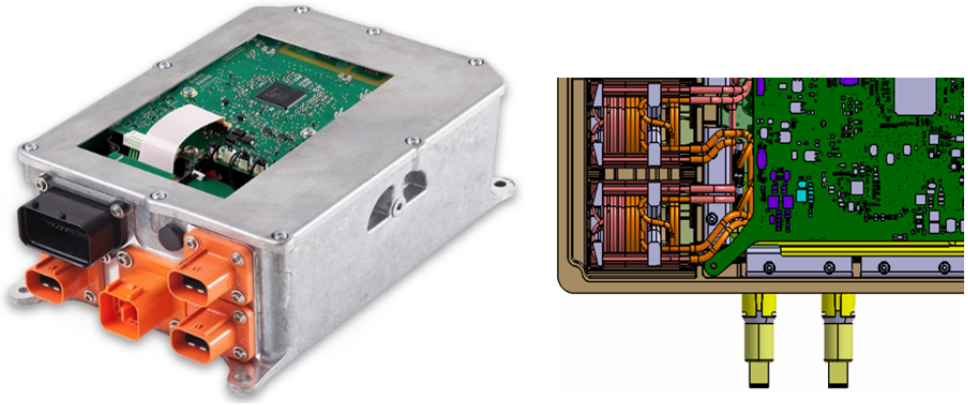


Bild 6-2: OBC der die Funktion Derating *umsetzt. Im linken Teil sind die Ein- und Ausgänge zur Energietransformation dargestellt. Im rechten Teil sind die Kühlmittelanschlüsse zu sehen.*

ID	**Anforderung**
Req1	No temperature related derating of the available output power shall be commanded, if the coolant inlet temperature sensor reads values between $-40°C$ and $65°C$.
Req2	If the coolant inlet temperature sensor reads values between $65°C$ and $75°C$, linear derating with $1/10$ of maximal output power per $1°C$ shall be commanded.
Req3	Power-down above $75°C$ coolant inlet temperature.
Req4	Power-down if coolant inlet temperature increases more than $5°C$ within $5s$.
Req5	Power-down if PCB temperature increases more than $20°C$ within $3s$.

Tabelle 6-1: Anforderungen der Funktion Derating [WGK19].

nicht reduziert werden (*Req1*). Liegt die Kühlmitteltemperatur in einem Bereich zwischen $65°C$ und $75°C$, soll die Ausgangsleistung linear um $1/10$ der maximalen Ausgangsleistung reduziert werden (*Req2*). Bei einer Kühlmitteltemperatur von über $75°C$ soll die Energietransformation abgeschaltet werden (*Req3*). Neben der Messung der Kühlmitteltemperatur soll auch der zeitliche Anstieg der Temperatur berücksichtigt werden. Hier ist zum einen festgelegt, dass eine Abschaltung erfolgen soll, wenn die Kühlmitteltemperatur in einem Zeitraum von $5s$ um $5°C$ ansteigt (*Req4*). Zum anderen soll eine Abschaltung erfolgen, wenn die Leiterkartentemperatur in einem Zeitraum von $3s$ um $20°C$ ansteigt (*Req5*).

6.1.2 Umsetzung nach dem etablierten Entwicklungsprozess

Der etablierte Prozess innerhalb des Anwendungsunternehmens basiert auf den Vorgaben des ASPICE® Standards (vgl. Kapitel 3.1.2). Die Analyse der Anforderung erfolgt im Anwendungsunternehmen manuell, d.h. die Anforderungen werden einzeln bewertet, und die Ergebnisse werden in Reviews dokumentiert. Nachgelagert erfolgt die Umsetzung der Anforderungen über Implementierungsmodelle, die zur Codegenerierung verwendet werden. Im Anwendungsunternehmen wurde hierfür das Werkzeug Simulink®[2] verwendet. Das Ergebnis der Umsetzung wurde im Rahmen von Model-in-the-Loop (MIL) Tests verifiziert. Zur Implementierung der Testfälle wurde das Werkzeug MES Test Manager®[3] verwendet. Dieses Werkzeug nutzt die proprietäre Sprache MARS (MTest Assessable Requirements Syntax) zur Spezifikation von funktionalen Anforderungen, um auf Basis der so spezifizierten Anforderungen eine automatisierte Verifikation der Simulink-Modelle durchzuführen. Die in Tabelle 6-1 dargestellten Anforderungen waren demnach zunächst manuell in das vorgegebene Format zu überführen, um daraus automatisiert die Testfälle für den MIL Test abzuleiten und ausführen. Die für die Verifikation angepasste Anforderung ist am Beispiel von *Req2* in Listing 6.1 dargestellt.

```
1  WHILE signal CoolantTemperature is greater than 65 and signal CoolantTemperature is less
       than 75  THE signal DeratingFactor SHALL be equal to parameter DeratingOffset +
       parameter DeratingSlope * signal CoolantTemperature
```

Listing 6.1: Anforderung für die automatisierte Verifikation des Simulink-Modells am Beispiel von Req2

Das Ergebnis der Durchführung ist in Bild 6-3 dargestellt. Obwohl die vorgelagerten

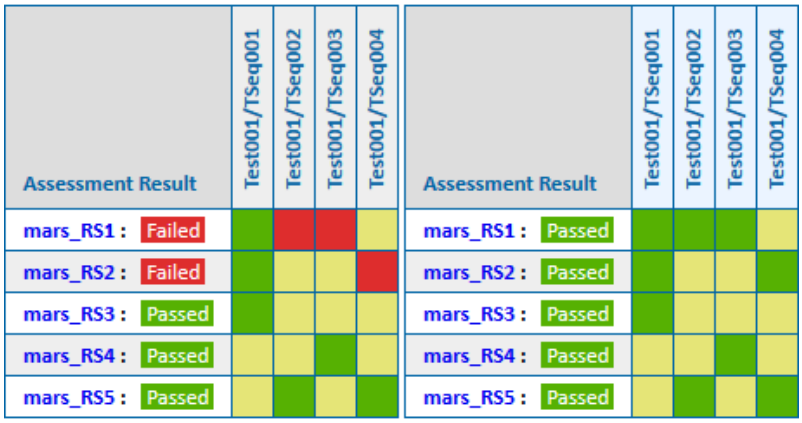


Bild 6-3: Ergebnisse des MIL-Tests: Auf der linken ist zu sehen, das die Verifikation von Req1 und Req2 nicht erfolgreich war. In einem zweiten Testlauf wurden die Anforderungen und das Modell angepasst, was zu den positiven Testergebnissen auf der rechten Seite führte.

Schritte insbesondere die Anforderungsanalyse und die Implementierung des Modells ohne

[2] https://de.mathworks.com, zuletzt abgerufen am 18.8.2022

[3] https://model-engineers.com/en/quality-tools/mtest/ zuletzt abgerufen am 19.8.2022

Auffälligkeiten durchgeführt wurden, zeigt der MIL-Test, dass *Req1* und *Req2* fehlerhaft sind. Die Analyse der Testergebnisse hat gezeigt, dass ein Widerspruch zwischen den Anforderungen *Req1* bzw. *Req2* und der Anforderung *Req4* besteht. Einerseits ist hier beschrieben, dass die Ausgangsleistung nicht (*Req1*) oder teilweise (*Req2*) reduziert werden soll. Andererseits legt *Req4* fest, dass bei einem Temperaturanstieg von $5°C$ innerhalb von $5s$ die Energiewandlung abgeschaltet werden soll. Da dieser Temperaturanstieg auch in den durch *Req1* und *Req2* definierten Temperaturbereichen auftreten kann, entsteht der Konflikt zwischen den Anforderungen.

6.1.3 Anwendung der TDSS Methode in der Steuergeräteentwicklung

Ebenso wie die zuvor beschriebene Umsetzung nach dem etablierten Prozess basiert die Anwendung der TDSS Methode auf den Anforderungen in Tabelle 6-1. Im gleichen Zeitraum und unabhängig voneinander wurden diese Anforderungen mit der TDSS Methode (s. Bild 5-10) umgesetzt, was in den folgenden Schritten beschrieben ist:

Schritt 1: schreibe einen neuen Testfall: Auf Basis der in den Anforderungen enthaltenen Informationen konnten Testfälle manuell abgeleitet werden. Für *Req2* wurde bswp. der Fall berücksichtigt, in dem die Temperatur des Kühlmittels $70°C$ beträgt. In diesem Fall wird erwartet, dass die Ausgangsleistung um 50% reduziert wird, was einem Deratingfaktor von 0.5 entspricht. Die Umsetzung des Testfalls mit Hilfe von SMLK und dem JUnit Framewok hat die in Listing 6.2 dargestellte Form. In Zeile 5 wird der Wert der Kühlmitteltemperatur auf 70 gesetzt. Folglich wird in Zeile 7 erwartet, dass durch den OBC der DeratingFaktor auf 0.5 gesetzt wird. Entsprechend wird in Zeile 7 auf das Event `setDeratingFactor(0.5)` gewartet. In Zeile 4 ist über das Hinzufügen von `forbiddenEvents` festgelegt, dass nur der in Zeile 7 erwartete Deratingfaktor von 0.5 auftreten darf.

```
1  @Test
2  fun `At 70 degree the DeratingFactor is 0-5`()
3  = runTest(deratingScenarioProgram){
4   forbiddenEvents.add(deratingComponent receives DeratingComponent::setDeratingFactor)
5   request(deratingComponent.setCoolantTemperature(70))
6   request(deratingComponent.startCycle())
7   waitFor(deratingComponent.setDeratingFactor(0.5))
8   waitFor(deratingComponent.endCycle())
9  }
```

Listing 6.2: Implementierung eines Testfalls mit Hilfe des JUnit Frameworks

Schritt 2: führe diesen Testfall aus: Der zuvor modellierte Testfall konnte anschließend im zweiten Schritt ausgeführt werden. Da die Szenariospezifikation zu diesem Zeitpunkt weder das erwartete `setDeratingFactor()` Event noch eine Verhaltensbeschreibung der `deratingComponent` enthielt, hatte die Ausführung des Testfalls wie erwartet ein negatives Ergebnis.

Schritt 3: adaptiere oder erweitere die Spezifikation: Folglich wurde Schritt drei durch Hinzufügen des in Listing 6.3 dargestellten Szenarios erweitert. Hier erfolgt in Zeile 3 zunächst die Abfrage, ob sich der eingelesene Wert in dem Temperaturbereich von $65°C$ bis $75°C$ befindet. Da die im Testfall angegebene Temperatur in diesem Bereich liegt,

wird in Zeile 4 die Ausführung des Events `setDeratingFactor()` angefordert, wobei der Parameter dieses Events den berechneten Deratingfaktor enthält.

```
1 // Derating factor derates linearly from 1.0 to 0.0 when coolant temperature is between 65
      and 75
2 cycleScenario(deratingComponent){
3  if (deratingComponent.coolantTemp in 65..75)
4  request(deratingComponent.setDeratingFactor((75.0-deratingComponent.coolantTemp)/10))
5 }
```

Listing 6.3: Formalisierte Anforderung Req2 zur Leistungsreduzierung in einem definierten Temperaturbereich

Schritt 4: führe alle Testfälle aus: Die erneute Ausführung des Testfalls (Listing 6.2) führte durch das Zusammenspiel mit der modellierten Anforderung *Req2* (Listing 6.3) zu einem positiven Testergebnis.

Schritt 5: überarbeite Testfälle und die Spezifikation: Der letzte Schritt bestand in der Überarbeitung des Testfalls und der Spezifikation. Um bspw. nicht bei jedem neuen Testfall die Sequenz von Temperatur setzen, Ausführungszyklus starten, auf das Ergebnis warten und Ausführungszyklus stoppen implementieren zu müssen, konnte der Testfall wie in Listing 6.4 dargestellt vereinfacht werden. Hierbei geben die ersten beiden Parameter die Temperatur des Kühlmittels sowie der Leiterkarten an, und der dritte Parameter gibt den erwarteten Faktor für die Regulierung der Ausgangsleistung an.

```
1 @Test
2 fun `At 70 degree the DeratingFactor is 0-5`() = runTest(deratingScenarioProgram){
3  deratingIOSequence(Triple(70, 70, 0.5))
4 }
```

Listing 6.4: Vereinfachte Implementierung des Testfalls

Das Vorgehen nach der TDSS Methode wurde für alle verfügbaren Anforderungen durchgeführt. *Req1* und *Req3* ließen sich in gleicher Form formalisieren, wobei auch die Ausführung aller Testfälle in Schritt 4 keine Auffälligkeiten zeigte.

Die Formalisierung der Anforderungen *Req4* und *Req5* war mit einem höheren Modellierungsaufwand verbunden, da hier neben den Temperaturwerten die zeitliche Abfolge der Werte zu berücksichtigen war. Das Ergebnis der Formalisierung von *Req4* ist das Szenario in Listing 6.5. Das Szenario wird ausgeführt, wenn die Kühlmitteltemperatur innerhalb eines Testfalls gesetzt wird (Trigger Event `setCoolantTemperature` in Zeile 2). Anschließend wird in Zeile 6 die Variable `currentTemp` mit dem Temperaturwert des Kühlmittels aus dem Trigger Event initialisiert. Diese Variable wird in Zeile 21 in den folgenden 50 Ausführungszyklen mit dem jeweils aktuellen Temperaturwert überschrieben, wobei angenommen wird, dass ein Ausführungszyklus 10ms entspricht, was zu dem in der Anforderung spezifizierten Zeitraum von 5s führt. Innerhalb dieses Zeitraums wird in Zeile 14 überprüft, ob die initiale Temperatur von der aktuellen Temperatur um mehr als 5°*C* abweicht. Ist dies der Fall, wird in Zeile 16 das Event `exceptionalTemperatureIncrease()` angefordert. Zusätzlich wird in Zeile 17, entsprechend der Anforderung *Req4*, das Abschalten der Energietransformation mit dem Event `setDeratingFactor(0.0)` angefordert.

Über die Spezifikation von `forbiddenEvents` innerhalb des Szenarios erfolgt die Synchronisation der zyklischen Ausführung, was bedeutet, dass die Events `startCycle()` und `endCycle()` nur an den Stellen auftreten dürfen, an denen explizit auf diese Events gewartet wird (Zeilen 13 und 20). Zusätzlich erfolgt über das Hinzufügen von `deratingComponent receives DeratingComponent::setDeratingFactor` zu den `forbiddenEvents`, dass im Falle eines unerlaubten Temperaturanstiegs nur das explizit in Zeitle 17 angeforderte Event `setDeratingFactor(0.0)` auftreten darf.

```
1  // Shutdown if coolant temp increases more than 5 degrees within a period of 5 seconds
2  scenario(deratingComponent receives DeratingComponent::setCoolantTemperature){
3
4   // read new coolant temperature value
5   val initialTemp = it.parameters[0] as Int
6   var currentTemp = initialTemp
7
8   // allow cycles' start/end only where waited for below.
9   forbiddenEvents.add(deratingComponent.startCycle())
10  forbiddenEvents.add(deratingComponent.endCycle())
11
12  for(i in 1..50){ // for 50 cycles = 5 seconds...
13   waitFor(deratingComponent.startCycle())
14   if (currentTemp - initialTemp > 5){
15    forbiddenEvents.add(deratingComponent receives DeratingComponent::setDeratingFactor)
16    request(deratingComponent.exceptionalTemperatureIncrease())
17    request(deratingComponent.setDeratingFactor(0.0))
18    break // end the for loop and, hence, the scenario
19   }
20   waitFor(deratingComponent.endCycle())
21   currentTemp = waitFor(deratingComponent receives DeratingComponent::
         setCoolantTemperature).parameters[0] as Int
22  }
23 }
```

Listing 6.5: Szenario zur Abschaltung der Energietransformation bei plötzlichen Temperaturanstiegen des Kühlmittels (Req4)

Das Szenario verdeutlicht, dass auch Anforderungen mit einem zeitlichen Bezug modelliert werden können. Dabei fällt auf, dass das Szenario in jedem Ausführungszyklus durch eine neue Kühlmitteltemperatur getriggert wird. Hierdurch können bis zu 50 Instanzen des Szenarios aktiv sein, was in der späteren Implementierung der Anforderung so nicht umgesetzt wurde, aber eine intuitive Modellierung der Anforderung unterstützt.

Der TDSS Methode folgend führte die Ausführung dieser Anforderung in Kombination mit den bereits zuvor formalisierten Anforderungen zu den in Bild 6-4 dargestellten Testergebnissen. Interessanterweise zeigen die Ergebnisse den gleichen Widerspruch in den Anforderungen, der auch im MIL-Test identifiziert wurde. Um diesen Widerspruch aufzulösen, wurde festgelegt, dass *Req4* einen Sonderfall berücksichtigt und höher zu priorisieren ist als *Req1* und *Req2*. Um das in den bereits modellierten Anforderungen zu berücksichtigen, wurden diese wie in Listing 6.6 dargestellt erweitert. Hier ist in Zeile 3 durch Hinzufügen des Events `exceptionalTemperatureIncrease()` zu den `interruptingEvents` spezifiziert, dass die Ausführung dieses Szenarios unterbrochen wird, falls in dem Szenario in Listing 6.5 ein unerlaubter Temperaturanstieg identifiziert wurde.

```
1  // Derating factor derages linearly from 1.0 to 0.0 between [65..75]
2  cycleScenario(deratingComponent){
3   interruptingEvents.add(deratingComponent.exceptionalTemperatureIncrease())
4   if (deratingComponent.coolantTemp in 65..75)
5   request(deratingComponent.setDeratingFactor((75.0-deratingComponent.coolantTemp)/10))
```

```
6 }
```

Listing 6.6: Szenario aus Listing 6.3 mit Berücksichtigung der festgelegten Prioritäten der Anforderungen

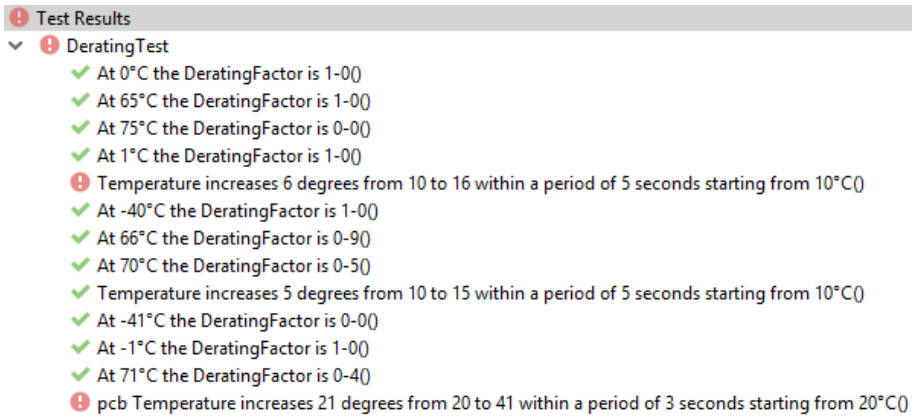


Bild 6-4: Testergebnisse durch die Ausführung der TDSS Methode.

6.1.4 Vergleich und Bewertung der Ergebnisse

Die Anwendung der TDSS Methode zeigt, dass die Formalisierung und automatisierte Analyse der ausgewählten Anforderungen mittels SMLK möglich ist. Einfache Anforderungen wie *Req1*, *Req2* und *Req3* waren direkt und ohne großen Modellierungsaufwand umsetzbar. Komplexere Anforderungen wie *Req4* und *Req5* sind ebenfalls umsetzbar, hierfür wurde jedoch mehr Zeit benötigt und das resultierende Szenario ist umfangreicher.

Dass durch das etablierte Vorgehen und durch die TDSS Methode die gleichen Inkonsistenzen in den Anforderungen identifiziert wurden ist als positiv zu bewerten, insbesondere da die TDSS Methode als Hilfsmittel innerhalb der Anforderungsanalysephase (vgl. Kapitel 3.1.2) zur früheren Identifikation von Inkonsistenzen zwischen den Anforderungen beiträgt. Durch das etablierte Vorgehen wurden die Inkonsistenzen zu einem vergleichsweise späten Zeitpunkt in der Entwicklung im Rahmen des MIL Tests aufgedeckt.

Die Kombination eines testgetriebenen Entwicklungsansatzes mit einer szenariobasierten textuellen Modellierung kommt ebenfalls zu einem positiven Ergebnis, da hierdurch die Formalisierung in kleinen Iterationen forciert wurde und durch das Ausführen der Testfälle eine unmittelbare Rückmeldung des Systemverhaltens erfolgt. Einerseits unterstützt TDSS die fokussierte Formalisierung und Analyse einzelner Anforderungen, getrieben durch das Ausführen von einzelnen Testfällen (Schritt 1 und 2). Andererseits wird durch die resultierende Szenariospezifikation und das Ausführen aller bereits implementierten Testfälle die automatisierte und zusammenhängende Analyse aller Anforderungen möglich (Schritt 4), was schließlich zu der Identifikation der Inkonsistenzen führte.

Schlussfolgerung: TDSS ermöglicht die frühe Identifikation von Inkonsistenzen in funktionalen Anforderungen. Dabei unterstützt das testgetriebene Vorgehen und die verwendete

Modellierungssprache eine iterative Modellierung der Szenarien. Durch die testgetriebene und szenariobasierte Modellierung wird eine intuitive Modellierung in kleinen Iterationen adressiert, die eine anwendungsorientierte Modellierung forciert. Dennoch ist für die Modellierung mittels SMLK im Vergleich zu den etablierten manuellen Analysen zusätzlich Wissen nötig, und es entsteht zusätzlicher Modellierungsaufwand. Der Aufwand für die Anforderungsmodellierung und der Aufwand zur Korrektur der Anforderungen und des Implementierungsmodells innerhalb des etablierten Prozesses wurden als identisch angesehen. Somit ist fraglich, ob für kleine abgegrenzte Funktionen mit einer geringen Anzahl von Anforderungen eine formalisierte Anforderungsanalyse einen Mehrwert erzeugt, die den zusätzlichen Aufwand und die Einführung neuer Methoden und Werkzeuge rechtfertigt.

6.2 Ansatz für die integrierte Anforderungsanalyse und Testspezifikation (Iteration 2)

Die Ergebnisse der zweiten Iteration wurden beim *Modeling in Automotive System and Software Engineering Workshop* (MASE20) und im darauf folgenden Jahr bei der *24th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (MODELS'21) vorgestellt. Die Ergebnisse der durchgeführten Fallstudie sind zudem Teil eines im *Journal of Systems and Software* (JSS) veröffentlichten Artikels. Ergänzend wurde eine Erweiterung der TDSS Methode und ein dazugehöriges Werkzeug bei der *27th International Working Conference on Requirements Engineering: Foundations for Software Quality* (REFSQ21) vorgestellt.

[FFV+23] FISCHBACH, J.; FRATTINI, J.; VOGELSANG, A.; MÉNDEZ, D.; UNTERKALMSTEINER, M.; WEHRLE, A.; HENAO, P. R.; YOUSEFI, P.; JURICIC, T.; RADDUENZ, J.; WIECHER, C.: Automatic Creation of Acceptance Tests by Extracting Conditionals from Requirements: NLP Approach and Case Study. In: Journal of Systems and Software, 2023

[WFG+21] WIECHER, C.; FISCHBACH, J.; GREENYER, J.; VOGELSANG, A.; WOLFF, C.; DUMITRESCU, R.: Integrated and Iterative Requirements Analysis and Test Specification: A Case Study at Kostal. In: 24th International Conference on Model Driven Engineering Languages and Systems, MODELS, Fukuoka, Japan, October 10-15, 2021, IEEE, 2021, S. 112–122

[WG21] WIECHER, C.; GREENYER, J.: BeSoS: A Tool for Behavior-driven and Scenario-based Requirements Modeling for Systems of Systems. In: REFSQ 2021 Workshops, Essen, Germany, April 12, 20, CEUR-WS.org, 2021

[WGW+21] WIECHER, C.; GREENYER, J.; WOLFF, C.; ANACKER, H.; DUMITRESCU, R.: Iterative and Scenario-Based Requirements Specification in a System of Systems Context. In: Requirements Engineering: Foundation for Software Quality - 27th International Working Conference, REFSQ, Essen, Germany, April 12-15, 2021, Springer, 2021, S. 165–181

[WJK+20] WIECHER, C.; JAPS, S.; KAISER, L.; GREENYER, J.; DUMITRESCU, R.; WOLFF, C.: Scenarios in the loop: integrated requirements analysis and automotive system validation. In: MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Canada, 18-23 October, 2020, Companion Proceedings, ACM, 2020, S. 35:1–35:10

Bild 6-5 veranschaulicht das Vorgehen der zweiten DSR Iteration. Hier ist ebenfalls das **Problembewusstsein** der Ausgangspunkt. Auf Basis der zuvor beschriebenen Schluss-

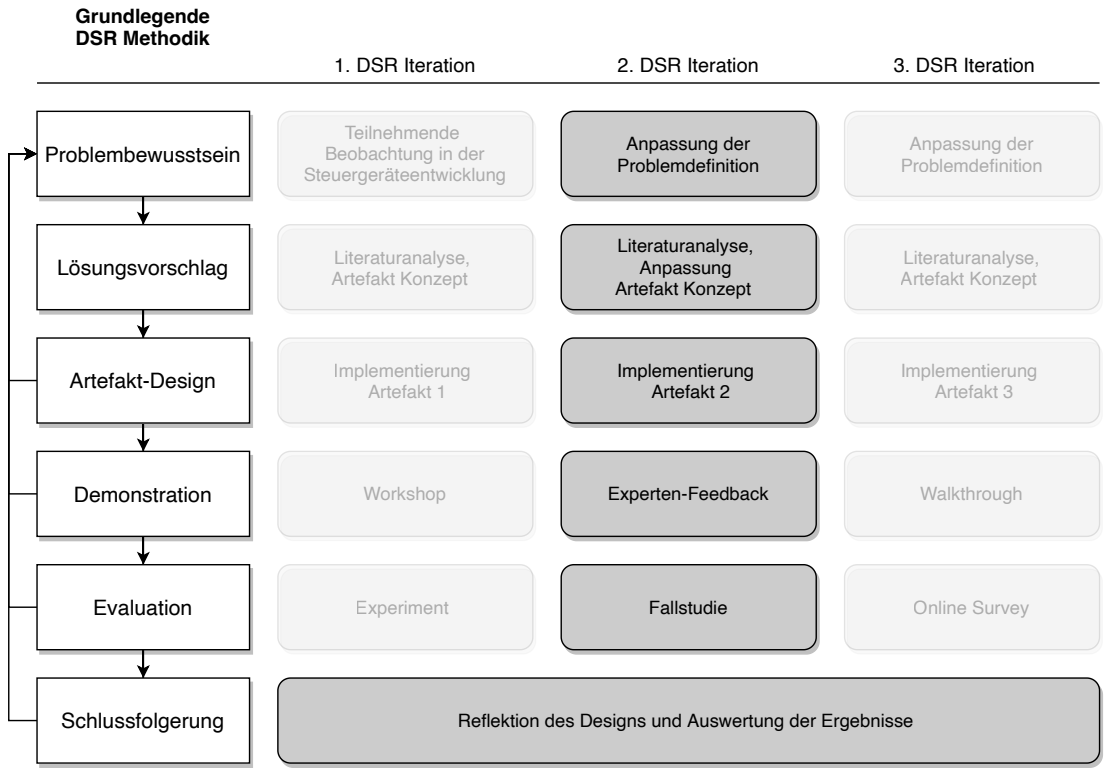


Bild 6-5: Vorgehensweise für die zweite DSR Iteration.

folgerung der ersten Iteration wurden die Literaturanalyse weiter fortgesetzt und die Handlungsfelder konkretisiert. Die Ergebnisse der Literaturanalyse sind in den in Kapitel 3.3.4 beschriebenen Herausforderungen für den Entwurf und die Spezifikation von Tests zusammengefasst. Der zweite **Lösungsvorschlag** basiert auf den identifizierten Ansätzen für den Entwurf und die Spezifikation von Testfällen (Kapitel 4.3). Das **Artefakt-Design** umfasst die Technik zur Testfallgenerierung [FFV+23; Fis22] und die Erweiterungen der Technik zur Modellierung der Anforderungen [WGW+21] (Kapitel 5.4.2 und 5.4.3) mit den zugehörigen Werkzeugen (Kapitel 5.6.2). Die **Demonstration** des entwickelten Artefakts ist mit Hilfe von Experten-Feedback innerhalb des beteiligten Industrieunternehmens dokumentiert [WJK+20]. Die **Evaluation** und **Schlussfolgerung** sind im Folgenden beschrieben und in [WFG+21] veröffentlicht.

Da die Anwendung der TDSS-Methode im Kern zu positiven Ergebnissen kommt, ist für die Evaluierung der zweiten Iteration von Bedeutung, wie die Methode effizient in

vorhandene Entwicklungsprozesse integriert werden kann, und ob die Aktivitäten weiter automatisiert werden können, um den zusätzlichen Aufwand für die Anwendung der Methode zu verringern. Dabei sind, ergänzend zu einfachen Funktionen mit einer geringen Anzahl von Anforderungen, komplexere Anwendungsfälle zu berücksichtigen. Bei der Automatisierung von Aktivitäten der TDSS-Methode ist insbesondere der Entwurf von Testfällen von Bedeutung. Da die Anforderungsmodellierung durch die Testfälle getrieben wird, hängt diese stark von der Qualität der Testfälle ab. Werden bspw.. Testfälle übersehen, wirkt sich dies direkt auf die zu modellierenden Anforderungen aus. Weitergehend ist für die Evaluierung relevant, ob das resultierende Anforderungsmodell für komplexere Anwendungsfälle einen Mehrwert liefert, der den zusätzlichen Aufwand für die Modellierung rechtfertigt.

Die Evaluierung ist in zwei Teile strukturiert. Zunächst ist die Anwendung der Testfallgenerierung und anschließend sind die Ergebnisse der erweiterten TDSS-Methode beschrieben. In beiden Fällen basiert die Evaluierung auf der Funktion *Plug Interlock* des OBCs (vgl. Kapitel 1.2).

6.2.1 Beschreibung der Funktion Plug-Interlock

Die Funktion *Plug Interlock* realisiert in Zusammenspiel mit der Ladedose des Fahrzeugs und den darin enthaltenen Sensoren und Aktuatoren die Verriegelung des Ladesteckers. Wenn der Stecker in die Ladedose gesteckt wurde, empfängt der OBC Pilot-Signale des Steckers. Nach Auswertung dieser Pilot-Signale erfolgt durch den OBC eine Ansteuerung der Aktuatoren innerhalb der Ladedose zur Verriegelung des Ladesteckers. Erst nach erfolgreicher Verrieglung des Ladesteckers ist die Aktivierung des Ladevorgangs möglich. Hierdurch soll verhindert werden, dass der Ladestecker bei einem aktiven Ladevorgang durch den Nutzer gezogen werden kann. Die Plug-Interlock Funktion ist nach den Vorgaben in [ISOa] mit einem ASIL Level A zu bewerten, was eine umfassende Spezifikation und Analyse von Anforderungen und Tests erfordert.

6.2.2 Anwendung der Testfallgenerierung

Für die Evaluierung der Testfallgenerierung wurde eine existierende Anforderungsspezifikation verwendet. Diese Anforderungsspezifikation beschreibt die Auswertung der Pilot-Signale als Teil der übergeordneten *Plug-Interlock* Funktion. Die Spezifikation umfasst insgesamt 255 Anforderungen, von denen 135 als funktionale Anforderungen klassifiziert werden können. 89 Anforderungen der Spezifikation definieren Schnittstellen. 4 Anforderungen beinhalten Hinweise für die Konfiguration der Komponente, und die übrigen 27 Anforderungen enthalten allgemeine Informationen für die Umsetzung. Eine Übersicht der Anforderungstypen ist in Bild 6-6 dargestellt. Relevant für die Fallstudie sind die 135 funktionalen Anforderungen. Wobei diese weiter unterteilt werden können in 79 Anforderungen, die kausale Abhängigkeiten beschreiben und 56 Anforderungen, die eine Funktionalität in einer statischen Weise beschreiben (bspw.: "*The signal* `signalName`

shall be set to `InitValue`). Diese Anforderungen wurden innerhalb der Fallstudie nicht

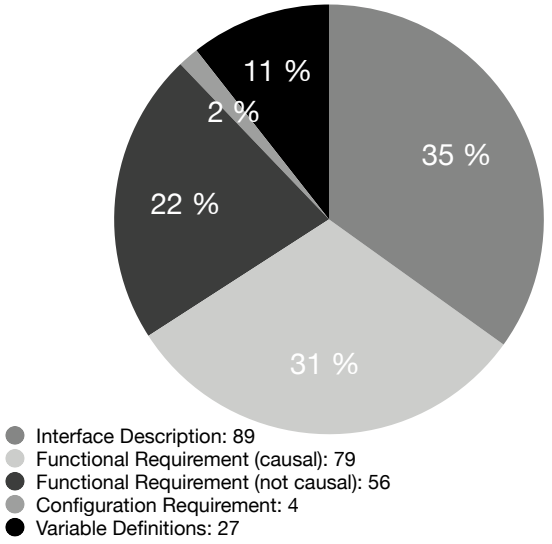


Bild 6-6: Anforderungstypen innerhalb der verwendeten Anforderungsspezifikation [WFG+21].

berücksichtigt. Für eine automatisierte Verarbeitung sind hier mehr Informationen nötig. Es sollte z.B. die Bedingung spezifiziert sein, *wann* das Signal zu setzen ist. Da diese Informationen in der bereitgestellten Spezifikation jedoch nicht enthalten waren, sind die 79 funktionalen Anforderungen Gegenstand der Fallstudie.

Ausgehend von diesen 79 Anforderungen wurde die Technik angewendet. Die Auswertung der Anwendung basiert auf einem Vergleich der generierten Testfälle mit den innerhalb des Industrieunternehmens manuell erstellten Testfallspezifikationen. Die Ergebnisse des Vergleichs lassen sich in drei Bereiche unterteilen. Der erste Bereich umfasst eine Überschneidung von manuell und automatisiert abgeleiteten Testfällen. Der zweite Bereich zeigt Testfälle die nur manuell abgeleitet wurden, und der dritte Bereich zeigt die Testfälle, die ausschließlich durch den automatisierten Ansatz erzeugt wurden. Bild 6-7 veranschaulicht die Ergebnisse.

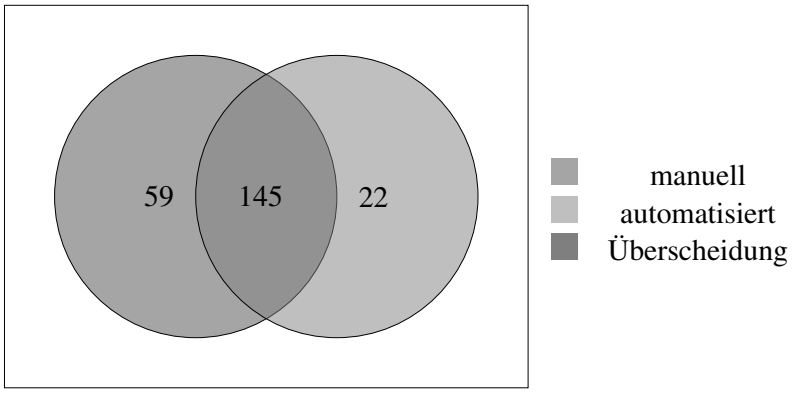


Bild 6-7: Vergleich von manuell und automatisiert erstellten Testfällen [WFG+21].

Bei der Analyse der manuell erstellten Testfallspezifikation wurden 5 Anforderungen

identifiziert, zu denen kein Testfall zugeordnet werden konnte. Insgesamt beinhaltet die existierende Testfallspezifikation 204 Testfälle, was eine Anzahl von 2,58 Testfällen pro Anforderung entspricht. Mit der Technik zur Testfallgenerierung konnten 68 der 79 Anforderungen verarbeitet werden. Insgesamt wurden so 167 Testfälle generiert. Der Vergleich der Testfälle verdeutlicht, dass die Mehrzahl (145) der manuell erstellten Testfälle auch automatisiert hergeleitet werden kann. Die Überschneidung liegt hier bei etwa 71%. 22 Testfälle wurden ausschließlich durch den automatisierten Ansatz generiert und 59 konnten lediglich manuell hergeleitet werden. Die Analyse der 22 ausschließlich generierten Testfälle kommt zu dem Ergebnis, dass diese technisch korrekt und relevant sind. Dass 59 Testfälle lediglich manuell abgleitet wurden, ist auf eine fehlerhafte Grammatik, komplexe Variablennamen oder fehlende Informationen in den Anforderungen zurückzuführen.

6.2.3 Anwendung der Anforderungsmodellierung

Ausgangspunkt für den zweiten Teil der Evaluierung war die Integration der Testfallgenerierung mit der TDSS-Methode (s. Kapitel 5.6.2). Die Anwendung ermöglicht die erfolgreiche Modellierung der *Plug Interlock* Anforderungen [WFG+21]. Ebenso ermöglicht das Anforderungsmodell die Simulation der Anforderungen. Wie in 6-8 veranschaulicht, sind die zur Realisierung der Funktion notwendigen Komponenten in dem Simulationswerkzeug dargestellt. Die Pfeile deuten den Informationsaustausch zwischen den Komponenten an, wobei die Interaktion schrittweise ausgeführt und mit den Erwartungen der Anwender abgeglichen werden kann.

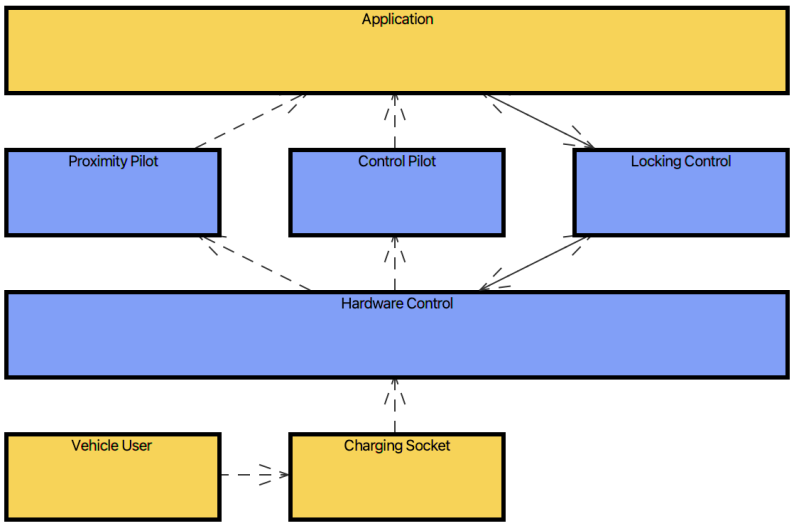


Bild 6-8: Simulation des Verhaltens der Plug Interlock Funktion

Als Ergebnis eines erfolgreichen Durchlaufs der Simulation konnte das in Bild 6-9 dargestellte Sequenzdiagramm generiert werden, was zur Dokumentation einer Systemanforderung der Funktion *Plug Interlock* angewendet werden kann.

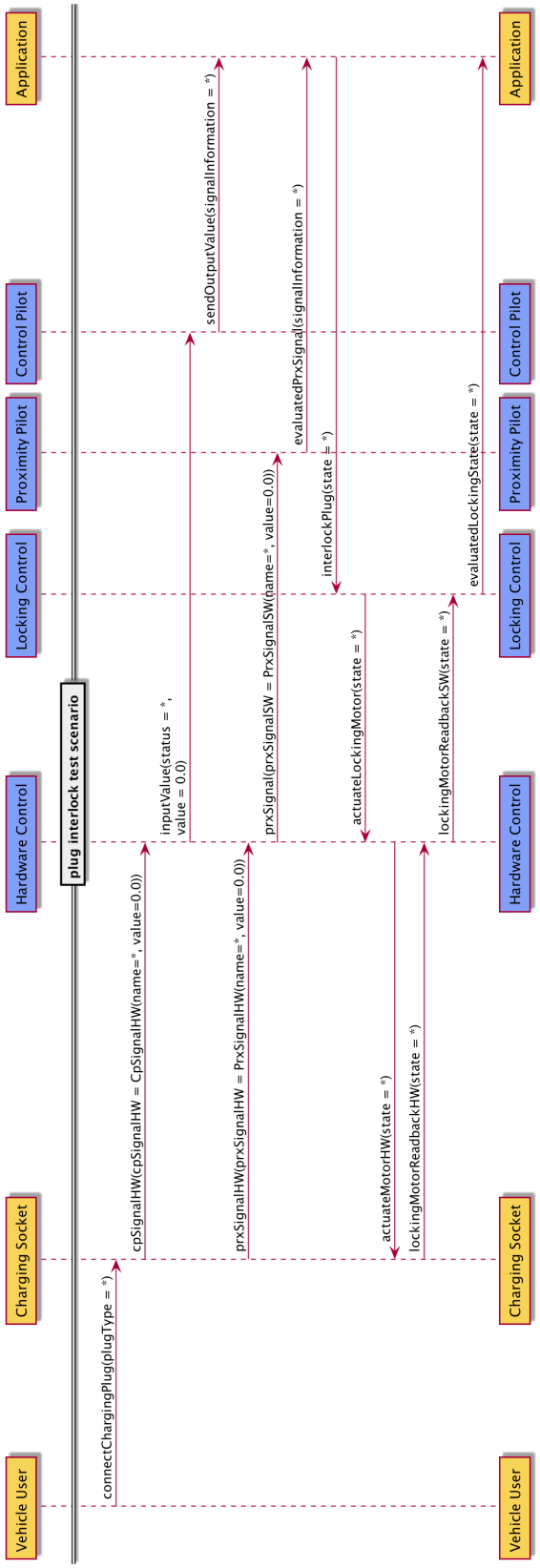


Bild 6-9: Dokumentierte Systemanforderung der Funktion Plug Interlock über ein Sequenzdiagramm.

6.2.4 Bewertung der Ergebnisse

Die Anwendung der Testfallgenerierung zeigt, dass diese Technik für komplexe Spezifikationen innerhalb der Automobilindustrie angewendet werden kann. Dabei kann ein Großteil der Testfälle automatisiert hergeleitet werden. Da sogar Testfälle generiert wurden, die in der manuell erstellen Testfallspezifikation nicht enthalten waren, kann der Ansatz den Testentwurfsprozess in Teilen verbessern.

Die Kombination der Testfallgenerierung mit der TDSS-Methode kann positiv bewertet werden, da durch die Automatisierung der Aufwand in Schritt 1 der TDSS Methode reduziert werden kann. Gleichzeitig wird durch den Vergleich der manuell und automatisiert erstellten Testfallspezifikation deutlich, dass die Testfälle eine ausreichende Qualität haben, um für die Modellierung der Anforderungen verwendet zu werden.

Durch das entstandene Anforderungsmodell wird die Simulation der Systemanforderung möglich, wobei die Simulation die Konsolidierung der Systemanforderungen zwischen dem Test-Designer und dem Systemarchitekten unterstützt. Wie in Kapitel 4.4.1 beschrieben, fördert TDM über die Testfälle und Testergebnisse die disziplinübergreifende Kommunikation. Da in der Simulation die Ausführung schrittweise durch die Testfälle angestoßen wird, wird die Konsolidierung des Systemverhaltens zusätzlich gefördert und ermöglicht weitergehende Analysen, wenn das Anforderungsmodell nicht den Erwartungen des Test-Designers oder des Systemarchitekten entspricht.

Schlussfolgerung: Die Ergebnisse der ersten Iteration zeigen durch die Anwendung der TDSS-Methode, wie Anforderungen anwendungsorientiert formalisiert und für die Anforderungsanalyse eingesetzt werden können. Der zusätzlich Aufwand ist jedoch für überschaubare Funktionen mit wenigen Anforderungen nicht gerechtfertigt. Die Ergebnisse der zweiten Iteration zeigen weitergehend, dass der Aufwand durch die Kombination geeigneter Techniken reduziert werden kann und dass die Verwendung des resultierenden Anforderungsmodells für die Simulation verwendet werden kann und über die Generierung von Systemanforderungen in Form von Sequenzdiagrammen den Mehrwert erhöht. Somit ist das Artefakt der zweiten DSR-Iteration potentiell geeignet, um in der Anforderungsanalysephase bzw. bei der Testspezifikation für komplexe Funktionen zu unterstützen. Für eine abschließende Bewertung sind diese Techniken jedoch im Kontext der ganzheitlichen Systementwicklung zu betrachten.

6.3 Ansatz für die Systemodellierung und die Wiederverwendung von Lösungswissen (Iteration 3)

Die Ergebnisse der dritten Iteration wurden im *Systems Engineering Journal* veröffentlicht.

[WMG+24] WIECHER, C.; MANDEL, C.; GÜNTHER, M.; FISCHBACH, J.; GREENYER, J.; GREINERT, M.; WOLFF, C.; DUMITRESCU, R.; MENDEZ, D.; ALBERS, A.: Model-based Analysis and Specification of Functional Requirements

and Tests for Complex Automotive Systems, Systems Engineering, Wiley, 2024, DOI: 10.1002/sys.21748

Wie in den beiden vorangegangen Iterationen ist auch hier das **Problembewusstsein** der Ausgangspunkt. Nach der zuvor beschriebenen Schlussfolgerung ist zu untersuchen, wie die bisher evaluierten Techniken in die Systementwicklung integriert werden können und wie Systemarchitekten und Test-Designer bei der Anwendung unterstützt werden können. Die Literaturanalyse führt zu den in Kapitel 4.1 beschriebenen Ansätzen für die Systemmodellierung. Der **Lösungsvorschlag** basiert auf den in Kapitel 4.1.1 beschriebenen MBSE Grundlagen und den Techniken zur kontinuierlichen Validierung (Kapitel 4.1.3). Das **Artefakt-Design** ist beschrieben durch die Referenzarchitektur für die Systementwicklung (Kapitel 5.2), das Vorgehensmodell (Kapitel 5.3) und den Ansatz für die Wiederverwendung von Lösungswissen (Kapitel 5.5) sowie die zugehörige Werkzeugunterstützung (Kapitel 5.6.1). Die **Demonstration** erfolgt über eine Anwendung der Systematik in dem Industrieunternehmen, und die **Evaluierung** wurde über ein Online-Survey durchgeführt. Sowohl die Demonstration als auch die Evaluierung erfolgt anhand der Funktion *Timer-Charging* (s. Kapitel 1.2).

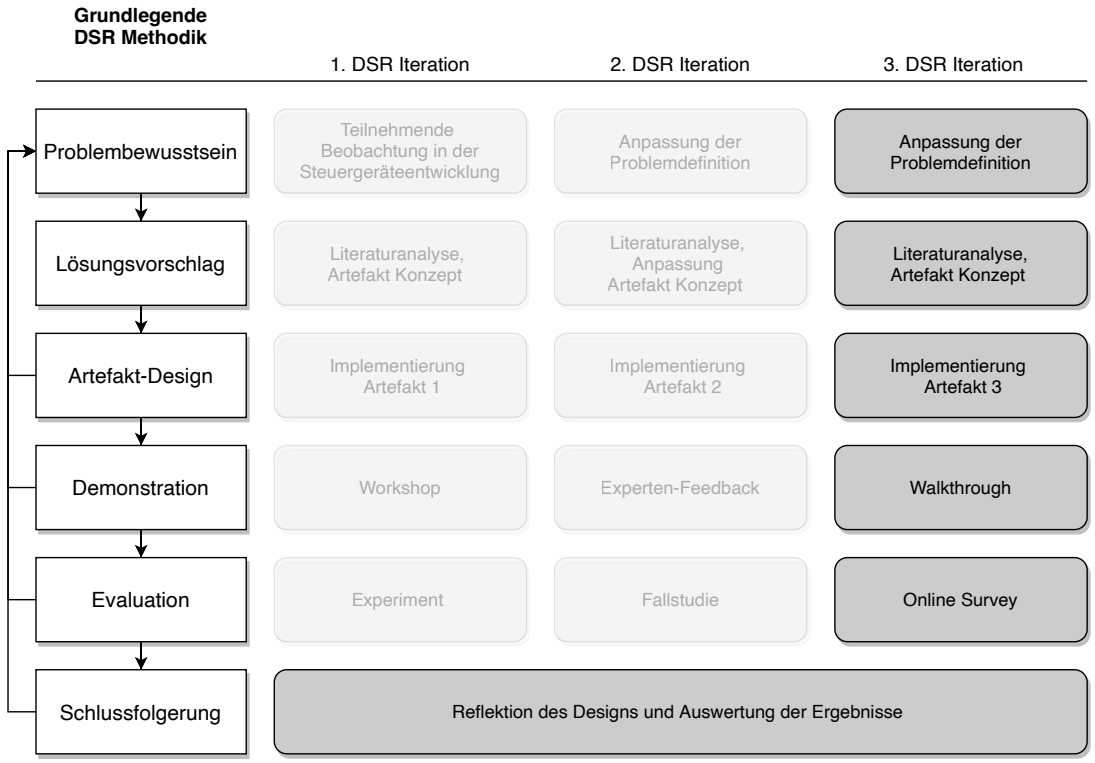


Bild 6-10: Vorgehensweise für die dritte DSR Iteration.

6.3.1 Beschreibung der Funktion Timer-Charging

Wie in Kapitel 1.2 bereits angedeutet, ermöglicht diese Funktion die Konfiguration eines zeitgesteuerten Ladevorgangs. Somit wird die grundlegende Ladefunktionalität des

Fahrzeugs um Konfigurationsmöglichkeiten erweitert, wobei die Parameter, wie z.B. der Ladezustand (SOC), zu einer definierten Abfahrtzeit über Bediengeräte vorgegeben werden können. Diese Bediengeräte können z.B. über ein Human Machine Interface (HMI) innerhalb des Fahrzeugs oder auch über Fahrzeug externe Geräte realisiert sein, die wiederum in auch anderen Kontexten eingebunden sein können. Das zu Beginn eingeführte Bild 1-1 skizziert die Zusammenhänge zwischen den Systemen auf unterschiedlichen Ebenen.

6.3.2 Anwendung der Systematik - Industrieunternehmen

Zur Demonstration des entwickelten Artefakts wurde die Methode zur integrativen Systemmodellierung (Kapitel 5.3.5) innerhalb des Industrieunternehmens angewendet. Ausgangspunkt waren 67 Stakeholder-Anforderungen, die auch der Startpunkt für die Entwicklung der *Timer Charging* Funktion nach dem etablierten Entwicklungsprozess waren. Wie in Bild 6-11 dargestellt, konnten diese Anforderungen in drei unterschiedliche Kategorien eingeteilt werden, wobei die größte Gruppe funktionale Anforderungen sind.

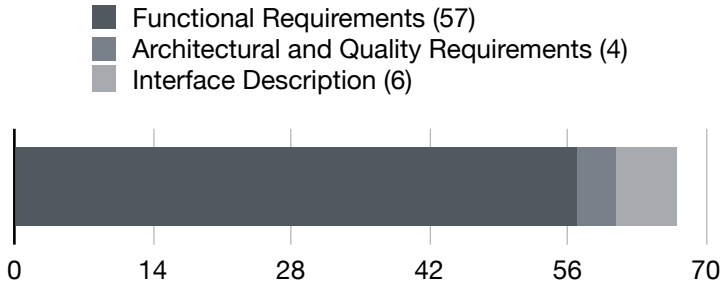


Bild 6-11: Unterschiedliche Anforderungstypen als Ausgangspunkt für die Entwicklung der Timer Charging Funktion.

Zur Anwendung der entwickelten Methode war das in iQUAVIS umgesetzte MBSE-Framework (Kapitel 5.6.1) der Startpunkt.

Problemraum (I1):

Da aus Sicht eines Tier1 Zulieferers die Entwicklung eines Systems üblicherweise mit einem initialen Satz an Stakeholder-Anforderungen erfolgt, wurde auch hier mit der Modellierung der Stakeholder-Anforderungen innerhalb des Problemraums begonnen. Dazu wurde in dem Werkzeug die Aktivität *Modellierung von Stakeholdern und deren Anwendungsfällen* ausgewählt, um über die in der Aktivität hinterlegten Diagramme auf Basis der definierten Sichten die Modellierung durchzuführen (vgl. Bild 4-3). Als Resultat der Aktivität wurde das in Bild 6-12 dargestellte Diagramm ausgearbeitet. Es wurde ein Stakeholderbedarf definiert: *As a user I want to configure the SOC that shall be reached at a certain time*. Diesem Stakeholderbedarf wurde ein Anwendungsfall zugeordnet: *The user configures SOC and departure time on an HMI*. Ausgehend von den bereitgestellten 67 Stakeholder-Anforderungen konnten insgesamt 8 Anwendungsszenarien abgeleitet werden, die dem Anwendungsfall zugeordnet werden konnten. Über die Aktivität *Modellierung*

von Stakeholder-Anforderungen war eine Vernetzung der Stakeholder-Anforderungen mit den identifizierten Anwendungsszenarien möglich, die analog zu Bild 6-12 über eine Baumstruktur erfolgte. Ebenso war mit den vorhandenen Informationen die Ausführung

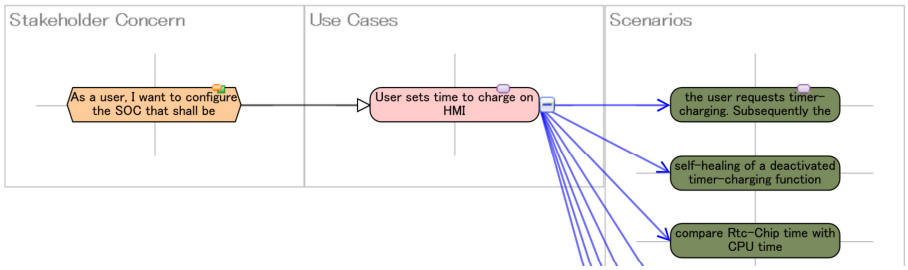


Bild 6-12: Unterschiedliche Anforderungstypen als Ausgangspunkt für die Entwicklung der Timer Charging Funktion.

der Aktivität *Modellierung Systemkontext* möglich, die durch das in Bild 6-13 dargestellte Diagramm unterstützt wurde. Demnach waren für die Umsetzung des Anwendungsfalls 3 interagierende Systeme notwendig, die mit dem zu entwickelnden System vernetzt wurden.

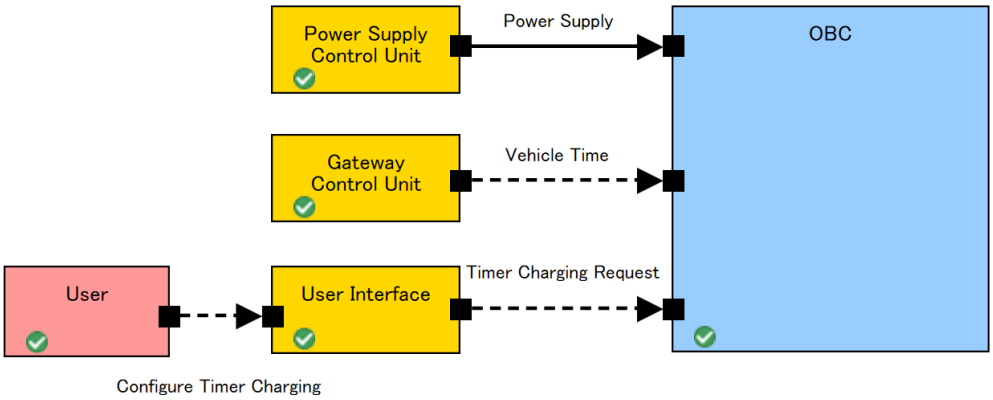


Bild 6-13: Blockdiagramm zur Ausführung der Aktivität Modellierung Systemkontext.

Validierungssystem (I2):

Als Einstiegspunkt für die Modellierung des Validierungssystems wurde die Aktivität *Ableiten von Validierungsbedarfen* gewählt. Hierzu wurde als Diagramm eine Baumstruktur verwendet. Wie in Bild 6-14 dargestellt, war so das Anlegen und Vernetzen von Validierungsbedarfen mit den Elementen des Problemraums möglich.

Zur weiteren Ausgestaltung des Validierungssystems wurden die Aktivitäten *Konsolidieren von Validierungszielen und Tests* und *Modellierung von Testumgebungen* ausgeführt, wobei hier ebenfalls Baumstrukturen für das Anlegen und Vernetzen der Elemente des Validierungssystems verwendet wurden. Zur Konsolidierung der Modellinhalte wurden Tabellen gewählt, die eine übersichtliche Darstellung der Modellinhalte ermöglichten. Bild 6-15 zeigt, wie die aus den Stakeholder-Anforderungen generierten Testfälle innerhalb des Systemmodells über die Tests miteinander vernetzt sind.

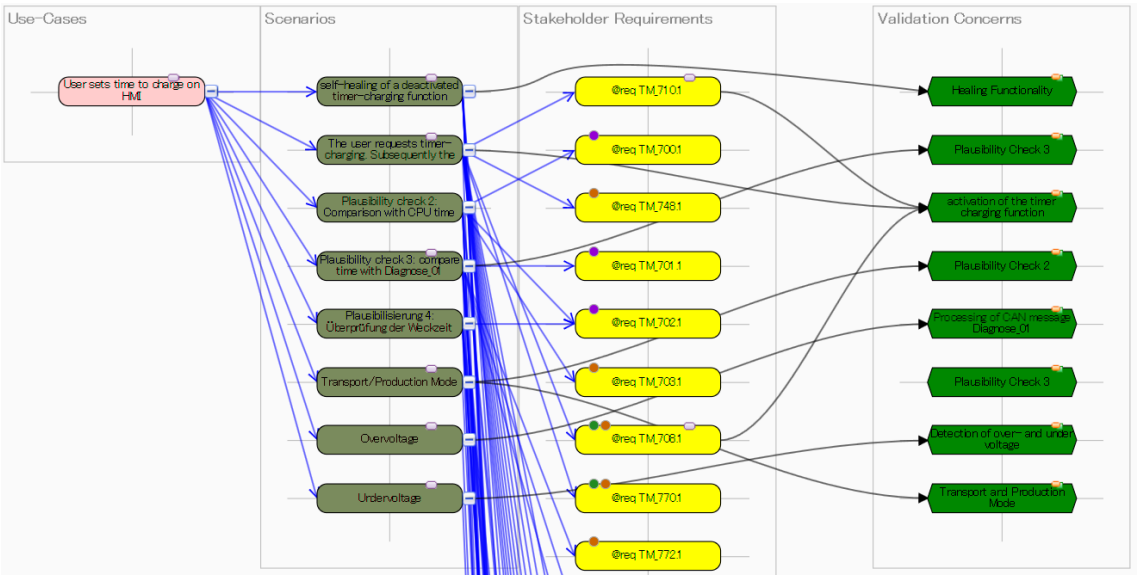


Bild 6-14: Vernetzung von Validierungsbedarfen mit den Elementen des Problemraums.

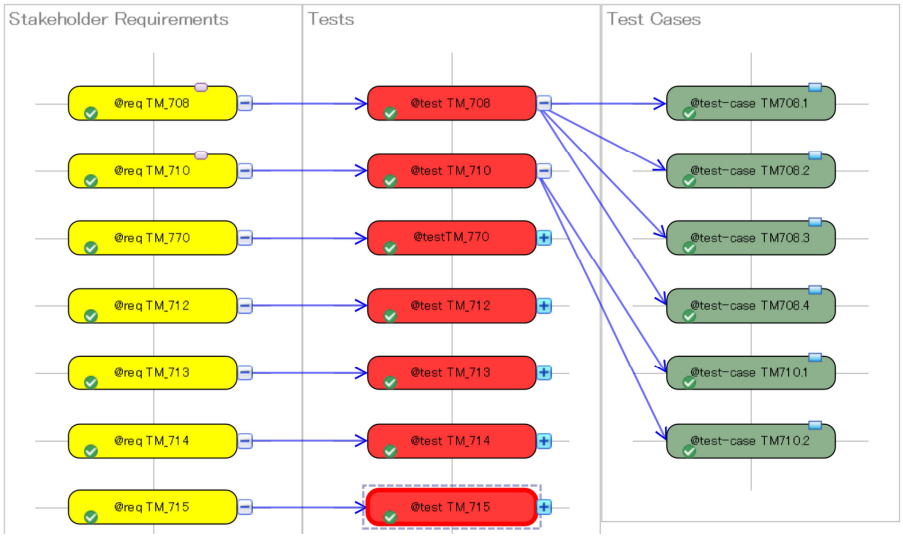


Bild 6-15: Vernetzung von Testfällen innerhalb des Validierungssystems.

Die aus dem Systemmodell ausgeleiteten Tabellen waren gleichzeitig die Basis für die Aktivitäten *Validierung von Anforderungen* und *Verifikation von Anforderungen*. Die Tabelle in 6-16 zeigt die für die Verifikation verwendete Testfallspezifikation. Hier wird deutlich, dass die Stakeholder-Anforderung mit einem Test vernetzt ist. Aus der Stakeholder-Anforderung wurden vier Testfälle generiert, die wiederum dem Test zugeordnet sind. Ebenso ist der Test mit einer Testumgebung vernetzt, die zur Testdurchführung erforderlich ist. Ein weiterer wichtiger Inhalt der Testfallspezifikation sind die vernetzten Systemanforderungen, die über die jeweilige Bezeichnung (z.B. SMLK TM_708.1) mit dem Anforderungsmodell verlinkt sind. Über die Anwendung der TDSS-Methode führte die Testdurchführung dabei neben der formalisierten Systemanforderung zu dem in der Tabelle dargestellten Testergebnis.

Test	Requirement		System Requirement	Test Case		Test Environment	Test Result
	ID	Text		ID	Description		
@test TM_708	@req TM_708	After the customer has activated the timer charging function and set the destination SOC and the departure time, the software of the OBC shall transmit this data to the application	SMLK TM_708.1	@test-case TM708.1	activate timer charging, set destination SOC, set departure time. The data is transmitted to the application	SysIT	passed
			SMLK TM_708.2	@test-case TM708.2	don't activate timer charging, set destination SOC, set departure time. The data is not transmitted to the application		passed
			SMLK TM 708.3	@test-case TM708.3	activate timer charging, don't set destination SOC, set departure time. The data is not transmitted to the application		passed
			SMLK TM 708.4	@test-case TM708.4	activate timer charging, set destination SOC, don't set departure time. The data is not transmitted to the application		passed
@test TM_710	@req TM_710	The software of the OBC must set the clock time of the Hardware-Timer to the time contained in the CAN message "CanMessageTimerCharging" when triggered by the signal "TimerChargingRequest"	SMLK TM 710.1	@test-case TM710.1	set TimerChargingRequest. Clock time is set.	SwIT	passed
			SMLK TM 710.2	@test-case TM710.2	don't set TimerChargingRequest. Clock time is not set.		passed

Bild 6-16: Testfallspezifikation für die Verifikation von Anforderungen.

Zur Validierung des Anforderungsmodells war die in Bild 6-17 dargestellte Testzielspe-

Validation Goal			Test Scenario	Test	Test Cases	Test Environment	Test Result
User can set SOC and departure time	The plausibility of the clock time can be checked	The OBC can repair a deactivated timer-charging function	self-healing	@testTM_712	@test-case TM712.1	SwIT	passed
			plausibility check	@testTM_770	@test-case TM_770.3	SwIT	not operated
				@test TM_770	@test-case TM770.1	SwIT	passed
			user activates the timer charging function	@test TM_708	@test-case TM708.1	SysIT	passed
				@test TM_710	@test-case TM710.1	SwIT	passed

Bild 6-17: Testzielspezifikation für die Validierung von Anforderungen.

zifikation die Basis. Hier waren die Validierungsziele der Ausgangspunkt. Diese sind mit den Testszenarien vernetzt, die wiederum aus einer Kombination der vorhandenen Testfälle ausführbar sind. Wie auch in der Testfallspezifikation, ist die Vernetzung mit den Testumgebungen und dem Testergebnis dargestellt.

Systemanforderungen (I3):

Somit beinhalten die Testfall- und Testzielspezifikation sowohl die Eingangsdaten (Testszenarien und Testfälle) als auch die Ergebnisse (Testergebnisse und Systemanforderungen) der TDSS-Methode. Die Modellierung der Systemanforderungen erfolgte außerhalb des Systemmodells. Zur Veranschaulichung zeigt Listing 6.7 den modellierten Testfall @test-case TM710.1. Der Testfall beinhaltet Kontextinformationen aus dem Systemmodell, wie die Interaktion mit dem HMI. In diesem Fall wurde die *Timer Charging* Funktion aktiviert, ein SOC und die Abfahrtzeit gesetzt. Zuletzt wird erwartet, dass der OBC die korrekte Zeit zurücksendet.

```
@Test
fun `@test-case 710 1`() = runTest(timerChargingScenarioProgram) {
 val currentTimeCAN = localTime
 request(hmiControlUnit sends obc.comStack.timerChargingRequest(
 activateTimerCharging = true,
 destinationSOC = 100,
 departureTime = localTime.plusHours(6)))
 request(gatewayControlUnit sends obc.comStack.clockTime(currentTimeCAN))
 waitFor(obc.rtcHandler.readbackClockTime(localTime))
}
```

Listing 6.7: Modellierung des Testfalls @test-case TM710.1 zur Modellierung der Systemanforderung SMLK TM710.1.

Getrieben durch diesen Testfall wurde die Systemanforderung SMLK_TM710.1 wie in Listing 6.8 umgesetzt. Das Listing beschreibt die Interaktionen von Komponenten des OBCs um schließlich die im Testfall erwarteten Daten bereitzustellen.

```
// SMLK TM 710.1
scenario(hmiControlUnit sends ComStack::timerChargingRequest.symbolicEvent()){
 val activateTimerCharging = it.parameters[0] as Boolean
 val destinationSOC = it.parameters[1] as Int
 val departureTime = it.parameters[2] as LocalDateTime
 request(obc.comStack sends obc.rtcHandler.timerChargingRequest(
 activateTimerCharging, destinationSOC, departureTime))
 request(obc.rtcHandler sends obc.comStack.getCurrentTime())
 request(obc.comStack sends obc.rtcHandler.currentTime(obc.comStack.vehicleClockTime))
 request(obc.rtcHandler sends obc.rtcChip.setClockTime(obc.rtcHandler.vehicleClockTime))
 request(obc.rtcChip sends obc.rtcHandler.readbackClockTime(obc.rtcChip.clockTime))}
```

Listing 6.8: Modellierung der Sytemanforderung SMLK TM710.1.

Auf diese Weise war es möglich, auf Basis der vernetzten Informationen innerhalb des Systemmodells ein Anforderungsmodell zu erstellen. Um eine weitergehende Analyse der

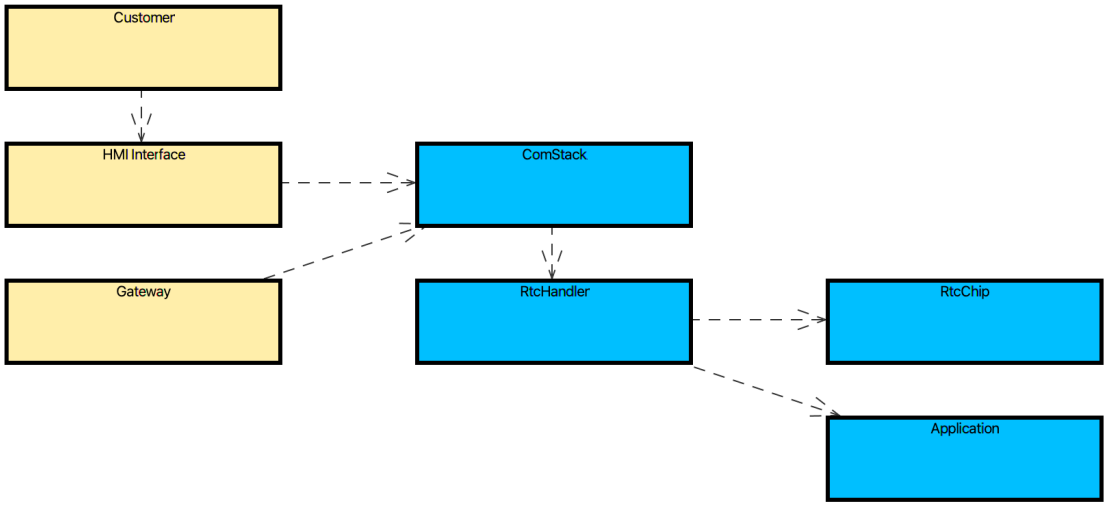


Bild 6-18: Interagierende Systeme und Komponenten des OBCs

Systemanforderungen zu unterstützen, war das Anforderungsmodell gleichzeitig die Basis für die Simulation des Systemverhaltens. Bild 6-18 zeigt die Anwendung des Werkzeugs. Dargestellt sind die interagierenden Systeme und die internen Komponenten des OBCs als Ergebnis der Anforderungsmodellierung.

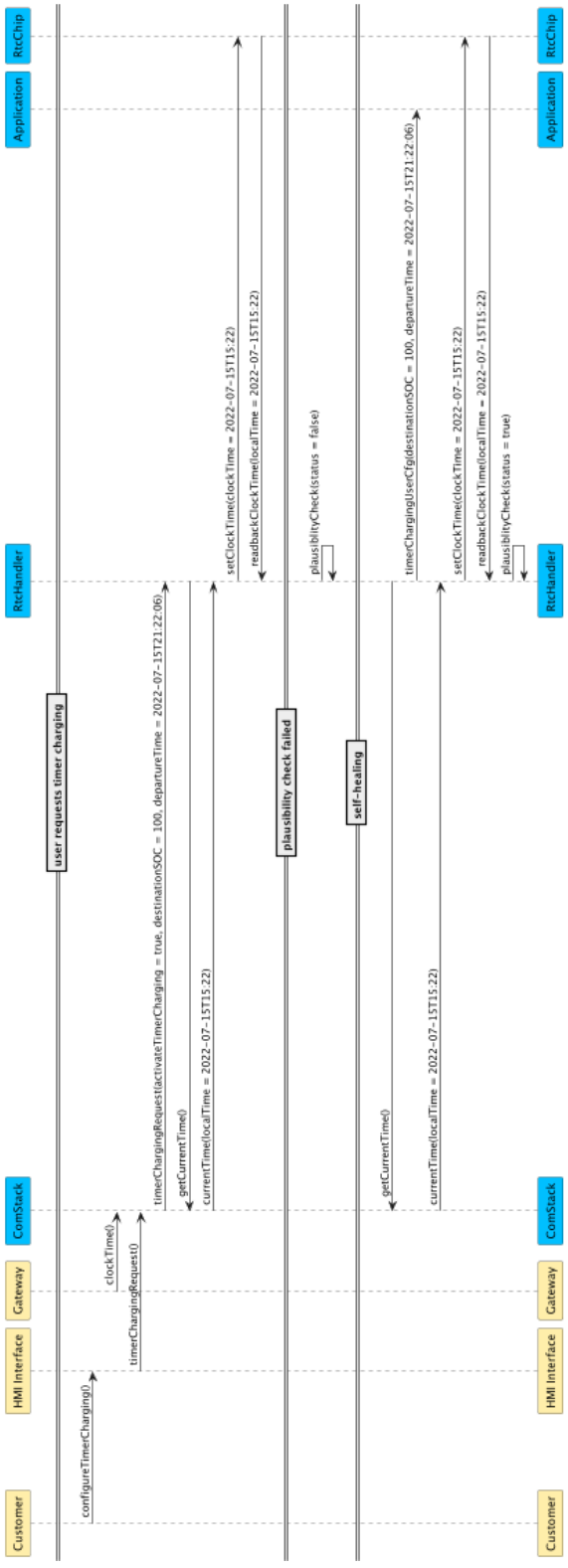


Bild 6-19: Dokumentation der Systemanforderung als Ergebnis der Anforderungssimulation.

Die Simulation der Anforderung war wiederum Basis für die Generierung des in Bild 6-19 dargestellten Sequenzdiagramms, das zur grafischen Dokumentation der Systemanforde-

rung verwendet wurde.

6.3.3 Bewertung der Anwendung

Die Integration der bisher evaluierten Artefakte mit den Techniken zur Systemmodellierung ermöglichen die Anwendung in der Systementwicklung. Über die TDSS-Methode erfolgt eine Verbindung der Elemente des Validierungssystems mit den Systemanforderungen innerhalb des Produktsystems, und über die Validierung und Verifikation von Anforderungen wird eine integrative Analyse von Anforderungen und Tests unterstützt. Da die Ein- und Ausgangsdaten der TDSS-Methode mit dem Systemmodell vernetzt sind, wird die vollständige und konsistente Spezifikation von Anforderungen und Tests möglich. Auf Basis der definierten Sichten stehen hierzu innerhalb des verwendeten Werkzeugs geeignete Diagramme bereit (u.a. die Testfallspezifikation in Bild 6-16 und die Testzielspezifikation in Bild 6-17).

Die Anwendung zeigt, dass das Konzept der testgetriebenen Entwicklung auf die Systemmodellierung erweitert werden kann. Ausgehend von den Stakeholder-Anforderungen und abgeleiteten Validierungsbedarfen können Validierungsziele definiert und mit den Testszenarien und Testfällen vernetzt werden. Durch die integrative Modellierung mittels TDSS und die anschließende Simulation der Systemanforderungen wird deutlich, dass eine durchgängige Kette von den Elementen des Problemraums, über das Validierungssystem bis hin zu den über Sequenzdiagramme visualisierten Systemanforderungen hergestellt werden kann.

6.3.4 Online-Survey und Experten Feedback

Die beschriebene Anwendung diente zur Demonstration des dritten Artefakts im Kontext des Industrieunternehmens. Für eine weitergehende Evaluierung wurden einerseits Experten (Fachleiter in der Entwicklung) innerhalb des Industrieunternehmens befragt, andererseits wurde das dritte Artefakt im Rahmen der Lehrveranstaltung *Automotive Systems* im internationalen Masterstudiengang *Embedded Systems Engineering*[4] an der Fachhochschule Dortmund in studentischen Projekten angewendet. Die studentischen Projekte wurden durch zwei Industriepartner begleitet. Die Anwendung des MBSE-Frameworks wurde mit dem MBSE-Werkzeug iQUAVIS (s. Kapitel 5.6.1) ermöglicht und durch die *TwoPillars GmbH*[5] unterstützt. Die von den Studenten bearbeiteten Projekte wurden gemeinschaftlich mit der *KOSTAL Automobil Elektrik GmbH & Co. KG*[6] definiert. Zum Abschluss der studentischen Projekte wurden die Erfahrungen über ein Online-Survey erfasst.

---

[4] https://www.fh-dortmund.de/en/programs/embedded-systems-engineering-master.php

[5] https://www.two-pillars.de, zuletzt abgerufen am 31.1.2023

[6] https://www.kostal.com, zuletzt abgerufen am 31.1.2023

Die Expertenbefragung und das Online-Survey basieren auf Aussagen, die von den Befragten zu bewerten waren. Dabei war für jede Aussage auf einer Skala zwischen 1 (keine Zustimmung) bis 4 (volle Zustimmung) eine Auswahl zu treffen. Da ein wesentlicher Bestandteil der dritten Iteration die Anwendung des MBSE-Frameworks war, wurden das Konzept und die Inhalte des Online-Surveys in weiten Teilen von MANDEL et al. [MMA22] übernommen, da diese Arbeit eine geeignete Auswahl an Aussagen zusammenfasst, die eine Bewertung hinsichtlich der individuellen und organisatorischen Akzeptanz eines MBSE-Ansatzes ermöglichen. Zu den existierenden Aussagen wurden lediglich Aussagen hinzugefügt, die im speziellen die testgetriebene und szenariobasierte Vorgehensweise adressieren, da das dritte Artefakt neben dem MBSE-Ansatz die Teillösungen zur Test- und Anforderungsmodellierung beinhaltet. Eine Übersicht der einzelnen Kategorien und den entsprechenden Quellen, aus denen die Kategorien abgeleitet wurden, ist in Tabelle 6-2 dargestellt.

Kategorie (individuelle Akzeptanz)	**abgeleitet von**
Wahrgenommene Leistungsfähigkeit einzelner Nutzer	[DAR+21], [BKD19], [GDS+15], [LAR+14], [MSM+14]
Intuitive Anwendbarkeit	[LAR+14]
Flexibilität und Anpassungsfähigkeit der Methodik	[CB18], [DAR+21], [Fri09], [LAR+14], [GDS+15]
Benutzerfreundlichkeit des Modellierungswerkzeugs	[AZ13], [CB18], [DAR+21], [GDS+15], [LAR+14], [MSM+14]
Zielvorstellung und Modellierungsverfahren für die Nutzer klar erkennbar	[DAR+21], [CB18], [Fri09]
Angemessenes Maß an Formalisierung und Ontologie, um die Kommunikation zwischen den Benutzern zu unterstützen und nicht zu behindern	[GDS+15], [TDB+15], [AZ13]
Kategorie (organisatorische Akzeptanz)	**abgeleitet von**
(Monetäres) Nutzen-Kosten/Aufwand-Verhältnis der Anwendung der Methodik	[BKD19], [CB18], [DAR+21], [GDS+15], [LAR+14]
Lehr- und Lernbarkeit der Methodik	[GDS+15], [LAR+14], [FMS14], [BC10]
Wiederverwendbarkeit und Erweiterbarkeit der Methode und der erstellten Modelle	[LAR+14], [CB18]
Problemorientierung/Unterstützung bei der Modellierung des Problemraums	[Clo19], [TDB+15]

Tabelle 6-2: Kategorien zur Strukturierung des Fragebogens nach [MMA22].

Die Ergebnisse des Expertenbefragung und des Online-Surveys sind im Anhang A2 aufgeführt. Die einzelnen Aussagen sind dort den jeweiligen Kategorien zugeordnet. Die Diagramme beinhalten sowohl das Feedback der Experten, als auch das der Studenten. Die

wesentliche Punkte zu den einzelnen Kategorien sind im Folgenden zusammengefasst.

Sowohl die Experten als auch die Studenten gaben an, dass sie einen Mehrwert durch die Anwendung des MBSE-Frameworks sehen. Nach Meinung der Experten hängt dieser Mehrwert jedoch von der Projektgröße ab. Für kleine Projekte ist der zusätzliche Aufwand für die System- und Anforderungsmodellierung möglicherweise zu groß, obwohl die Anwendbarkeit, Flexibilität und Anpassungsfähigkeit des Ansatzes überwiegend als positiv bewertet wurde (s. Anhang A2). Diese Rückmeldung stimmt mit den Ergebnissen aus der ersten Iteration überein.

Ein wichtiger Punkt für die *individuelle Akzeptanz* des Ansatzes ist, ob die *Zielvorstellung und das Modellierungsverfahren für die Nutzer klar erkennbar* sind [DAR+21; CB18; Fri09]. Die in Bild 6-20 dargestellten Rückmeldungen zeigen, dass der testgetriebene Ansatz hierbei unterstützt. Es wurde bestätigt, dass die testgetriebene Modellierung, die schließlich zu den im Systemmodell vernetzten und über Sequenzdiagramme visualisierten Systemanforderungen führt, die in den jeweiligen Projekten identifizierten Probleme adressieren. Der Ansatz kommt also zu dem gewünschten Ergebnis, was durch die Anwender

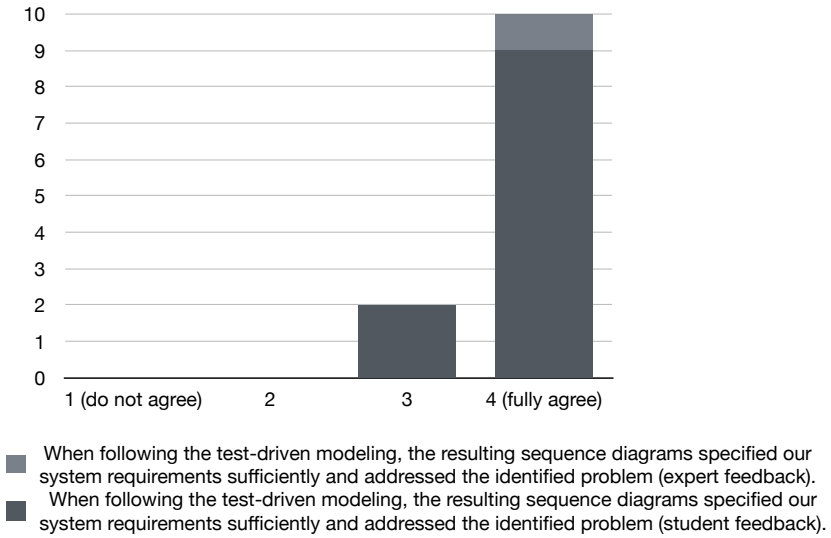


Bild 6-20: Zielvorstellung und Modellierungsverfahren für die Nutzer klar erkennbar

auch so wahrgenommen wurde. Diese Ergebnisse werden durch die in Bild 6-21 dargestellten Rückmeldungen bestätigt. Hier wurde der testgetriebene Ansatz in Kombination mit den vernetzten Elementen (u.a. Validierungsziele) des Validierungssystems betrachtet. Die Rückmeldungen verdeutlichen, dass dadurch eine problemorientierte Modellierung forciert und insbesondere die Entscheidung, welche Funktionalität in welchem Umfang zu modellieren ist, unterstützt wird.

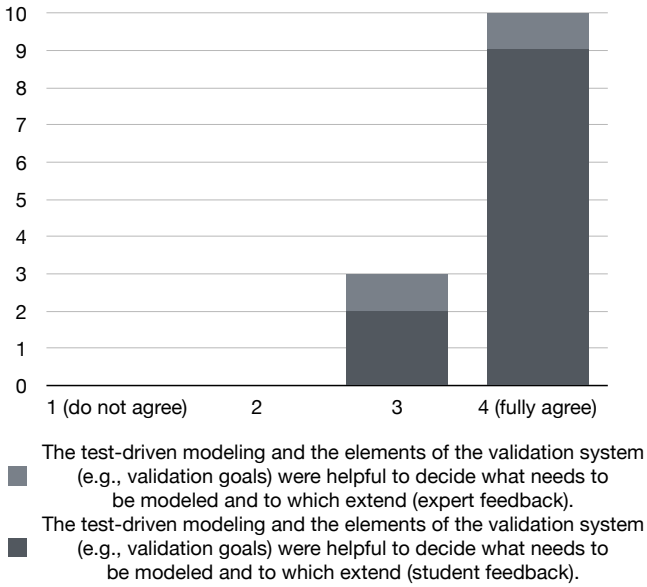


Bild 6-21: Problemorientierung/Unterstützung bei der Modellierung des Problemraums

Ein weiterer wichtiger Punkt ist die szenariobasierte Modellierung, die einerseits für die Systemmodellierung zur Ausgestaltung des Problemraums verwendet wurde, aber auch zur formalen Modellierung der Systemanforderungen. Die Rückmeldungen zeigen, dass dieser szenariobasierte Ansatz insbesondere für die Abstimmung innerhalb der Teams als geeignet gesehen wurde (Bild 6-22).

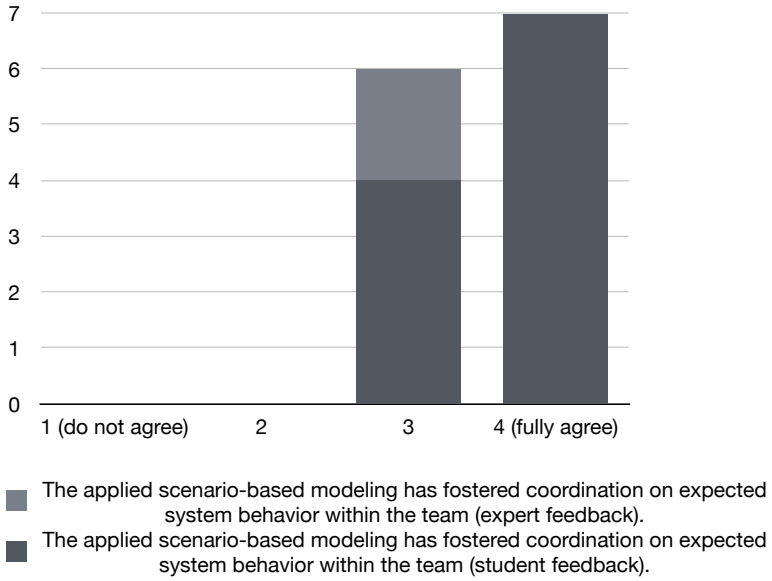


Bild 6-22: Angemessenes Maß an Formalisierung und Ontologie, um die Kommunikation zwischen den Benutzern zu unterstützen und nicht zu behindern

Neben der testgetriebenen und szenariobasierten Modellierung zur Förderung einer ziel- und anwendungsorientierten Modellierung war ein wesentlicher Aspekt der dritten Iteration

die Wiederverwendung von Lösungswissen. Wie in Bild 6-23 kommt die Anwendung des Ansatzes auch zu diesem Punkt zu einem positiven Ergebnis.

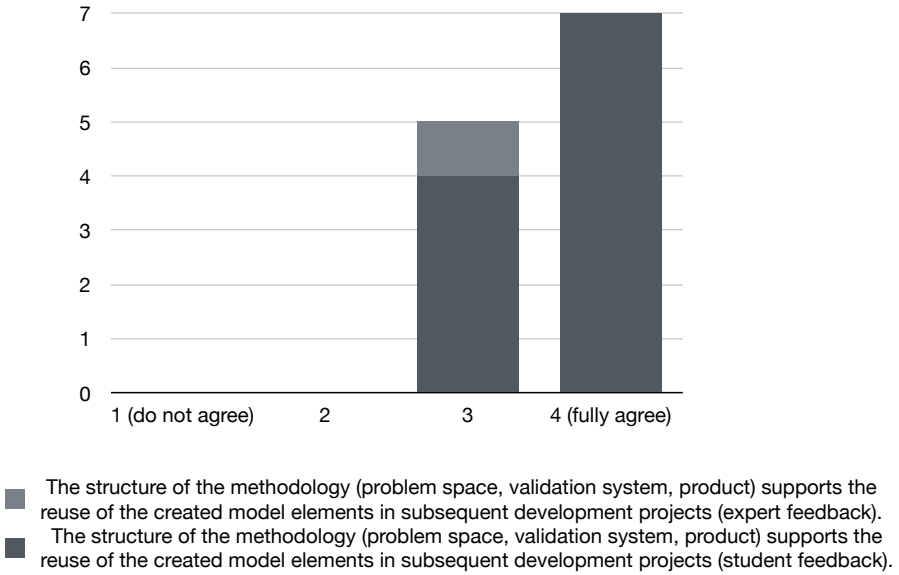


Bild 6-23: Wiederverwendbarkeit und Erweiterbarkeit der Methode und der erstellten Modelle

6.4 Bewertung der Arbeit an den Anforderungen

Die in Kapitel 3.6 beschriebenen Anforderungen an die *Systematik zur integrativen Anforderungsanalyse und Testspezifikation für die Entwicklung von Systemen im Automobilbereich* sind nach der in Kapitel 2 beschriebenen Vorgehensweise die Basis für die Evaluierung der entwickelten Artefakte. Ausgehend von den Ergebnissen der Evaluierungsaktivitäten als Teil der zuvor beschriebenen DSR Iterationen ist im Folgenden aufgelistet, inwieweit die Anforderungen an die Systematik erfüllt werden.

A1) Verständlichkeit: Das Ziel ist, Systemarchitekten und Test-Designer ohne Expertenwissen in der formalen System- und Anforderungsmodellierung zu unterstützen. Damit die Systematik für diese Rollen innerhalb eines Entwicklungsprojekts verständlich ist, wurden die definierten Sichten innerhalb des MBSE-Werkzeugs über Diagramme realisiert, die über den zentralen Einstiegspunkt für die Modellierung (Bild 5-26) niederschwellig anwendbar sind. Ebenso forciert die verwendete szenariobasierte Modellierung die Verständlichkeit. Die Anwendung des MBSE-Ansatzes und der Anforderungsmodellierung in der 3. DSR Iteration bestätigt die Verständlichkeit der Systematik.

A2) Ganzheitliche Betrachtung des Systemkontextes: Als Erweiterung zu etablierten Ansätzen unterstützt die Systematik die System- und Anforderungsmodellierung innerhalb eines ASE Kontextes aus Sicht eines Zulieferer Unternehmens. Dabei ist es entscheidend, die Rolle des zu entwickelnden Systems in einem Systemverbund zu verstehen, um die

für die Entwicklung und Validierung notwendigen Informationen systematisch zu erfassen und kontinuierlich mit den Stakeholdern abstimmen zu können. Das wird unterstützt durch die in Phase 1 beschriebenen Aktivitäten zur Modellierung des Problemraums. Zusätzlich ermöglichen die in Kapitel 5.4.4 beschriebenen Erweiterungen der Anforderungsmodellierung eine Anforderungsanalyse, die sowohl die Systeminteraktionen eines übergeordneten Anwendungsfalls eines *Advanced Systems* als auch das konkrete Verhalten des zu entwickelnden Teilsystems berücksichtigt.

A3) Orientierung an etablierten Standards in der Steuergeräteentwicklung: Durch die Resultate des in Kapitel 5.3 beschriebenen Vorgehensmodells wird die Integration der Systematik in bestehende Prozesse für die Steuergeräteentwicklung möglich. Ein zentrales Resultat der Phase 1 sind strukturierte Stakeholder-Anforderungen als Ausgangspunkt der Systementwicklung. Diese werden über die Validierungssystemmodellierung und die testgetriebene Anforderungsmodellierung systematisch und modellbasiert in Systemanforderungen und Testspezifikationen überführt. Als Ergebnis können so in Phase 4 Anforderungs- und Testspezifikationen aus dem Systemmodell ausgeleitet werden, wie sie u.a. durch den ASPICE Standard eingefordert werden. Ein Beispiel ist die in Bild 6-16 dargestellte Testfallspezifikation.

A4) Unterstützung von funktionalen Anforderungen in natürlicher Sprache: In der Problemanalyse wurde deutlich, dass Anforderungen zwischen Entwicklungspartnern überwiegend in unstrukturierter natürlicher Sprache ausgetauscht werden und somit die Grundlage für die Koordinierung der Entwicklungstätigkeiten sind. Über geeignete Sichten und deren Umsetzung über geeignete Diagramme können diese textuellen Anforderungen in das Systemmodell übertragen und vernetzt werden. Zudem wird die weitere Modellierung und Analyse der Anforderungen umfassend methodisch und werkzeugtechnisch unterstützt. Ein wesentlicher Aspekt ist die Kombination der Techniken zur Generierung der Testfälle, die wiederum die Modellierung und automatisierte Analyse funktionaler Systemanforderungen treiben. Die Ergebnisse aus der ersten DSR Iteration zeigen, dass auf diese Weise Widersprüche in funktionalen Anforderungen automatisiert identifiziert werden können. Die Kombination der Techniken und der resultierende Mehrwert für die Konsolidierung und Dokumentation von Anforderungen wird in der zweiten DSR Iteration deutlich.

A5) Unterstützung einer iterativen Spezifikation von Anforderungen: Die TDSS-Methode (Kapitel 5.4.1) und die verwendete szenariobasierte Modellierungssprache SMLK (Kapitel 4.2.4) adressieren von Grund auf eine iterative Anforderungsspezifikation. Ziel der TDSS-Methode ist, dass Systemarchitekten und Test-Designer durch die Ausführung von Testfällen in kurzen Iterationen bei der kollaborativen Modellierung unterstützt werden. Die Ergebnisse der ersten DSR Iteration zeigen, dass die kleinschrittige Modellierung erfolgreich eingesetzt werden kann. Zudem steht die TDSS-Methode im Zentrum einer iterativen Systemmodellierung. Durch die in Bild 5-9 dargestellte Methode wird ein stark iteratives und problemorientiertes Vorgehen adressiert. Die Ergebnisse der dritten DSR Iteration dokumentiert die erfolgreiche Anwendung.

A6) Unterstützung einer iterativen Spezifikation von Tests: Dem TDM Ansatz folgend basiert die TDSS-Methode auf einer iterativen Spezifikation von Testfällen, die die Anforderungsmodellierung treiben. Somit ist die TDSS-Methode die Basis für eine integrative und iterative Test- und Anforderungsspezifikation. Durch die Validierungssystemmodellierung (I2 in Bild 5-9) sind die Testfälle mit den Elementen des Validierungssystems verzahnt und ermöglichen so ausgehend von der TDSS-Methode eine iterative Analyse und Spezifikation von Tests. Durch die verwendete Technik zur Testfallgenerierung kann die Qualität der Eingangsdaten für die TDSS-Methode sichergestellt werden. Hierbei werden Testfälle aus einzelnen Stakeholder-Anforderungen generiert, die dann wiederum iterativ zum Systemmodell hinzugefügt und vernetzt werden. Die Unterstützung einer iterativen Spezifikation von Tests ist über die zweite und dritte DSR Iteration erfolgreich evaluiert.

A7) Unterstützung einer frühen Validierung von Anforderungen Die Systematik basiert auf dem Konzept einer frühen und kontinuierlichen Validierung (s. Kapitel 4.1.3). Über die beschriebene Ontologie und die daraus abgeleiteten Standpunkte und Sichten auf das Systemmodell können Validierungsbedarfe frühzeitig festgelegt werden. Über die Unterstützung zur Modellierung des Problemraums und die enge Verzahnung der Elemente des Problemraums mit Validierungsbedarfen entsteht eine für die Validierung wichtige Fokussierung auf die Bedarfe einzelner Stakeholder. Damit sind die vernetzten Validierungsbedarfe der Ausgangspunkt für eine frühe Validierung. Da die weitergehende Validierungssystemmodellierung dem TDM Ansatz folgt und der Anforderungsmodellierung vorangestellt ist, wird die Validierung von Anforderungen über die Ausführung der TDSS-Methode zu einem frühestmöglichen Zeitpunkt adressiert. Die Validierungsbedarfe sind über die Ontologie mit den Testszenarien und Testfällen vernetzt, die die Anforderungsmodellierung treiben. Somit wird die Validierung im Gegensatz zu etablierten Ansätzen nicht als abschließende sondern initiierende Tätigkeit verstanden. Die dritte DSR Iteration dokumentiert die erfolgreiche Anwendung.

A8) Wiederverwenden von Lösungswissen Die Technik zur kontinuierlichen Validierung auf Basis einer testgetriebenen Anforderungsmodellierung ist die Basis für die systematische Externalisierung und Wiederverwendung von Lösungswissen. Die Ontologie ermöglicht die Beschreibung von Lösungsmustern in Form von Problem-Lösungspaaren. Die Kombination der Modellierung auf System- und Anforderungsebene ermöglicht die kontinuierliche Wissensaufbereitung über den digitalen Musterkatalog. Die Umsetzung des Musterkatalogs in dem MBSE-Werkzeug ist in Bild 5-27 verdeutlicht.

A9) Werkzeugunterstützung Die Werkzeugunterstützung ist in zwei Bereiche unterteilt. Zum einen ist das entwickelte MBSE-Framework über das Werkzeug iQUAVIS umgesetzt und zum anderen ist ein integriertes Werkzeug für die testgetriebene Anforderungsmodellierung beschrieben. Die Evaluierung der Systematik ist über die Anwendung der Werkzeuge in allen drei DSR Iterationen beschrieben. Bei der werkzeugtechnischen Umsetzung wurde insbesondere die Verständlichkeit (A1) adressiert. Die Ergebnisse der dritten DSR Iteration dokumentieren die erfolgreiche Anwendung der Werkzeuge in studentischen Projekten und innerhalb des Industrieunternehmens. Einerseits gelingt somit die Anwendung der

Systematik ohne Expertenwissen in der formalen System- und Anforderungsmodellierung, andererseits können reale Projektdaten verarbeitet werden.

7 Zusammenfassung

Zukünftige Systeme im Automobilbereich können als *Advanced Systems* charakterisiert werden. Diese zeichnen sich durch eine dynamische Vernetzung in komplexen Systemverbünden aus. Die durch den Kunden erlebbaren Funktionalitäten resultieren aus dem Zusammenspiel unterschiedlicher Systeme, die teils unabhängig voneinander in unterschiedlichen Entwicklungsnetzwerken entwickelt werden. Das ASE stellt den Rahmen für die Entwicklung dieser Systeme bereit. Es verbindet die Handlungsfelder *Advanced Systems*, *Systems Engineering* und *Advanced Engineering*.

Für die Unternehmen in der Automobilindustrie entstehen durch die Kollaboration mit neuen und heterogenen Entwicklungspartnern vielzählige Nutzenpotentiale. Gleichzeitig entstehen dabei jedoch durch die historisch gewachsenen OEM-Zulieferer-Strukturen und die starken regulatorischen Vorgaben Herausforderungen für die beteiligten Unternehmen. Dabei haben Anforderungs- und Testspezifikationen eine herausragende Bedeutung, da diese für die Abstimmungen zwischen den an der Systementwicklung beteiligten Unternehmen verwendet werden und die Basis für die erfolgreiche Systementwicklung sind. In diesem Kontext adressiert die vorliegende Arbeit die Entwicklung zukünftiger Systeme im Automobilbereich über die *Systematik zur integrativen Anforderungsanalyse und Testspezifikation*.

Die Problemanalyse kommt zu dem Ergebnis, dass Anforderungen und Testfälle überwiegend in unstrukturierter natürlicher Sprache spezifiziert werden. Hieraus resultiert, dass die Spezifikationen zwar niederschwellig erstellt und leicht zwischen den Entwicklungspartnern ausgetauscht werden können, durch die hohen Freiheitsgrade gelingt es jedoch kaum, Spezifikationen reproduzierbar in hoher Qualität zu erstellen. Um das zu adressieren, existieren zwar formale modellbasierte Ansätze, jedoch ist es herausfordernd, ein geeignetes Maß an Formalisierung und niederschwelliger Anwendbarkeit zu finden. In der Automobilindustrie kommen formale modellbasierte Techniken innerhalb der Anforderungsanalyse und der Testfallspezifikation kaum zum Einsatz.

Um Systemarchitekten und Test-Designer bei den Spezifikations- und Analysetätigkeiten zu unterstützen, beschreibt die Systematik einen umfassenden modellbasierten und anwendungsorientierten Ansatz. Ausgangspunkt ist die innerhalb des MoSyS Projekts entstandene Referenzarchitektur für die integrative Produkt- und Validierungssystementwicklung. Über die Ontologie und die daraus abgeleiteten aufgabenspezifischen Sichten auf ein zentrales Systemmodell werden Systemarchitekten und Test-Designer bei den Modellierungsaktivitäten unterstützt. Die Aktivitäten sind über ein Vorgehensmodell und eine Methode zur integrativen Produkt- und Validierungssystemmodellierung strukturiert. Im Zentrum steht dabei die TDSS-Methode, die eine testgetriebene formale Anforderungsmodellierung ermöglicht. Das resultierende Anforderungsmodell erlaubt die automatisierte Analyse und zielgerichtete Dokumentation der Systemanforderungen. Zusätzlich werden Systemarchitekten und Test-Designer durch den digitalen Musterkatalog unterstützt, wobei

die Definition und Anwendung neuer Lösungsmuster ebenfalls methodisch beschrieben ist und so die systematische Aufbereitung und Wiederverwendung von Lösungswissen für die Anforderungsanalyse und Testspezifikation fördert. Schließlich können so qualitativ hochwertige Spezifikationen aus dem Systemmodell ausgeleitet werden.

Die Systematik wurde über eine iterative Vorgehensweise auf Basis des DSR Paradigmas erarbeitet. Die einzelnen DSR Iterationen sind als Teil der Evaluierung beschrieben. Auf Basis von drei realen Funktion aus dem Bereich der Steuergeräteentwicklung konnten die Zwischenergebnisse erarbeitet und evaluiert werden. Zusätzlich zu der kontinuierlichen Einbindung des Industriepartners wurde die Systematik im Rahmen einer Lehrveranstaltung in studentischen Projekten angewendet und darüber evaluiert.

Die Evaluierung macht deutlich, dass das in dieser Arbeit beschriebene iterative modellbasierte Vorgehen mit den zugehörigen Methoden und der Werkzeugunterstützung eine formale aber dennoch intuitive Erstellung des System- und Anforderungsmodells ermöglicht. Der digitale Musterkatalog verwendet beide Modelle für die Wissensaufbereitung. Im Gegensatz zu den in der Industrie etablierten Copy-Paste-Tailoring-Ansätzen ist durch den musterbasierten Ansatz ein systematischeres Vorgehen möglich. Ziel zukünftiger Arbeiten sollte die Anwendung und weitere Evaluierung des Musterkatalogs sein.

Abkürzungsverzeichnis

DSR Design Science Research

SE Systems Engineering

RE Requirements Engineering

V&V Verifikation und Validierung

MBSE Model-Based Systems Engineering

ASE *Advanced Systems Engineering*

OEM Original Equipment Manufacterer

UML Unified Modeling Language

MSC Message Sequence Chart

LSC Live Sequence Chart

SML Scenario Modeling Language

SMLK Scenario Modeling Language for Kotlin

MSD Modal Sequene Diagram

OBC On-Board Charger

TDSS Test-Driven Scenario Specification

MoSyS Menschorientierte Gestaltung komplexer System of Systems

MSC Message Sequence Chart

SECI Socialisation Externalization Combination Internalization

HLM High-Level Modeling

LLM Low-Level Modeling

DE Domänenexperten

ME Modellersteller

EMF Eclipse Modeling Framework

DSL Domain Specific Language

PoC Point of Control

PoO Point of Observation

ACC Adaptive Cruise Control

ECS Electronically Controled Steering

CAN Controller Area Network

CEG Cause Effect Graph

NLP Natural Language Processing

MBT Model-Based Testing

ESC Electronic Stability Control

CUT Cluster under Test

GV Gestaltvariation

PV Prinzipvariation

ÜV Übernahmevariation

TDD Test-Driven Development

TDM Test-Driven Modeling

BDD Behavior-Driven Development

SoS System of Systems

CS Constituend System

CiRA *Causality in Requirement Artifacts*

HMI Human Machine Interface

SOC State of Charge

Literaturverzeichnis

Publikationen

[ABK+16] ALBERS, A.; BEHRENDT, M.; KLINGLER, S.; MATROS, K.: Verifikation und Validierung im Produktentstehungsprozess. In: Udo Lindemann: Handbuch Produktentwicklung. Carl Hanser Verlag, 2016, S. 541–569

[ABK+17] ALBERS, A.; BEHRENDT, M.; KLINGLER, S.; REISS, N.; BURSAC, N.: Agile product engineering through continuous validation in PGE - Product Generation Engineering. 3, Cambridge University Press (CUP), 2017

[ABN+06] ALMEFELT, L.; BERGLUND, F.; NILSSON, P.; MALMQVIST, J.: Requirements management in practice: findings from an empirical study in the automotive industry. 17, Springer Verlag, 2006, S. 113–134

[ABR17] ALBERS, A.; BURSAC, N.; RAPP, S.: PGE – Produktgenerationsentwicklung am Beispiel des Zweimassenschwungrads. 81, Springer Verlag, 2017, S. 13–31

[ABW15] ALBERS, A.; BURSAC, N.; WINTERGERST, E.: Produktgenerationsentwicklung – Bedeutung und Herausforderungen aus einer entwicklungsmethodischen Perspektive, Fraunhofer Verlag, 2015

[ADK+20] ANACKER, H.; DUMITRESCU, R.; KHARATYAN, A.; LIPSMEIER, A.: Pattern based Systems Engineering– Application of Solution Patterns in the Design of Intelligent Technical Systems, Cambridge University Press (CUP), 2020, S. 1195–1204

[AHH+19] ALBERS, A.; HAUG, F.; HEITGER, N.; FAHL, J.; HIRSCHTER, T.: Entwicklungsgenerationen zur Steuerung der PGE – Produktgenerationsentwicklung: Von der Bauteil-zur Funktionsorientierung in der Automobilentwicklung. In: Stuttgarter Symposium für Produktentwicklung SSP 2019, Fraunhofer-Institut für Arbeitswirtschaft und Organisation IAO, 2019, S. 253–262

[AIS77] ALEXANDER, C.; ISHIKAWA, S.; SILVERSTEIN, M.: A Pattern Language - Towns Buildings Constructions. Oxford University Press, 1977

[AL12] ALBERS, A.; LOHMEYER, Q.: Advanced systems engineering - Towards a model-based and human-centered methodology. In: Tools and methods of competitive engineering: Proceedings of the ninth International Symposium on Tools and Methods of Competitive Engineering (TMCE 2012), Karlsruhe, Germany, May 7-11, 2012. Ed.: I. Horváth, Delft University of Technology (TU Delft), 2012, S. 407–416

[ALE11] ALBERS, A.; LOHMEYER, Q.; EBEL, B.: Dimensions of objectives in interdisciplinary product development projects. In: ICED 11 - 18th International Conference on Engineering Design - Impacting Society Through Engineering Design, Design Society, 2011, S. 256–265

[Ale64] ALEXANDER, C.: Notes on the Synthesis of Form. Harvard University Press, 1964

[Ale79] ALEXANDER, C.: The Timeless Way of Building. Oxford University Press, 1979

[AMB+15] ALBERS, A.; MATROS, K.; BEHRENDT, M.; JETZINGER, H.: Das Pull-Prinzip der Validierung - Ein Referenzmodell zur effizienten Integration von Validierungsaktivitäten in den Produktentstehungsprozess. 67, VDI Fachmedien, 2015, S. 74–81

[AMY+18] ALBERS, A.; MANDEL, C.; YAN, S.; BEHRENDT, M.: System of systems approach for the description and characterization of validation environments. In: Proceedings of International Design Conference, DESIGN, 2018, S. 2799–2810

[Ana15] ANACKER, H.: Instrumentarium für einen lösungsmusterbasierten Entwurf fortgeschrittener mechatronischer Systeme, Dissertation, Universität Paderborn, 2015

[ARB+17] ALBERS, A.; RAPP, S.; BIRK, C.; BURSAC, N.: Die Frühe Phase der PGE - Produktgenerationsentwicklung. In: 4. Stuttgarter Symposium für Produktentwicklung 2017 (SSP) : Produktentwicklung im disruptiven Umfeld, Stuttgart, Deutschland, 28-29 Juni 2017, Fraunhofer Verlag, 2017

[AZ13] ALBERS, A.; ZINGEL, C.: Challenges of model-based systems engineering: A study towards unified term understanding and the state of usage of SysML. In: Smart product engineering. Springer, 2013, S. 83–92

[Bal09] BALZERT, H.: Lehrbuch der Softwaretechnik - Basiskonzepte und Requirements Engineering. Springer, 2009

[BBR21] BROY, M.; BÖHM, W.; RUMPE, B.: Model-Based Engineering of Collaborative Embedded Systems. In: Model-Based Engineering of Collaborative Embedded Systems. Springer, 2021, – ISBN: 9783030621360

[BC10] BONE, M.; CLOUTIER, R.: The current state of model based systems engineering: Results from the omg sysml request for information 2009. In: Proceedings of the 8th conference on systems engineering research, 2010

[Bec03] BECK, K.: Test-driven development: by example. Addison-Wesley, 2003

[BEF11] BARMI, Z. A.; EBRAHIMI, A. H.; FELDT, R.: Alignment of requirements specification and testing: A systematic mapping study. In: Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011, IEEE, 2011, S. 476–485

[BGH+14] BRENNER, C.; GREENYER, J.; HOLTMANN, J.; LIEBEL, G.; STIEGLBAUER, G.; TICHY, M.: ScenarioTools Real-Time Play-Out for Test Sequence Validation in an Automotive Case Study. In: Electronic Communications of the EASST, EASST, 2014

[BGP+13] BRENNER, C.; GREENYER, J.; PANZICA, V.; MANNA, L.: The ScenarioTools Play-Out of Modal Sequence Diagram Specifications with Environment Assumptions The ScenarioTools Pla. In: Electronic Communications of the EASST 12th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2013), 2013

[Bit01] BITSCH, F.: Safety Patterns — The Key to Formal Specification of Safety Requirements. In: Voges, U. (Hrsg.), Computer Safety, Reliability and Security, Springer Berlin Heidelberg, 2001, S. 176–189

[BK04] BERRY, D.; KAMSTIES, E.: Ambiguity in Requirements Specification, Springer, 2004

[BKD19] BRETZ, L.; KAISER, L.; DUMITRESCU, R.: An analysis of barriers for the introduction of Systems Engineering. 84, Elsevier, 2019, S. 783–789

[BRB+13] BJARNASON, E.; RUNESON, P.; BORG, M.; UNTERKALMSTEINER, M.; ENGSTRÖM, E.; REGNELL, B.; SABALIAUSKAITE, G.: Challenges and practices in aligning requirements with verification and validation: A case study of six companies. 19, Springer, 2013

[Bre21] BRETZ, L. H.: Rahmenwerk zur Planung und Einführung von Systems Engineering und Model-Based Systems Engineering - Framework for the introduction of systems engineering and model-based systems engineering, Dissertation, Universität Paderborn, 2021, DOI: 10.17619/UNIPB/1-1225, Unter: https://nbn-resolving.org/urn:nbn:de:hbz:466:2-39848

[Bro06] BROY, M.: Challenges in automotive software engineering. In: Proceedings - International Conference on Software Engineering, ACM, 2006, S. 33–42

[CA09] CHENG, B. H. C.; ATLEE, J. M.: Research directions in requirements engineering. In: Lecture Notes in Business Information Processing, IEEE, 2009, S. 11–43

[CB18] CHAMI, M.; BRUEL, J.-M.: A survey on MBSE adoption challenges, Wiley Interscience Publications, 2018

[CHQ+16] CZIHARZ, T.; HRUSCHKA, P.; QUEINS, S.; WEYER, T.: Handbook of Requirements Modeling According to the IREB Standard, 2016, Unter: https://www.ireb.org/en/downloads/tag:handbook

[Clo06] CLOUTIER, R. J.: Applicability of patterns to architecting complex systems, Dissertation, Stevens Institute of Technology, 2006

[Clo19] CLOUTIER, R.: 2018 MBSE survey results. In: Proceedings of the 2019 INCOSE MBSE Workshop, presented at the INCOSE 2019 International Workshop, 2019

[CSG15] CHANDRA, L.; SEIDEL, S.; GREGOR, S.: Prescriptive Knowledge in Is Research: Conceptualizing Design Principles in Terms of Materiality, Action, and Boundary Conditions. In: 2015 48th Hawaii International Conference on System Sciences, IEEE, 2015, S. 4039–4048

[CV06] CLOUTIER, R.; VERMA, D.: Applying Pattern Concepts to Enterprise Architecture. In: Journal of Enterprise Architecture, Jan. 2006

[DAC99] DWYER, M. B.; AVRUNIN, G. S.; CORBETT, J. C.: Patterns in Property Specifications for Finite-State Verification. In: Proceedings of the 21st International Conference on Software Engineering, Association for Computing Machinery, 1999, S. 411–420

[DAR+21] DUMITRESCU, R.; ALBERS, A.; RIEDEL, O.; STARK, R.; GAUSEMEIER (HRSG.), J.: Engineering in Deutschland Status quo in Wirtschaft und Wissenschaft Ein Beitrag zum Advanced Systems Engineering, 2021

[DB08] DAHMANN, J. S.; BALDWIN, K. J.: Understanding the Current State of US Defense Systems of Systems and the Implications for Systems Engineering. In: 2008 2nd Annual IEEE Systems Conference, 2008, S. 1–7

[DFH16] DALPIAZ, F.; FRANCH, X.; HORKO, J.: iStar 2 . 0 Language Guide, 2016

[DH01] DAMM, W.; HAREL, D.: LSCs: Breathing Life into Message Sequence Charts. In: Formal Methods in System Design, Springer, 2001, S. 45–80

[DHK+17] DE OLIVEIRA NETO, F. G.; HORKOFF, J.; KNAUSS, E.; KASAULI, R.; LIEBEL, G.: Challenges of aligning requirements engineering and system testing in large-scale agile: A multiple case study, 2017, DOI: 10.1109/REW.2017.33

[DRF+15] DAVID D. WALDEN; ROEDLER, G. J.; FORSBERG, K. J.; HAMELIN, R. D.; SHORTELL, T. M.: INCOSE systems engineering handbook: a guide for system life cycle processes and activities, 2015

[DSV+07] DIAS NETO, A. C.; SUBRAMANYAN, R.; VIEIRA, M.; TRAVASSOS, G. H.: A survey on model-based testing approaches: A systematic review. In: Proc. - 1st ACM Int. Workshop on Empirical Assessment of Software Engineering Languages and Technologies, WEASELTech 2007, Held with the 22nd IEEE/ACM Int. Conf. Automated Software Eng., ASE 2007, 2007, S. 31–36

[Dud22] DUDEN: https://www.duden.de, 2022, Unter: https://www.duden.de/node/178535/revision/887957

[Dum10] DUMITRESCU, R.: Entwicklungssystematik zur Integration kognitiver Funktionen in fortgeschrittene mechatronische Systeme, Dissertation, Universität Paderborn, 2010

[Ebe15] EBEL, B.: Modellierung von Zielsystemen in der interdisziplinären Produktentstehung Modeling of System of Objectives in Interdisciplinary Product Engineering, Dissertation, Karlsruher Institut für Technologie, 2015

[Elm73] ELMENDORF, W. R.: Cause-effect Graphs in Functional Testing. IBM, 1973

[EM17] EHRLENSPIEL, K.; MEERKAMM, H.: Integrierte Produktentwicklung, Denkabläufe, Methodeneinsatz, Zusammenarbeit. Hanser Verlag, 2017

[Est+07] ESTEFAN, J. A. u. a.: Survey of model-based systems engineering (MBSE) methodologies. 25, Citeseer, 2007, S. 1–12

[FDN+22] FRIEDENTHAL, S.; DAVEY, C.; NIELSEN, P.; SCHREINEMAKERS, P.; OSTER, C.; SPARKS, E.; MATTHEWS, S.; RIETHLE, T.; STOEWER, H.; NICHOLS, D.; ROEDLER, G.: Systems Engineering Vision 2035 Engineering Solutions for a Better World INCOSE, 2022

[FFS+21] FISCHBACH, J.; FRATTINI, J.; SPAANS, A.; KUMMETH, M.; VOGELSANG, A.; MÉNDEZ, D.; UNTERKALMSTEINER, M.: Automatic Detection of Causality in Requirement Artifacts: The CiRA Approach. In: Dalpiaz, F.; Spoletini, P. (Hrsg.), Requirements Engineering: Foundation for Software Quality - 27th International Working Conference, REFSQ 2021, Essen, Germany, April 12-15, 2021, Proceedings, Springer, 2021, S. 19–36

[FFV+23] FISCHBACH, J.; FRATTINI, J.; VOGELSANG, A.; MÉNDEZ, D.; UNTERKALMSTEINER, M.; WEHRLE, A.; HENAO, P. R.; YOUSEFI, P.; JURICIC, T.; RADDUENZ, J.; WIECHER, C.: Automatic Creation of Acceptance Tests by Extracting Conditionals from Requirements: NLP Approach and Case Study. In: Journal of Systems and Software, 2023

[FG13] FELDHUSEN, J.; GROTE, K.-H.: Der Produktentstehungsprozess (PEP), in: Pahl/Beitz Konstruktionslehre: Methoden und Anwendung erfolgreicher Produktentwicklung, Feldhusen, J.; Grote, K.-H., Springer Berlin Heidelberg, 2013, S. 11–24, – ISBN: 978-3-642-29569-0, DOI: 10.1007/978-3-642-29569-0_2, Unter: https://doi.org/10.1007/978-3-642-29569-0_2

[FH14] FOCKEL, M.; HOLTMANN, J.: A requirements engineering methodology combining models and controlled natural language. In: 2014 IEEE 4th International Model-Driven Requirements Engineering Workshop (MoDRE), 2014, S. 67–76

[FH15] FOCKEL, M.; HOLTMANN, J.: ReqPat: Efficient documentation of high-quality requirements using controlled natural language. In: Zowghi, D.; Gervasi, V.; Amyot, D. (Hrsg.), 23rd IEEE International Requirements Engineering Conference, RE 2015, Ottawa, ON, Canada, August 24-28, 2015, IEEE Computer Society, 2015, S. 280–281

[FHK+16] FOCKEL, M.; HOLTMANN, J.; KOCH, T.; SCHMELTER, D.: Formal , Model- and Scenario-based Requirement Patterns. In: 6th International Conference on Model-Driven Engineering and Software Development, ACM, 2016

[Fis22] FISCHBACH, J.: Why and How to Extract Conditional Statements From Natural Language Requirements, Dissertation, Universität zu Köln, 2022

[FMJ+14] FEMMER, H.; MÉNDEZ FERNÁNDEZ, D.; JÜRGENS, E.; KLOSE, M.; ZIMMER, I.; ZIMMER, J.: Rapid Requirements Checks with Requirements Smells: Two Case Studies. In: ACM, 2014

[FMS14] FRIEDENTHAL, S.; MOORE, A.; STEINER, R.: A practical guide to SysML: the systems modeling language. Morgan Kaufmann, 2014

[FMV+22] FRANCH, X.; MÉNDEZ, D.; VOGELSANG, A.; HELDAL, R.; KNAUSS, E.; ORIOL, M.; TRAVASSOS, G. H.; CARVER, J. C.; ZIMMERMANN, T.: How do Practitioners Perceive the Relevance of Requirements Engineering Research? 48, IEEE, 2022, S. 1947–1964

[Foc18] FOCKEL, M.: Safety Requirements Engineering for Early SIL Tailoring, Dissertation, Universität Paderborn, 2018, DOI: 10.17619/UNIPB/1-490

[FPQ20] FRANCH, X.; PALOMARES, C.; QUER, C.: Industrial Practices on Requirements Reuse: An Interview-Based Study, in: International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer, 2020, S. 78–94, – ISBN: 978-3-030-44428-0, DOI: 10.1007/978-3-030-44429-7_6

[Fri09] FRIEDENTHAL, S.: SysML: Lessons from early applications and future directions. 12, Wiley Online Library, 2009, S. 10–12

[FV19] FEMMER, H.; VOGELSANG, A.: Requirements Quality Is Quality in Use. 36, IEEE, 2019, S. 83–91

[FWK+17] FERNÁNDEZ, D. M.; WAGNER, S.; KALINOWSKI, M.; FELDERER, M.; MAFRA, P.; VETRÒ, A.; CONTE, T.; CHRISTIANSSON, M.-T.; GREER, D.; LASSENIUS, C.; MÄNNISTÖ, T.; NAYABI, M.; OIVO, M.; PENZENSTADLER, B.; PFAHL, D.; PRIKLADNICKI, R.; RUHE, G.; SCHEKELMANN, A.; SEN, S.; SPINOLA, R.; TUZCU, A.; VARA, J. L. de la; WIERINGA, R.: Naming the pain in requirements engineering. 22, Springer, 2017, S. 2298–2338

[GCW+13] GAUSEMEIER, J.; CZAJA, A.; WIEDERKEHR, O.; DUMITRESCU, R.; TSCHIRNER, C.; STEFFEN, D.: Studie: Systems Engineering in der industriellen Praxis, 2013, S. 113–122

[GDE+18] GAUSEMEIER, J.; DUMITRESCU, R.; ECHTERFELD, J.; PFÄNDER, T.; STEFFEN, D.; THIELEMANN, F.: Innovationen für die Märkte von morgen: Strategische Planung von Produkten, Dienstleistungen und Geschäftsmodellen, 2018 – ISBN: 978-3-446-42824-9

[GDS+15] GAUSEMEIER, J.; DUMITRESCU, R.; STEFFEN, D.; CZAJA, A.; WIEDERKEHR, O.; TSCHIRNER, C.: Systems Engineering in industrial practice, 2015

[GFD+09] GAUSEMEIER, J.; FRANK, U.; DONOTH, J.; KAHL, S.: Specification technique for the description of self-optimizing mechatronic systems. In: Research in Engineering Design, 2009, S. 201

[GFK+17] GAROUSI, V.; FELDERER, M.; KUHRMANN, M.; HERKILOUGLU, K.: What Industry Wants from Academia in Software Testing? Hearing Practitioners Opinions. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, Association for Computing Machinery, 2017, S. 65–69

[GGG+17] GREENYER, J.; GRITZNER, D.; GUTJAHR, T.; KÖNIG, F.; GLADE, N.; MARRON, A.; KATZ, G.: ScenarioTools – A tool suite for the scenario-based modeling and analysis of reactive systems. In: Science of Computer Programming, Elsevier, 2017, S. 15–27

[GGK+16] GREENYER, J.; GRITZNER, D.; KATZ, G.; MARRON, A.: reactive systems, dynamic system structure, scenario-based speci - cation, graph transformation, analysis, speci cation inconsistency, realizability, controller synthesis, Scenario Modeling Language, Live Sequence Charts. In: Proceedings of the MoDELS 2016 Demo and Poster Sessions, co-located with ACM/IEEE

19th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2016), 2016

[GH13] GREGOR, S.; HEVNER, A. R.: POSITIONING AND PRESENTING DESIGN SCIENCE Types of Knowledge in Design Science Research. In: MIS Quarterly, 2013, S. 337–355

[GHJ+94] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Pearson Education, 1994 – ISBN: 9780321700698

[GHM+15] GREENYER, J.; HAASE, M.; MARHENKE, J.; BELLMER, R.: Evaluating a formal scenario-based method for the requirements analysis in automotive software engineering. In: 2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015 - Proceedings, ACM, 2015, S. 1002–1005

[GJ07] GREGOR, S.; JONES, D.: The anatomy of a design theory. In: Journal of the Association for Information Systems, 2007, S. 312–335

[Gre11] GREENYER, J.: Scenario-based Design of Mechatronic Systems, Dissertation, Universität Paderborn, 2011

[Gre21] GREENYER, J.: Scenario-Based Modeling and Programming of Distributed Systems. In: Proceedings of the International Workshop on Petri Nets and Software Engineering, CEUR-WS.org, 2021, S. 241–252

[Hab12] HABERFELLNER, R.: Systems Engineering: Grundlagen und Anwendung. Orell Füssli, 2012 – ISBN: 9783280040683

[Har87] HAREL, D.: Statecharts: a visual formalism for complex systems. In: Science of Computer Programming, 1987, S. 231–274

[HGB08] HASLING, B.; GOETZ, H.; BEETZ, K.: Model based testing of system requirements using UML use case models. In: Proceedings of the 1st International Conference on Software Testing, Verification and Validation, ICST 2008, 2008, S. 367–376

[HH03] HEUMESSER, N.; HOUDEK, F.: Towards Systematic Recycling of System Requirements. In: Proceedings - International Conference on Software Engineering, IEEE, 2003, S. 512–519

[Hil12] HILLENBRAND, M.: Funktionale Sicherheit nach ISO 26262 in der Konzeptphase der Entwicklung von Elektrik/Elektronik Architekturen von Fahrzeugen, 2012 – ISBN: 978-3-86644-803-2

[HJE+13] HAUPTMANN, B.; JUNKER, M.; EDER, S.; HEINEMANN, L.; VAAS, R.; BRAUN, P.: Hunting for smells in natural language tests. In: 2013 35th International Conference on Software Engineering (ICSE), 2013, S. 1217–1220

[HM03a] HAREL, D.; MARELLY, R.: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer, 2003

[HM03b] HAREL, D.; MARELLY, R.: Specifying and Executing Behavioral Requirements: The Play-In/Play-Out Approach. In: Software and Systems Modeling (SoSyM), 2003, S. 82–107

[HM08] HAREL, D.; MAOZ, S.: Assert and negate revisited: Modal semantics for UML sequence diagrams. In: Software and Systems Modeling (SoSyM), Springer, 2008, S. 237–252

[HM12] HAREL, D.; MARRON, A.: The quest for runware: On compositional, executable and intuitive models. In: Software and Systems Modeling, Springer, 2012, S. 599–608

[HMM+21] HAREL, D.; MARELLY, R.; MARRON, A.; SZEKELY, S.: Integrating Interobject Scenarios with Intraobject Statecharts for Developing Reactive Systems. In: IEE Design and Test, 2021, S. 35–47

[HMP+04] HEVNER, A. R.; MARCH, S. T.; PARK, J.; RAM, S.: Design Science in Information Systems Research. In: MIS Quarterly, Management Information Systems Research Center, University of Minnesota, 2004, S. 75–105

[HMW+11] HAREL, D.; MARRON, A.; WIENER, G.; WEISS, G.: Behavioral Programming, Decentralized Control, and Multiple Time Scales. In: Association for Computing Machinery, 2011, S. 171–182

[HMW12] HAREL, D.; MARRON, A.; WEISS, G.: Behavioral programming. 55, ACM, 2012, S. 90–100

[Hol05] HOLE, E.: Architectures as a Framework for Effective and Efficient Product Development in a Dynamic Business Environment. In: Conference on Systems Engineering Research, Stevens Institute of Technology, 2005

[Hol10] HOLTMANN, J.: Mit Satzmustern von textuellen Anforderungen zu Modellen. In: OBJEKTspektrum, 2010

[Hol19] HOLTMANN, J.: IMPROVEMENT OF SOFTWARE REQUIREMENTS QUALITY BASED ON SYSTEMS ENGINEERING, Dissertation, Universität Paderborn, 2019

[Hon13] HONOUR, E.: Verification and Validation Issues in Systems of Systems. In: Electronic Proceedings in Theoretical Computer Science, 2013, S. 2–7

[Hoo93] HOOKS, I.: Writing Good Requirements (A Requirements Working Group Information Report), 1993, Unter: http://scowen.asu.edu/Additional%20Reading/Writing%20Good%20Requirements.pdf

[HP00] HOUDEK, F.; POHL, K.: Analyzing requirements engineering processes: a case study. In: Proceedings 11th International Workshop on Database and Expert Systems Applications, 2000, S. 983–987

[HP19] HOLT, J.; PERRY, S.: SysML for Systems Engineering: A Model-Based Approach. IET Digital Library, 2019 – ISBN: 9781849196512

[HP85] HAREL, D.; PNUELI, A.: On the Development of Reactive Systems. In: Logics and Models of Concurrent Systems, 1985, S. 477–498

[HR18] HOEHNE, O.; RUSHTON, G.: A System of Systems Approach to Automotive Challenges. In: SAE International, 2018

[IPC+15] IVAN, G.; PACHECO, C.; CALVO-MANZANO, J.; ARCILLA, M.: A proposed model for reuse of software requirements in requirements catalog. In: Journal of Software: Evolution and Process, 2015

[IPP18] IRSHAD, M.; PETERSEN, K.; POULDING, S.: A systematic literature review of software requirements reuse approaches. In: Information and Software Technology, 2018, S. 223–245

[Jac01] JACKSON, M. A.: Problem Frames: Analysing and Structuring Software Development Problems. Addison-Wesley, 2001 – ISBN: 9780201596274

[JBC98] JARKE, M.; BUI, X. T.; CARROLL, J. M.: Scenario Management: An Interdisciplinary Approach. In: Requirements Engineering, 1998, S. 155–173

[JTH20] JUHNKE, K.; TICHY, M.; HOUDEK, F.: Challenges concerning test case specifications in automotive software testing: assessment of frequency and criticality. In: Software Quality Journal, 2020

[Juh21] JUHNKE, K.: Improving the quality of automotive test case specifications, Dissertation, Universität Ulm, 2021

[Kai13] KAISER, L.: Rahmenwerk zur Modellierung einer plausiblen Systemstruktur mechatronischer Systeme, Universität Paderborn, 2013

[Kan03] KANER, C.: An Introduction to Scenario Testing, 2003, Unter: https://kaner.com/pdfs/ScenarioIntroVer4.pdf

[KC02] KONRAD, S.; CHENG, B. H. C.: Requirements patterns for embedded systems. In: Proceedings of the IEEE International Conference on Requirements Engineering, 2002, S. 127–136

[KC05] KONRAD, S.; CHENG, B.: Real-time specification patterns. In: Proceedings - 27th International Conference on Software Engineering, ICSE05, 2005, S. 372–381

[KCC05] KONRAD, S.; CHENG, B.; CAMPBELL, L.: Object analysis patterns for embedded systems. In: Software Engineering, IEEE Transactions on, Jan. 2005, S. 970–992

[KDB+19] KIRPES, B.; DANNER, P.; BASMADJIAN, R.; MEER, H. de; BECKER, C.: E-Mobility Systems Architecture: a model-based framework for managing complexity and interoperability. 2, Energy Informatics, 2019

[KPM13] KASOJU, A.; PETERSEN, K.; MÄNTYLÄ, M.: Analyzing an automotive testing process with evidence-based software engineering. In: Information and Software Technology, Elsevier, 2013, S. 1237–1259

[LAR+14] LOHMEYER, Q.; ALBERS, A.; RADIMERSKY, A.; BREITSCHUH, J.; HORVATH, I.; RUSAK, Z.: Individual and organizational acceptance of systems engineering methods: Survey and recommendations. In: Proc. TMCE, 2014, S. 1531–1540

[Lig09] LIGGESMEYER, P.: Software-Qualität: Testen, Analysieren und Verifizieren von Software. Spektrum Akademischer Verlag, 2009 – ISBN: 9783827420565

[LK23] LIEBEL, G.; KNAUSS, E.: Aspects of Modelling Requirements in Very Large Agile Systems Engineering. In: Journal of Systems and Software, 2023

[LTK+18] LIEBEL, G.; TICHY, M.; KNAUSS, E.; LJUNGKRANTZ, O.; STIEGLBAUER, G.: Organisation and communication problems in automotive requirements engineering. 23, Springer London, 2018, S. 145–167

[LTK19] LIEBEL, G.; TICHY, M.; KNAUSS, E.: Use, potential, and showstoppers of models in automotive requirements engineering. 18, Springer Berlin Heidelberg, 2019, S. 2587–2607

[Luh04] LUHMANN, N.: Einführung in die Systemtheorie, 2004 – ISBN: 3896704591

[Mai99] MAIER, M. W.: Architecting principles for systems-of-systems. In: INCOSE Systems Engineering, 1999

[MBB+21] MANDEL, C.; BONING, J.; BEHRENDT, M.; ALBERS, A.: A Model-Based Systems Engineering Approach to Support Continuous Validation in PGE - Product Generation Engineering. In: 2021 International Symposium on Systems Engineering, 2021, S. 1–8

[McG99] MCGREGOR, J. D.: Test patterns: please stand by. In: Journal of Object-Oriented Programming, SIGS PUBLICATIONS INC 71 WEST 23RD ST, 3RD FLOOR, NEW YORK, NY 10010 USA, 1999, S. 14–19

[MGM+22] MANDEL, C.; GUENTHER, M.; MARTIN, A.; WINDISCH, E.; BURSAC, N.; RAPP, S.; ANACKER, H.; ALBERS, A.: Towards a System of Systems Engineering Architecture Framework. In: 2022 17th Annual System of Systems Engineering Conference (SOSE), 2022, S. 221–226

[MMA22] MANDEL, C.; MARTIN, A.; ALBERS, A.: Addressing Factors for User Acceptance of Model-Based Systems Engineering. In: The XXXIII ISPIM Innovation Conference İnnovating in a Digital World", held in Copenhagen, Denmark on 05 June to 08 June 2022. Event Proceedings: LUT Scientific and Expertise Publications: ISBN 978-952-335-694-8, 2022

[MMM15] METH, H.; MUELLER, B.; MAEDCHE, A.: Designing a requirement mining system. In: Journal of the Association for Information Systems, 2015, S. 799–837

[MSB11] MYERS, G. J.; SANDLER, C.; BADGETT, T.: The Art of Software Testing. Wiley Publishing, 3rd, 2011 – ISBN: 1118031962

[MSM+14] MATTHIESEN, S.; SCHMIDT, S.; MOESER, G.; MUNKER, F.: The Karlsruhe SysKIT Approach – A Three-step SysML Teaching Approach for Mechatronic Students. In: Procedia CIRP, 2014

[MW03] MAXIMILIEN, E. M.; WILLIAMS, L.: Assessing test-driven development at IBM. In: 25th International Conference on Software Engineering, 2003. Proceedings. 2003, S. 564–569

[MW11] MICSKEI, Z.; WAESELYNCK, H.: The many meanings of UML 2 Sequence Diagrams: a survey. In: Software and Systems Modeling (SoSyM), 2011, S. 489–514

[MW19] MAVIN, A.; WILKINSON, P.: Ten Years of EARS. In: IEEE Software, 2019, S. 10–14

[MWB+20] MANDEL, C.; WOLTER, K.; BAUSE, K.; BEHRENDT, M.; HANF, M.; ALBERS, A.: Model-Based Systems Engineering methods to support the reuse of knowledge within the development of validation environments. In: SYSCON 2020 - 14th Annual IEEE International Systems Conference, Proceedings, 2020

[MWH+09] MAVIN, A.; WILKINSON, P.; HARWOOD, A.; NOVAK, M.: Easy approach to requirements syntax (EARS). In: 2009 17th IEEE International Requirements Engineering Conference, IEEE, 2009, S. 317–322

[NFL+06] NEBUT, C.; FLEUREY, F.; LE TRAON, Y.; JÉZÉQUEL, J. M.: Automatic test generation: A use case driven approach. In: IEEE Transactions on Software Engineering, 2006, S. 140–155

[NL18] NCUBE, C.; LIM, S. L.: On systems of systems engineering: A requirements engineering perspective and research agenda. In: Proceedings - 2018 IEEE 26th International Requirements Engineering Conference, RE 2018, IEEE, 2018, S. 112–123

[NLF+15] NIELSEN, C.; LARSEN, P.; FITZGERALD, J.; WOODCOCK, J.; PELESKA, J.: Systems of Systems Engineering. 48, 2015, S. 1–41

[Nor06] NORTH, D.: Introducing behaviour-driven development, 2006, Unter: https://dannorth.net/introducing-bdd/

[NP95] NURSIMULU, K.; PROBERT, R. L.: Cause-Effect Graphing Analysis and Validation of Requirements. In: Proceedings of the 1995 Conference of the Centre for Advanced Studies on Collaborative Research, IBM Press, 1995, S. 46

[NT97] NONAKA, I.; TAKEUCHI, H.: Die Organisation des Wissens: wie japanische Unternehmen eine brachliegende Ressource nutzbar machen. Campus-Verlag, 1997 – ISBN: 9783593356433

[Pat82] PATZAK, G.: Systemtechnik — Planung komplexer innovativer Systeme: Grundlagen, Methoden, Techniken. Springer Berlin Heidelberg, 1982 – ISBN: 9783642818936

[PBD+16] POHL, K.; BROY, M.; DAEMBKES, H.; HÖNNINGER, H.: Advanced model-based engineering of embedded systems. In: Advanced Model-Based Engineering of Embedded Systems. Springer, 2016, S. 3–9

[PBS+13] PAHL, G.; BEITZ, W.; SCHULZ, H.-J.; JARECKI, U.: Pahl/Beitz Konstruktionslehre: Grundlagen erfolgreicher Produktentwicklung. Methoden und Anwendung. Springer-Verlag, 2013

[Pol85] POLANYI, M.: Implizites Wissen. In: Suhrkamp, 1985

[PQF17] PALOMARES, C.; QUER, C.; FRANCH, X.: Requirements reuse and requirement patterns: a state of the practice survey. In: Empirical Software Engineering, 2017, S. 2719–2762

[Pre03] PRETSCHNER, A.: Zum modellbasierten funktionalen Test reaktiver Systeme, Dissertation, Technische Universität München, 2003

[PRR10] PROBST, G.; RAUB, S.; ROMHARDT, K.: Wissen managen: Wie Unternehmen ihre wertvollste Ressource optimal nutzen. Gabler Verlag, 2010 – ISBN: 9783834985972

[PTR+07] PEFFERS, K.; TUUNANEN, T.; ROTHENBERGER, M. A.; CHATTERJEE, S.: A design science research methodology for information systems research. In: Journal of Management Information Systems, 2007, S. 45–77

[Rop09] ROPOHL, G.: Allgemeine Technologie : eine Systemtheorie der Technik. KIT Scientific Publishing, 2009 – ISBN: 978-3-86644-374-7

[Rop75] ROPOHL, G.: Systemtechnik – Grundlagen und Anwendung. Von G. Ropohl. Carl Hanser Verlag, München – Wien 1975. 1. Aufl., 356 S., 129 Abb., 5 Tab., geb. DM 64, –. 47, 1975, S. 582

[SK18] SALADO, A.; KANNAN, H.: A mathematical model of verification strategies. In: Systems Engineering, 2018

[SK19] SALADO, A.; KANNAN, H.: Elemental patterns of verification strategies. In: Systems Engineering, 2019, S. 370–388

[SL20] SCHNITZLER, J.; LUTTER, T.: Efficient Automotive On-Board Chargers for the Volume Ramp-up of E-Mobility. In: Automotive meets Electronics, 2020, S. 154–159

[SLS11] SPILLNER, A.; LINZ, T.; SCHAEFER, H.: Software Testing Foundations: A Study Guide for the Certified Tester Exam. Rocky Nook, 3rd, 2011 – ISBN: 9781933952789

[SM07] SEYBOLD, M. G. C.; MEIER, S.: Simulation-driven creation, validation and evolution of behavioral requirements models. In: Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme III, 2007, S. 103

[Sop16] SOPHIST: Master - Schablonen für alle Fälle, 2016, Unter: www.sophist.de

[Sta73] STACHOWIAK, H.: Allgemeine Modelltheorie. Springer, 1973 – ISBN: 9783211811061

[STP11] SIKORA, E.; TENBERGEN, B.; POHL, K.: Requirements engineering for embedded systems: An investigation of industry needs. In: Proceedings of the 17th international working conference on Requirements engineering: foundation for software quality, 2011, S. 151–165

[STP12] SIKORA, E.; TENBERGEN, B.; POHL, K.: Industry needs and research directions in requirements engineering for embedded systems. In: Requirements Engineering, 2012, S. 57–78

[Sut03] SUTCLIFFE, A.: Scenario-based requirements engineering. In: Proceedings of the IEEE International Conference on Requirements Engineering, 2003, S. 320–329

[TDB+15] TSCHIRNER, C.; DUMITRESCU, R.; BANSMANN, M.; GAUSEMEIER, J.: Tailoring Model-Based Systems Engineering concepts for industrial application. In: 2015 Annual IEEE Systems Conference (SysCon) Proceedings, 2015, S. 69–76

[UPL12] UTTING, M.; PRETSCHNER, A.; LEGEARD, B.: A taxonomy of model-based testing approaches. In: SOFTWARE TESTING, VERIFICATION AND RELIABILITY, Wiley Online Library, 2012

[Van01] VAN LAMSWEERDE, A.: Goal-oriented requirements engineering: A guided tour, 2001, DOI: 10.1109/isre.2001.948567

[Van04] VAN LAMSWEERDE, A.: Goal-oriented requirements engine ring: A roundtrip from research to practice. In: Proceedings of the IEEE International Conference on Requirements Engineering, 2004, S. 4–7

[Vog20] VOGELSANG, A.: Feature dependencies in automotive software systems: Extent, awareness, and refactoring. In: Journal of Systems and Software, 2020

[VWH+20] VOM BROCKE, J.; WINTER, R.; HEVNER, A.; MAEDCHE, A.: Special issue editorial – accumulation and evolution of design knowledge in design science research: A journey through time and space. In: Journal of the Association for Information Systems, 2020, S. 520–544

[WAB+10] WYNER, A.; ANGELOV, K.; BARZDINS, G.; DAMLJANOVIC, D.; DAVIS, B.; FUCHS, N.; HOEFLER, S.; JONES, K.; KALJURAND, K.; KUHN, T.; LUTS, M.; POOL, J.; ROSNER, M.; SCHWITTER, R.; SOWA, J.: On controlled natural languages: Properties and prospects. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2010, S. 281–289

[Web09] WEBER, J.: Automotive development processes: Processes for successful customer oriented vehicle development. Springer, 2009

[Wei16] WEILKIENS, T.: SYSMOD-The systems modeling toolbox-pragmatic MBSE with SysML. Lulu. com, 2016

[WFG+21] WIECHER, C.; FISCHBACH, J.; GREENYER, J.; VOGELSANG, A.; WOLFF, C.; DUMITRESCU, R.: Integrated and Iterative Requirements Analysis and Test Specification: A Case Study at Kostal. In: 24th International Conference on Model Driven Engineering Languages and Systems, MODELS, Fukuoka, Japan, October 10-15, 2021, IEEE, 2021, S. 112–122

[WG21] WIECHER, C.; GREENYER, J.: BeSoS: A Tool for Behavior-driven and Scenario-based Requirements Modeling for Systems of Systems. In: REFSQ 2021 Workshops, Essen, Germany, April 12, 20, CEUR-WS.org, 2021

[WGK19] WIECHER, C.; GREENYER, J.; KORTE, J.: Test-Driven Scenario Specification of Automotive Software Components. In: 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion, Munich, Germany, September 15-20, 2019, IEEE, 2019, S. 12–17

[WGW+21] WIECHER, C.; GREENYER, J.; WOLFF, C.; ANACKER, H.; DUMITRESCU, R.: Iterative and Scenario-Based Requirements Specification in a System of Systems Context. In: Requirements Engineering: Foundation for Software Quality - 27th International Working Conference, REFSQ, Essen, Germany, April 12-15, 2021, Springer, 2021, S. 165–181

[WHK+20] WOHLRAB, R.; HORKOFF, J.; KASAULI, R.; MARO, S.: Boundary Objects and Methodological Islands. In: 39th International Conference on Conceptual Modeling (ER 2020), 2020

[WJK+20] WIECHER, C.; JAPS, S.; KAISER, L.; GREENYER, J.; DUMITRESCU, R.; WOLFF, C.: Scenarios in the loop: integrated requirements analysis and automotive system validation. In: MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Canada, 18-23 October, 2020, Companion Proceedings, ACM, 2020, S. 35:1–35:10

[WMG+24] WIECHER, C.; MANDEL, C.; GÜNTHER, M.; FISCHBACH, J.; GREENYER, J.; GREINERT, M.; WOLFF, C.; DUMITRESCU, R.; MENDEZ, D.; ALBERS, A.: Model-based Analysis and Specification of Functional Requirements and Tests for Complex Automotive Systems, Systems Engineering, Wiley, 2024, DOI: 10.1002/sys.21748

[WW02] WEBER, M.; WEISBROD, J.: Requirements Engineering in Automotive Development — Experiences and Challenges. In: IEEE Software, 2002, S. 331–340

[YBB+19] YAN, S.; BI, L.; BEHRENDT, M.; ALBERS, A.: Initial reference process model for the engineering of validation environments comprising physical and virtual models. In: Procedia CIRP, Elsevier B.V., 2019, S. 1075–1081

[YBL11] YUE, T.; BRIAND, L. C.; LABICHE, Y.: A systematic review of transformation approaches between user requirements and analysis models. In: Requirements Engineering, 2011, S. 75–99

[YMB+19] YAN, S.; MANDEL, C.; BEHRENDT, M.; ALBERS, A.: Dokumentation von Erfahrungswissen in der Entwicklung von vernetzten Validierungsumgebungen Documentation of Empirical Knowledge in the Development of Connected Validation Environments. In: Stuttgarter Symposium für Produktentwicklung SSP 2019, 2019

[Yu95] YU, E.: Modeling Strategic Relationships for Process Reengineering, Dissertation, University of Toronto, 1995

[Zha04] ZHANG, Y.: Test-driven modeling for model-driven development. 21, 2004, S. 80–86

[ZHP+13] ZUGAL, S.; HAISJACKL, C.; PINGGERA, J.; WEBER, B.: Empirical evaluation of test driven modeling. In: International Journal of Information System Modeling and Design, 2013, S. 23–43

[ZPW12] ZUGAL, S.; PINGGERA, J.; WEBER, B.: Creating Declarative Process Models Using Test Driven Modeling Suite. In: International Conference on Advanced Information Systems Engineering, 2012, S. 16–32

Normen und Richtlinien

[ISOa] Road vehicles – Functional safety – Part 6: Product development at the software level, ISO 26262-6:2018, 2018

[ISOb] Road vehicles — Cybersecurity engineering, 2021

[ISOc] ISO/IEC/IEEE 26515:2011 Systems and software engineering — Developing user documentation in an agile environment, 2011

[ISOd] ISO/IEC/IEEE 42010 Systems and software engineering — Architecture description, 2011

[ISOe] ISO/IEC/IEEE 15288:2015 Systems and software engineering — System life cycle processes, 2015

[ISOf] ISO/IEC/IEEE 29148:2018 Systems and software engineering — Life cycle processes — Requirements engineering, 2018

[ISOg] Software and systems engineering — Software testing, 2022

[ITU] Message Sequence Chart (MSC), 2011

[VDA] Automotive SPICE Process Assessment: Reference Model, AutomotiveSIG, 2015

[VDI] Entwicklungsmethodik für mechatronische Systeme – Design methodology for mechatronic systems, 2004

Anhang

Inhaltsverzeichnis

Seite

A1 Ontologie aus dem MoSyS Projekt

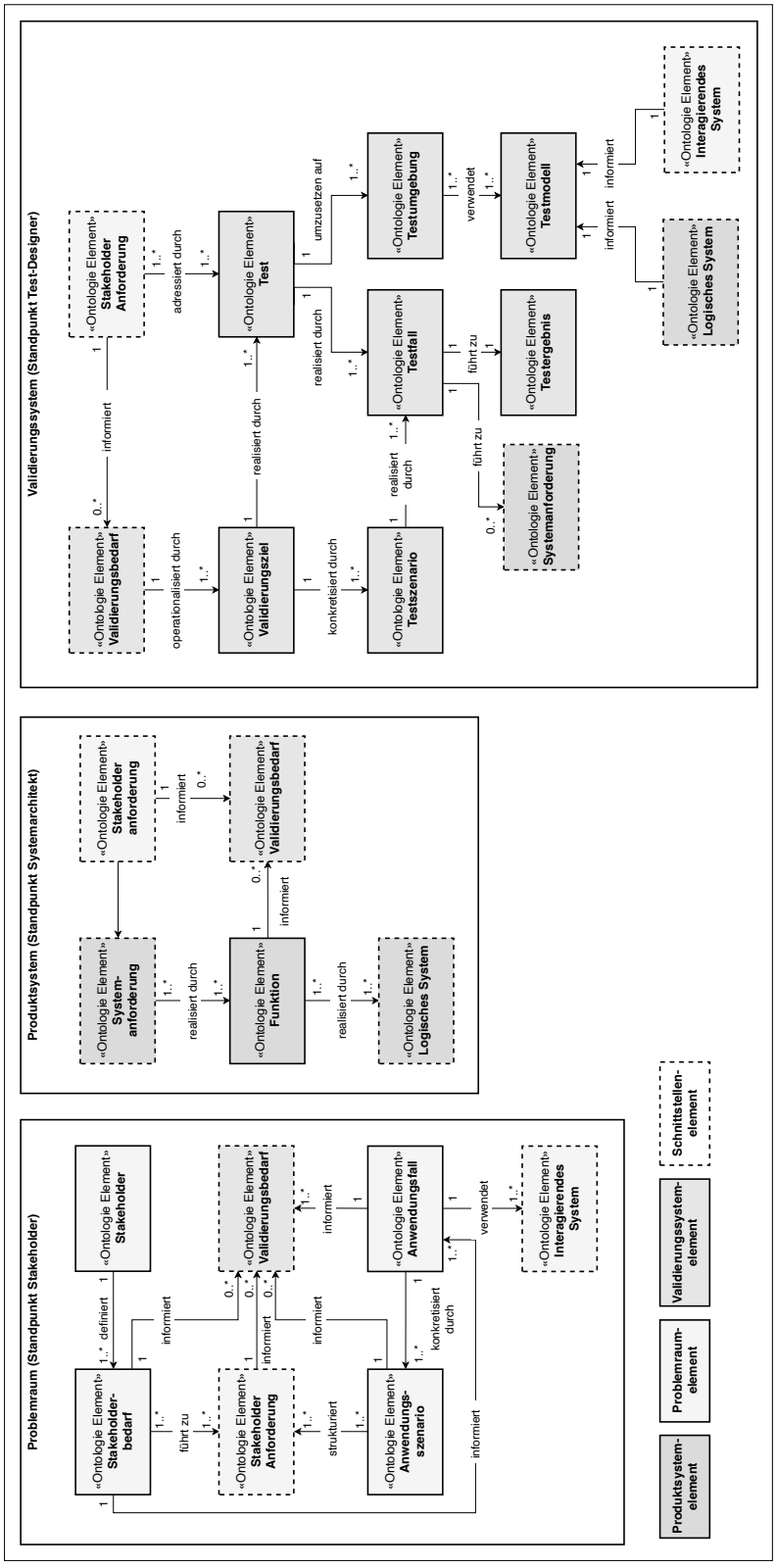


Bild A-1: Zusammenhängende Darstellung der Ontologie.

A2 Umfrageergebnisse

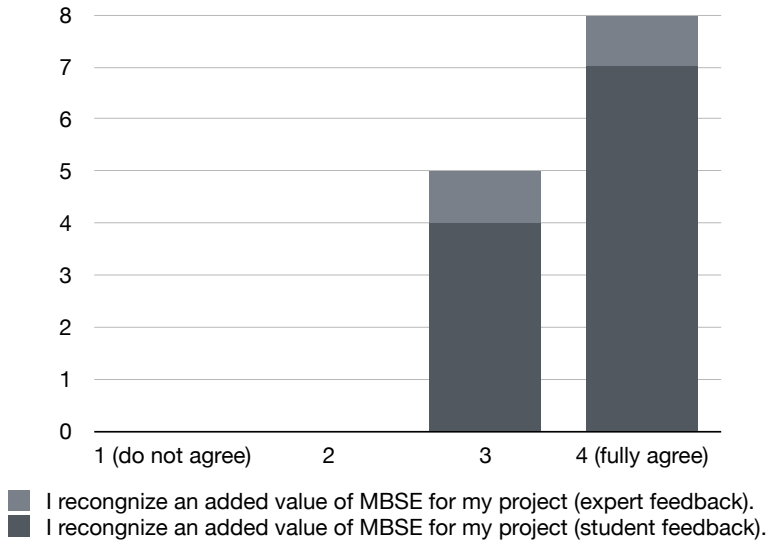


Bild A-1: Individuelle Akzeptanz - Wahrgenommene Leistungsfähigkeit von einzelnen Anwendern 1

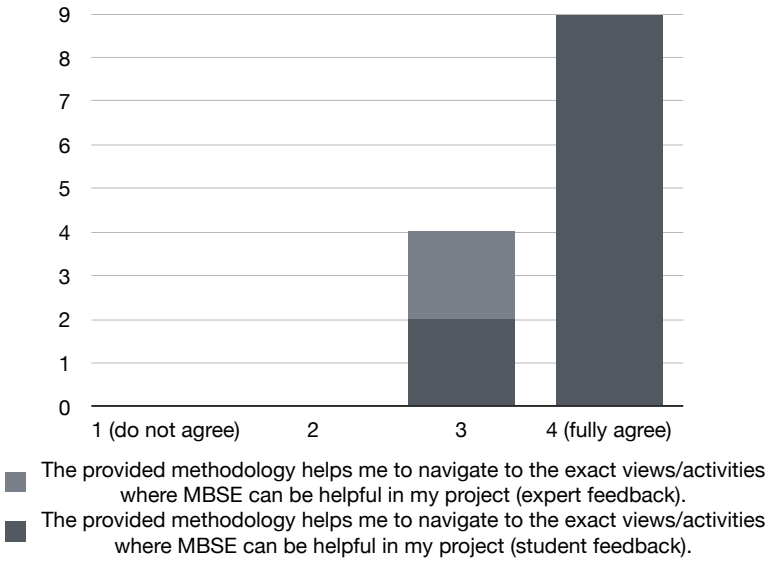


Bild A-2: Individuelle Akzeptanz - Wahrgenommene Leistungsfähigkeit von einzelnen Anwendern 2

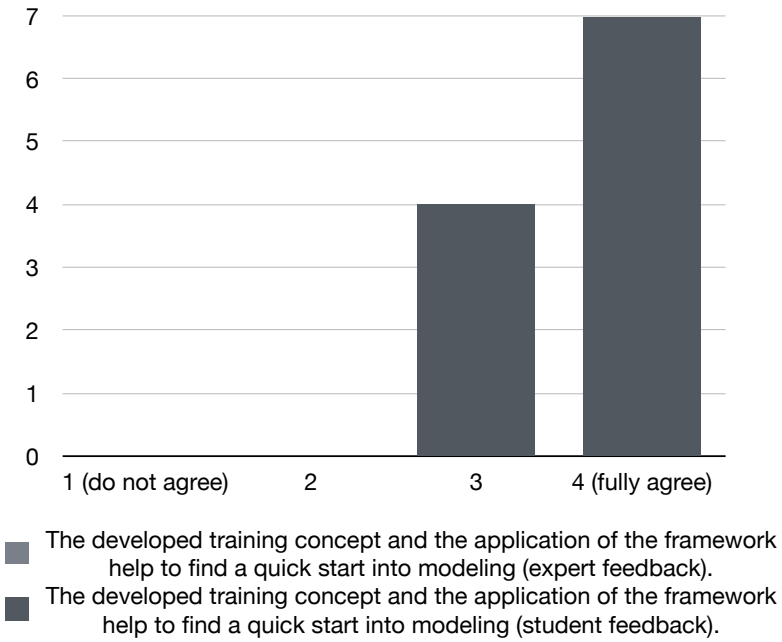


Bild A-3: Individuelle Akzeptanz - Intuitive Anwendbarkeit 1

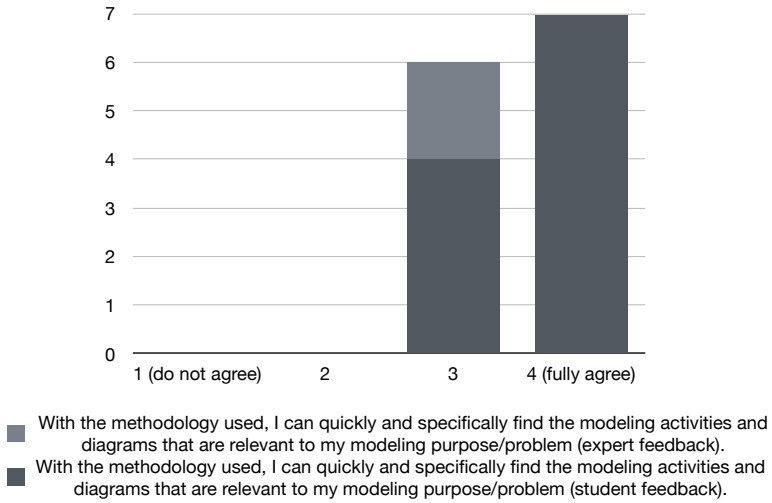


Bild A-4: Individuelle Akzeptanz - Intuitive Anwendbarkeit 2

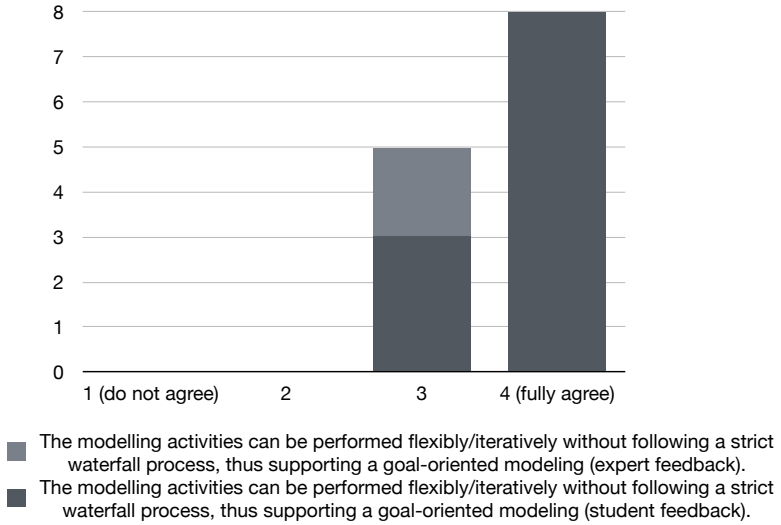


Bild A-5: Individuelle Akzeptanz - Flexibilität und Anpassungsfähigkeit der Methodik

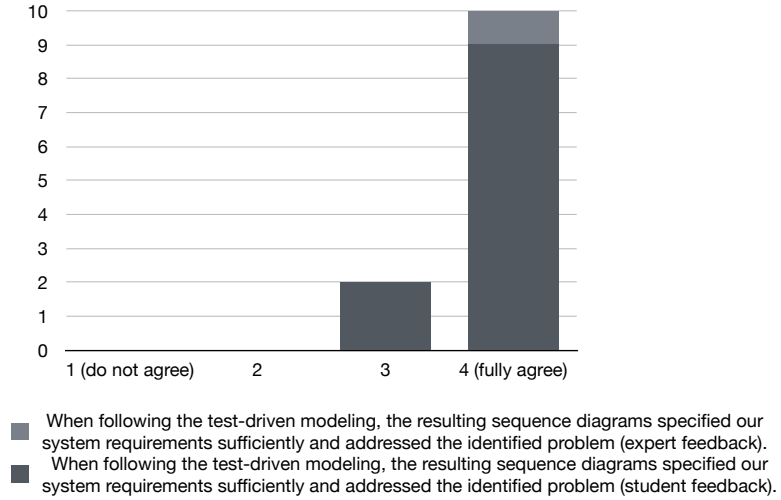


Bild A-6: Individuelle Akzeptanz - Zielvorstellung und Modellierungsverfahren für die Nutzer klar erkennbar

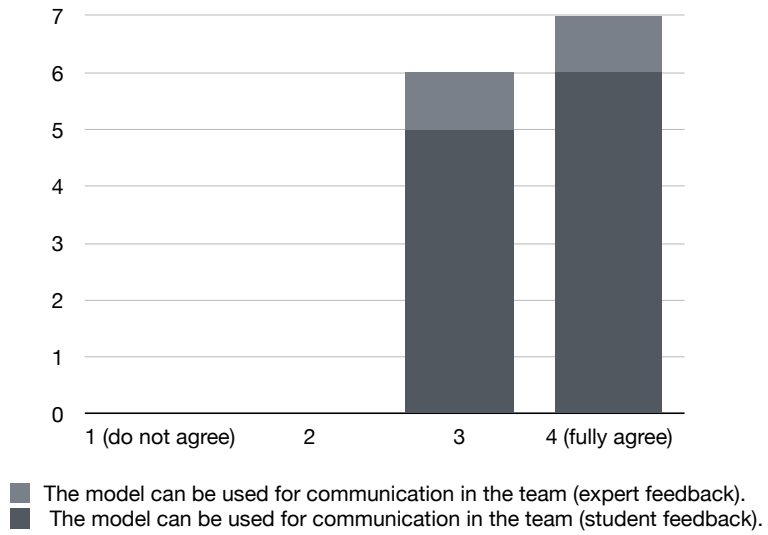


Bild A-7: Individuelle Akzeptanz - Angemessenes Maß an Formalisierung und Ontologie, um die Kommunikation zwischen den Benutzern zu unterstützen und nicht zu behindern 1

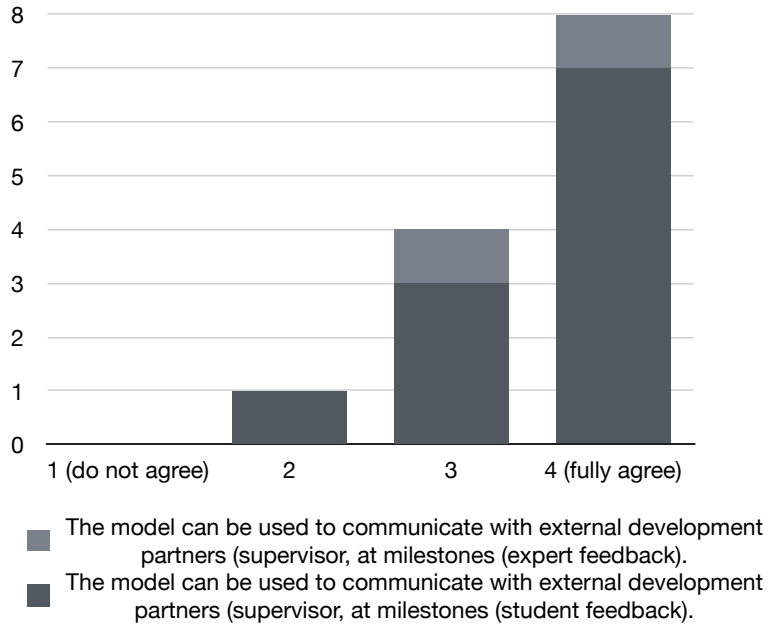


Bild A-8: Individuelle Akzeptanz - Angemessenes Maß an Formalisierung und Ontologie, um die Kommunikation zwischen den Benutzern zu unterstützen und nicht zu behindern 2

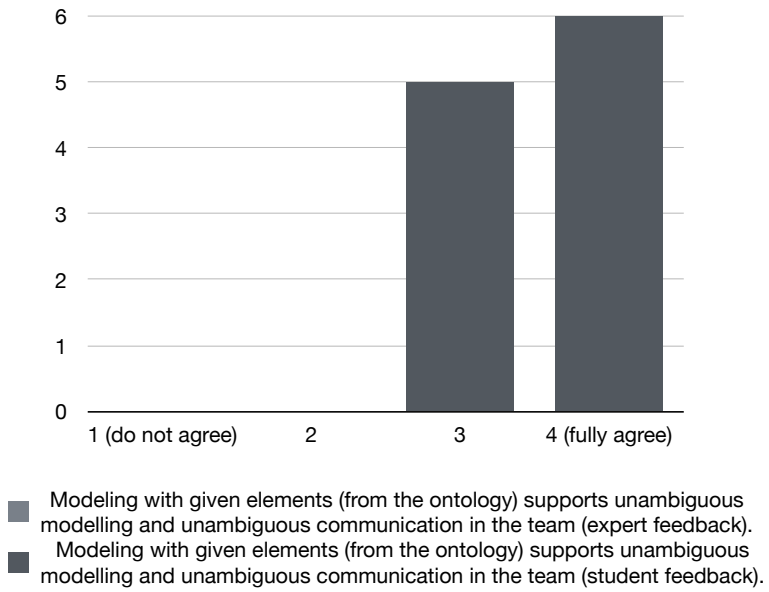


Bild A-9: Individuelle Akzeptanz - Angemessenes Maß an Formalisierung und Ontologie, um die Kommunikation zwischen den Benutzern zu unterstützen und nicht zu behindern 3

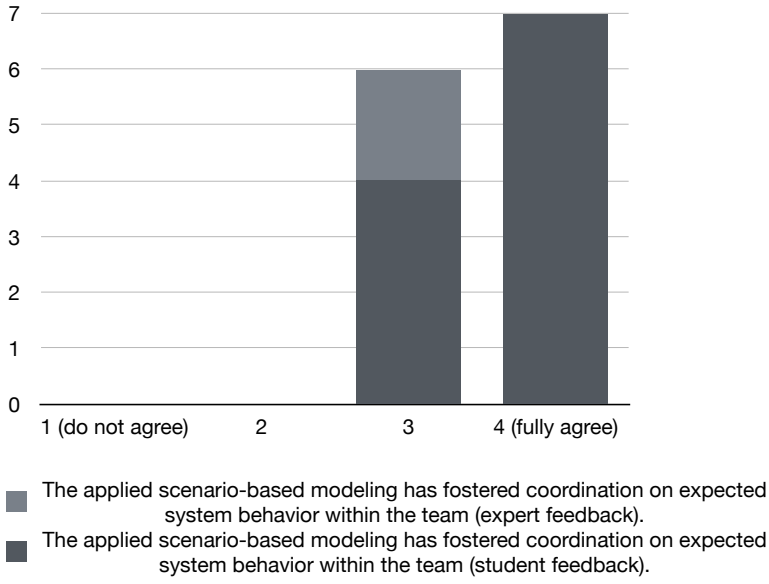


Bild A-10: Individuelle Akzeptanz - Angemessenes Maß an Formalisierung und Ontologie, um die Kommunikation zwischen den Benutzern zu unterstützen und nicht zu behindern 4

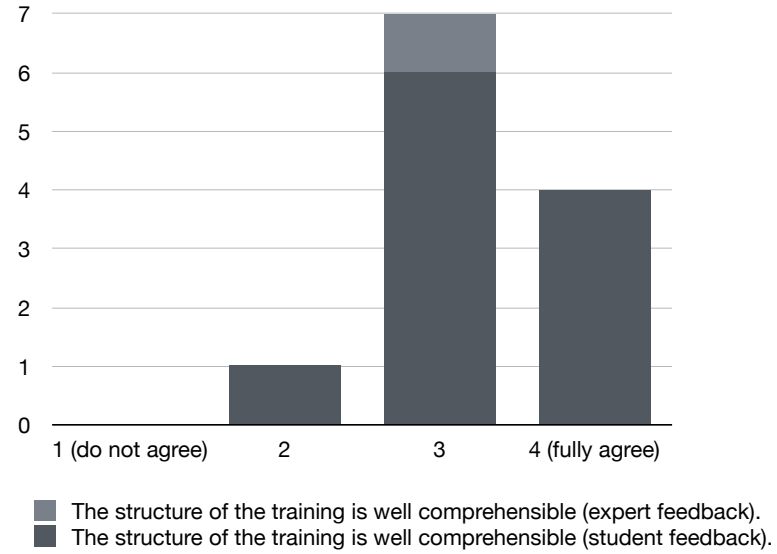


Bild A-11: Organisatorische Akzeptanz - Lehr- und Lernbarkeit der Methodik

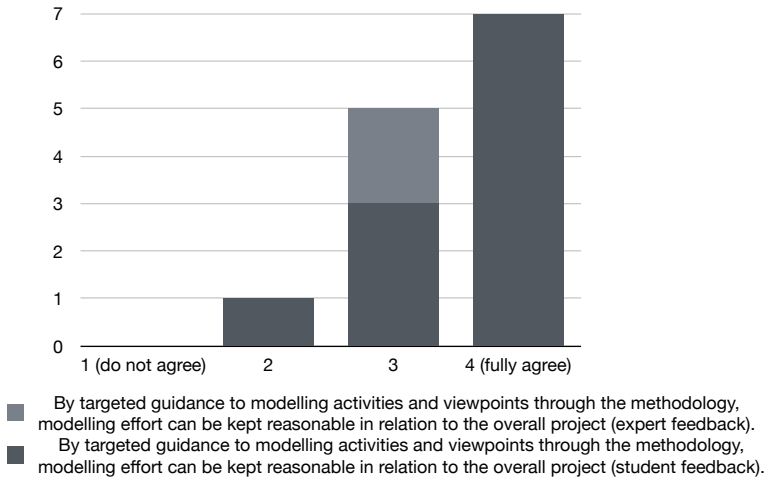


Bild A-12: Organisatorische Akzeptanz - (Monetäres) Nutzen-Kosten/Aufwand-Verhältnis der Anwendung der Methodik

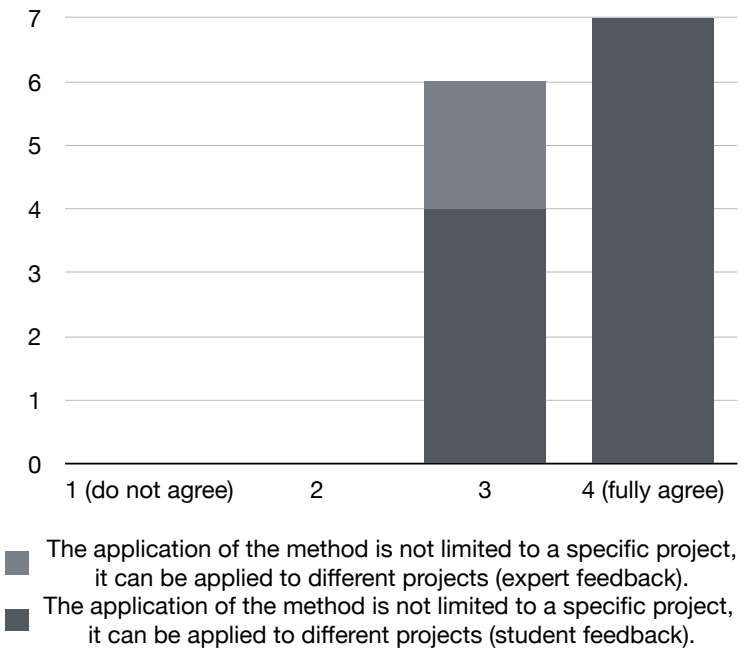


Bild A-13: Organisatorische Akzeptanz - Wiederverwendbarkeit und Erweiterbarkeit der Methode und der erstellten Modelle 1

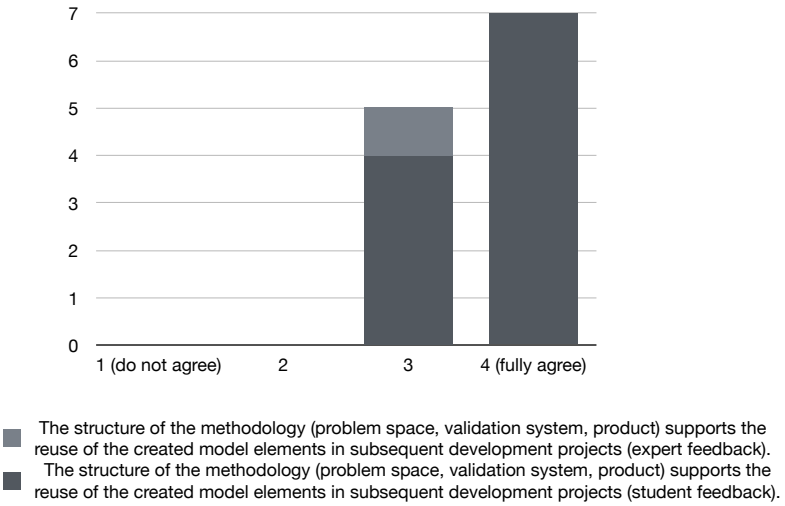


Bild A-14: Organisatorische Akzeptanz - Wiederverwendbarkeit und Erweiterbarkeit der Methode und der erstellten Modelle 2

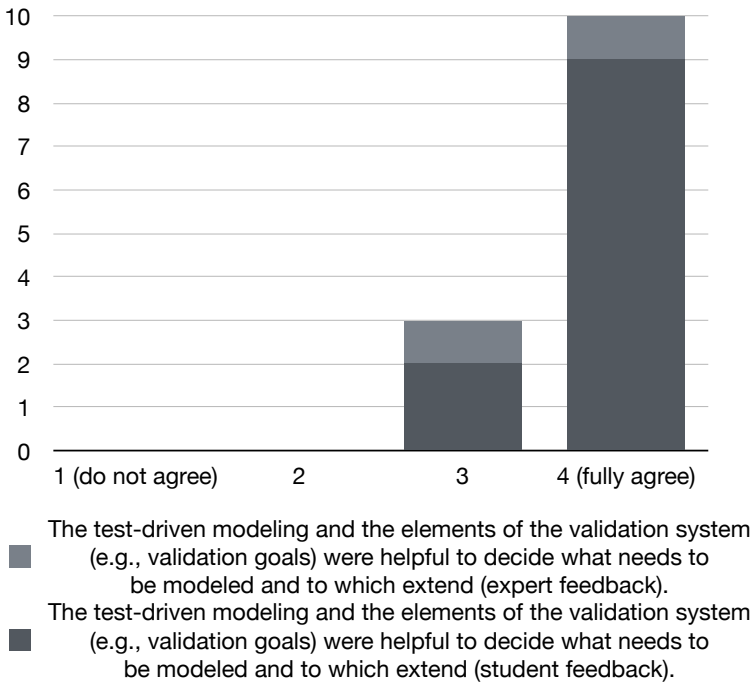


Bild A-15: Organisatorische Akzeptanz - Problemorientierung/Unterstützung bei der Modellierung des Problemraums