Analyse von Dekompositionsprozessen zur Umsetzung einer modularen Absicherung automatisierter Fahrzeuge

Björn Klamann* und Hermann Winner†

Zusammenfassung: Die in der Automobilindustrie übliche Sicherheitsvalidierung auf der Systemebene verstärkt für immer komplexere Systeme, wie automatisierte Fahrzeuge, den Nachteil, dass Änderungen an einzelnen Komponenten eine erneute Freigabe des gesamten Systems erfordern. Mit dem Konzept einer modularen Absicherung erfolgt die Sicherheitsvalidierung bereits auf Modulebene. In diesem Beitrag werden mit einem neu entwickelten Ansatz Dekompositionsprozesse vom System zu Modulen hinsichtlich möglicher Ungewissheiten und Ungenauigkeiten untersucht. Aus den Erkenntnissen werden Regeln definiert, deren Einhaltung dem Entstehen von Ungewissheiten bei einer modularen Absicherung entgegenwirkt.

Schlüsselwörter: Automatisiertes Fahren, Modulare Absicherung, Dekomposition.

1 Einleitung und Definitionen

Nach heutigem Stand der Technik konzentriert sich die Sicherheitsvalidierung in der Automobilindustrie auf die Fahrzeug- bzw. die Systemebene. Da das System Fahrzeug jedoch zunehmend komplexer geworden ist, beinhalten verknüpfte Entwicklungsprozesse inzwischen bereits auf niedrigen hierarchischen Ebenen umfassende Tests, z. B. auf Komponentenebene. Dies ist einerseits eine Konsequenz erreichter Fortschritte in Bereichen der Software- und Hardware-in-the-Loop-Tests [1], aber gleichzeitig ein Grund für intensive Weiterentwicklungen im Bereich des simulativen Testens. Trotz einhergehender Verbesserungen in der Systemzuverlässigkeit ist die endgültige Sicherheitsvalidierung nach wie vor nur auf Systemebene zulässig [2]. Es wurde bereits gezeigt, dass Feldtests auf Systemebene mit zunehmender Fahrzeugkomplexität insbesondere bei der Einführung hochautomatisierter Fahrfunktionen nicht mehr wirtschaftlich zu bewältigen sind [3]. Für automatisierte Fahrfunktionen wird das szenariobasierte Testen als vielversprechender Ansatz gesehen (siehe z. B. [4, 5]). Nach einer Sicherheitsvalidierung auf Systemebene ist es jedoch bereits aufgrund geringer Änderungen oder Systemvariationen notwendig, alle Szenarien erneut zu testen, sofern nicht anderweitig nachgewiesen werden kann, dass die Änderungen keinen Einfluss auf die Ergebnisse der Sicherheitsvalidierung haben. Darüber hinaus zeigen Amersbach und Winner [6], dass eine praktikable Anwendung des

* Björn Klamann ist wissenschaftlicher Mitarbeiter am Fachgebiet Fahrzeugtechnik der TU Darmstadt, Otto-Berndt-Straße 2, 64287 Darmstadt (e-mail: bjoern.klamann@tu-darmstadt.de).

† Hermann Winner ist Professor für Fahrzeugtechnik an der TU Darmstadt, Otto-Berndt-Straße 2, 64287 Darmstadt (e-mail: hermann.winner@tu-darmstadt.de).

szenariobasierten Testens aufgrund der erforderlichen Anzahl an Szenarien immer noch eine große quantitative Herausforderung darstellt. Das von uns vorgestellte Konzept der modularen Absicherung mit Modulen als Subsysteme des übergeordneten Systems "Fahrzeug" stellt den Vorteil in Aussicht, dass Änderungen an einem einzelnen Modul nur die wiederholte Absicherung der modifizierten Module erfordern. Hierdurch sind auch verschiedene Kombinationen von Modulen für verschiedene Anwendungsfälle umsetzbar, ohne dass für jede einzelne Kombination eine eigene Absicherung erforderlich ist [7]. Zur Realisierung einer Absicherung auf Modulebene, stellt sich die Frage nach den Unterschieden zwischen dem Entwicklungs- und Testprozess auf Systemebene und besagter Modulebene. In der vorliegenden Arbeit wird daher der Dekompositionsprozess von System- auf die Modulebene betrachtet. Dabei dekomponierte Informationen unterliegen Ungewissheiten[1] und Ungenauigkeiten[2], die in der folgenden Analyse aufgedeckt werden.
Als Modul wird ein Subsystem mit modularen Eigenschaften verstanden, wodurch es eine relative Unabhängigkeit zu anderen Subsystemen seines Systems aufweist [9, S. 28, 10, S. 209]. Im Projekt UNICAR*agil* wird darüber hinaus zwischen zweckdifferenzierten modularen Architekturen unterschieden. Im vorliegenden Beitrag wird hieraus zusammen mit den Definitionen aus der zitierten Literatur folgende Definition abgeleitet:

„Ein **Modul** ist eine Gruppierung von Komponenten mit höherer Kopplung untereinander als zu Komponenten anderer Module in Bezug auf den Zweck der jeweiligen modularen Architektur. Ein Modul ist daraus folgend relativ unabhängig von anderen Modulen in Bezug auf den Zweck der jeweiligen modularen Architektur, besitzt aber wohldefinierte Schnittstellen zu diesen."

Zweck einer modularen Architektur kann die Darstellung von Zusammenhängen der Funktionen, Hardwarekomponenten oder Softwareteilen sein. Für die von uns angestrebte modulare Absicherung gilt der Zweck, eine möglichst gute Testbarkeit bzw. Absicherung zu erreichen. Die modularen Architekturen sind dabei voneinander abhängig. Mit der Definition für ein Modul und der Darstellung der Zusammenhänge zwischen den erwähnten Hierarchieebenen nach der ISO 26262 [2] und Steimle et al. [11] ergibt sich die Darstellung nach Abbildung 1.

[1] **Ungewissheit** (eng.: uncertainty) beschreibt das Fehlen der vollständigen Gewissheit, d. h. das Vorhandensein mehrerer Möglichkeiten. Das wahre Ergebnis/Zustand/Ergebnis/Wert ist nicht bekannt. (Übersetzung nach Hubbard et al. [8]) Ungewissheit kann in diesem Beitrag über ein System, Modul oder der jeweiligen Umgebung vorliegen, sodass das deren wahres Verhalten nicht für alle Bedingungen bekannt ist. Das unbekannte Verhalten ist jedoch nicht notwendigerweise unsicher, weshalb wir von der Übersetzung „Unsicherheit" absehen.

[2] **Ungenauigkeit** beschreibt eine Abweichung, z. B. eines Simulationsmodells von der Realität, dessen Maß das Residuum ist.

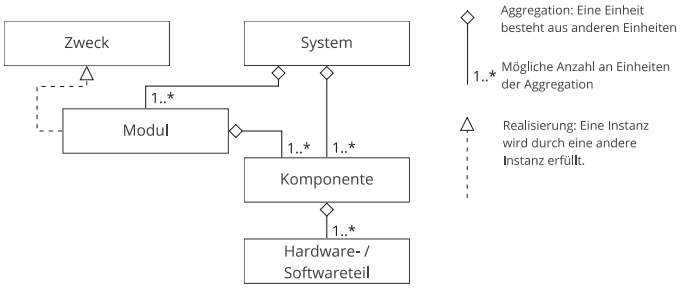

Abbildung 1: In diesem Beitrag betrachtete Hierarchieebenen und Zusammenhänge in Bezug auf Module, dargestellt als UML-Klassendiagramm.

2 Methodik und Struktur des Beitrags

Zur Ableitung möglicher fehlerhafter Schlüsse, die bei Anwendung eines modularen Absicherungsprozesses im Vergleich zur Absicherung auf Systemebene gezogen werden, ist die Aufdeckung möglicher Ungewissheiten und Ungenauigkeiten im Dekompositionsprozess notwendig. Dekomposition ist allgemein die Aufspaltung einer größeren Einheit in kleinere Einheiten. In der Systemtechnik wird angenommen, dass die dabei entstehenden kleineren Einheiten die Eigenschaften der größeren Einheit erfüllen [12, S. 127]. Diese Annahme ist jedoch erst nach Nachweis der Erfüllung der Eigenschaften der größeren Einheit möglich.

Während sich die Sicherheitsvalidierung in der Automobilindustrie vorwiegend auf das Testen konzentriert [13, S. 75], schließt der vorgestellte Ansatz neben dem Testprozedere auch den Entwicklungsprozess mit ein. Die Beherrschung des Entwicklungsprozesses ist für das Konzept einer modularen Absicherung zwingend erforderlich, da eine vollständige Testabdeckung nicht erreichbar ist [14, S. 208].

Im Entwicklungsprozess können falsche Schlussfolgerungen gezogen werden, sodass die Implementierung der Funktion davon betroffen wird. Der englische Begriff für eine fehlerhafte Implementierung („fault“) wird einheitlich in verschiedenen Domänen verwendet (siehe z.B. [2], [14], [15]). Die deutsche Übersetzung „Fehlzustand“ nach IEC [15] impliziert ggf. nur das Fehlen eines Zustands in der Implementierung. Eine fehlerhafte Implementierung aufgrund falscher Schlussfolgerungen wäre dadurch nicht eindeutig beschrieben. Wir verwenden daher den Begriff „Fehlerzustand“ nach Spillner und Linz [14, S. 7]. In der Softwareentwicklung ist eine falsche Schlussfolgerung im Entwicklungsprozess mit dem englischen Begriff „error“ belegt, der einen „fault“ verursacht. Der Begriff „error“ wird in der ISO 26262 [2] oder nach Avizieniz et al. [16] dagegen als Abweichung von einem korrekten Wert oder Zustand während des Betriebs verstanden, der erst durch einen „fault“ ausgelöst wird. Für die Entstehung des Fehlerzustands im Entwicklungsprozess wird kein Begriff genannt. Wir führen daher für eine falsche Schlussfolgerung bzw. falsche Ergebnisse im Entwicklungsprozess den Begriff „Irrtum“ ein. Dieser ist ebenfalls als mögliche Übersetzung von „error“ nach IEC aufgeführt [15]. Schlussendlich folgt nach Stolte et al. [17] aus einem Irrtum bzw. einem Fehlerzustand ggf. eine Beendigung des

gewünschten Verhaltens in Form eines Ausfalls oder Versagens (eng.: „failure“). Während der Begriff Ausfall in der Literatur breite Anwendung findet (siehe z.B. [12, S. 97], [15], [18, S. 48]), sind die Autoren des vorliegenden Artikels der Auffassung, dass der Begriff missverständlich ist und eine vollständige Beendigung einer Funktion impliziert. Dies ist für klassische Systeme im Automobil für die Mechanik und E/E-Systeme oft zutreffend, passt jedoch nicht zu einem fehlerhaften Verhalten einer Software aufgrund eines Irrtums in der Spezifikation. Daher wird der Begriff Versagen als Beendigung des beabsichtigten Verhaltens bevorzugt.
Die konkreten Irrtümer hängen von der konkreten Umsetzung des weiteren Entwicklungsprozesses ab. Diese Prozesse sind je nach Domäne und Unternehmen jedoch stark unterschiedlich und werden in anderen Publikationen analysiert (siehe z. B. [19, 20]), weshalb die vorliegende Arbeit den Dekompositionsprozess möglichst allgemein betrachtet. Die Identifizierung möglicher Irrtümer im Dekompositionsprozess erfolgt daher mit Hilfe einer Fehlerbaumanalyse ausgehend von dem Zustand eines nicht abgesicherten Moduls aufgrund von Irrtümern im Dekompositionsprozess. Zur Ergänzung des entwickelten Fehlerbaums verwenden wir einen ähnlichen Ansatz wie die Systemtheoretische Prozessanalyse (STPA) von Leveson et al. [21] Für beide identifizierten übergeordneten Irrtümer wird eine jeweils passende vereinfachte Architektur modelliert. Durch die Fehlerbaumanalyse identifizierte Irrtümer werden den modellierten Entitäten oder Verbindungen zugeordnet. Entitäten und Verbindungen ohne zugewiesene Irrtümer werden detailliert auf weitere mögliche Irrtümer durch die Zerlegungsprozesse analysiert. Aus den aufgedeckten Fehlermöglichkeiten werden daraufhin Regeln für den Dekompositions- und Testprozess definiert.

3 Irrtümer im Dekompositionsprozess

Die Analyse geht von dem Top-Ereignis aus, dass Module nach erfolgter modularer Absicherung nicht ausreichend abgesichert sind. Dies ist für den Dekompositionsprozess unter der Annahme, dass die Spezifikation auf Systemebene korrekt und vollständig ist, auf lediglich zwei allgemeine Ursachen zurückzuführen. Nach der Dekomposition und Spezifikation der Module könnten diese einerseits die Anforderungen des Gesamtsystems im Verbund nicht erfüllen, obwohl die Spezifikation eingehalten wird. Andererseits könnte eine vollständig korrekte Spezifikation nicht erfüllt sein. Für die zweite Ursache beschränken wir uns darauf, dass Modultests nicht offenlegen, dass die Spezifikation nicht erfüllt wird. Zur Aufdeckung der Irrtümer im Entwicklungsprozess, die zur Nichterfüllung der Spezifikation führen, wäre der gesamte Entwicklungsprozess zu analysieren. Dieser ist, wie zuvor erwähnt, abhängig von Unternehmen und Produkt und daher für eine generalisierte Analyse nicht geeignet. In dieser Arbeit wird folglich die Annahme getroffen, dass Modultests grundsätzlich eine Abweichung von der Modulspezifikation aufdecken können, sofern die richtigen Tests richtig durchgeführt würden. Die Ableitung solcher Testfälle und entsprechender Testumgebungen wird außerdem wesentlich vom Fokus der folgenden Analyse, dem Dekompositionsprozess, beeinflusst. Für eine modulare Absicherung müssen die Tests schlussendlich eine aggregierte Aussage über die Leistungsfähigkeit bzw. das Restrisiko des Systems liefern.

3.1 Irrtümer in Modulspezifikationen

Die erste Ursache, dass *die spezifizierte Funktion eines Moduls zusammen mit den Funktionen anderer Module nicht die Anforderungen auf Systemebene erfüllt,* wird als erster möglicher Irrtum angenommen, für den weitere untergeordnete Ursachen identifiziert werden. Die Ableitung weiterer Ursachen geschieht mit Unterstützung einer möglichst generischen funktionalen Architektur. Abbildung 2 zeigt eine solche Architektur, die aus einem System mit den Modulen 1 und 2 besteht, die die Funktionen F1, F2 und F3 haben.

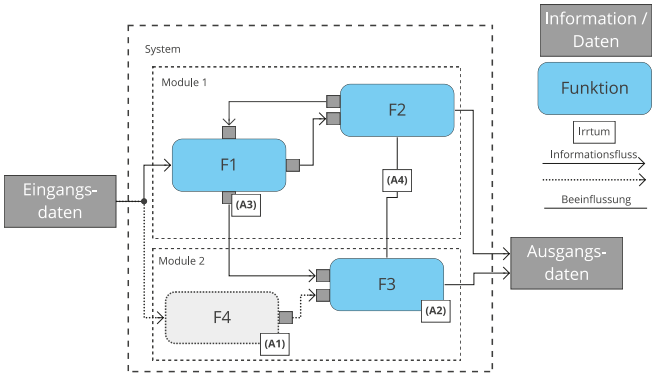

Abbildung 2: Beispielhafte abstrakte funktionale Architektur mit Irrtum an der Stelle ihres Auftretens.

Dargestellt durch einen grauen Block, stellt Funktion 4 eine fehlende Funktion dar, die eine Ursache für Irrtum A ist und als Irrtum A1 in den Fehlerbaum nach Abbildung 3 aufgenommen wird. Der Irrtum A1 kann dadurch entstehen und ggf. unentdeckt bleiben, dass Anforderungen zwar grundlegend erfüllt werden, dies jedoch für bestimmte Zustände nicht mehr gilt. In UNICAR*agil* konnte dies mit dem anfänglichen Fehlen der von Homolla et al. [21] beschriebenen Posen-Offset-Korrektur beobachtet werden. Inkonsistente Lokalisierungsdaten aus verschiedenen Lokalisierungsdiensten führen ohne die beschriebene Posen-Offset-Korrektur zu einer zunehmenden Abweichung von der Soll-Trajektorie, sodass die Anforderungen auf Systemebene nicht mehr erfüllbar sind.

Irrtum A2 bezieht sich auf fehlerhaft oder unvollständig spezifizierte Funktionen, sodass Anforderungen auf Systemebene nicht erfüllt werden. Ein einfaches Beispiel hierfür ist die Anforderung auf Systemebene, die maximale Geschwindigkeit einzuhalten, die durch eine Drehzahlbegrenzung als Funktion umgesetzt wird. Ist diese Begrenzung jedoch mit einer falschen Getriebeübersetzung berechnet, kann die Geschwindigkeitsbeschränkung auf Systemebene überschritten werden, obwohl die Anforderung auf Modulebene erfüllt ist.

Ebenso kann mit Irrtum A3 nur die Schnittstellenbeschreibung fehlerhaft, unvollständig oder inkonsistent in sich oder zu anderen Schnittstellen sein. Ein Beispiel hierfür ist die Verwendung unterschiedlicher Koordinatensysteme als Referenz für Werteangaben.

Schnittstellen können darüber hinaus auch ungewünschte Interaktionen auslösen, die unter Irrtum A4 im Dekompositionsprozess nicht erkannt werden. Insbesondere in der Mechanik ist dies oft nicht vermeidbar, da z.B. die Verbindung zweier Bauteile nicht nur Kraft,

sondern auch Wärmeenergie übertragt, deren Übertragung aber nicht gewünscht ist. Zwischen zwei Softwarediensten kann ebenfalls eine ungewünschte Interaktion, z.B. bei gemeinsamer Nutzung einer Recheneinheit auftreten, sodass für mindestens einen Dienst nicht mehr ausreichend Rechenkapazität zur Verfügung steht.

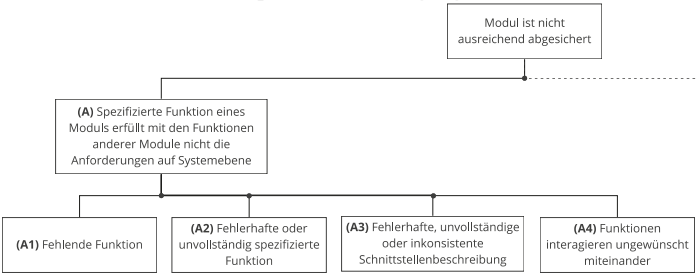

Abbildung 3: Fehlerbaum für Irrtum A.

3.2 Irrtümer in Modultests

Der folgende Abschnitt zeigt mögliche Ungewissheiten im Entwicklungsprozess auf, die als Konsequenz Ungewissheiten oder Ungenauigkeiten beim Testen verursachen. Der Irrtum, dass ein *Modul nicht ausreichend getestet* ist (B), ist entweder darauf zurückzuführen, dass die *richtigen Tests falsch durchgeführt* werden (B1) oder dass die *falschen Tests durchgeführt* werden (B2). Zur Erhöhung der Vollständigkeit des Fehlerbaums wird wieder eine Visualisierung möglicher Irrtümer eingesetzt. Abbildung 4 zeigt dazu einen generisch gehaltenen Prüfstand, der ein zu prüfendes Modul, eine spezifizierte Testumgebung sowie den Testfallgenerierungs- und Evaluationsprozess enthält.

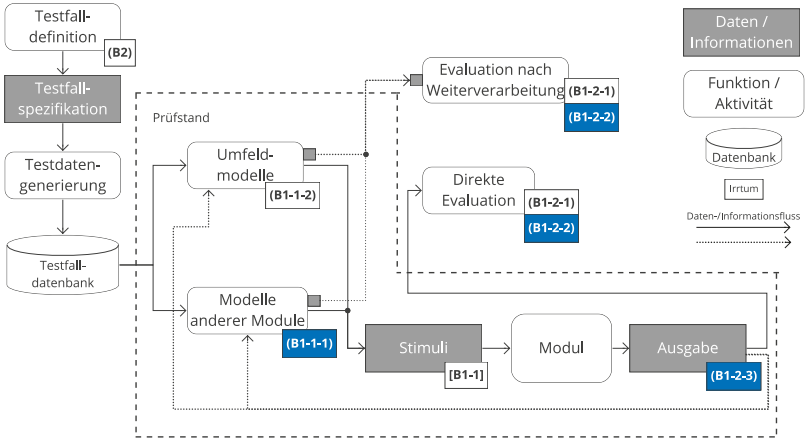

Abbildung 4: Beispielhafter generischer Prüfstandsaufbau für Modultests inkl. Testfallerzeugung und Evaluation. Irrtümer sind an der Stelle ihres Auftretens eingezeichnet. Blaue Kästen sind mit der Analyse des Prüfstands ergänzte Irrtümer.

Falsch durchgeführte Tests lassen sich auf invalide Stimulation (B1-1) oder invalide Evaluation (B1-2) zurückführen. Dabei wird für die modulare Absicherung die Bedingung gesetzt, dass Module unabhängig von anderen Modulen testbar sind. Die Einschränkung folgt aus der Anforderung der modularen Absicherung, dass Module veränderbar sind, ohne die Absicherung anderer Module erneut durchführen zu müssen. Im parallelisierten Entwicklungsprozess ist außerdem vorteilhaft, nicht auf einen ausreichend reifen Entwicklungsstand anderer Module warten zu müssen. Die Unabhängigkeit bezieht sich also auf zukünftige Änderungen anderer Module sowie deren aktuellen Entwicklungsstand bzw. die exakte Implementierung der Funktionen nach dem Konzept der Kapselung [22, S. 332 f.]. Dies ist nur möglich, wenn der Einfluss geplanter Änderungen auf das untersuchte Modul bekannt ist. Die Ermittlung zukünftiger Änderungen als auch deren Einfluss sind jedoch nicht nur aufgrund unbekannter Ausrichtung der zukünftigen Ziele eines Produkts oder des Unternehmens schwierig abzuschätzen. Auch die Aufdeckung unbekannter Unbekannter (z.B. zuvor unbekannte Szenarien, die einen Unfall verursacht und damit einen Fehlerzustand in einem Modul aufgedeckt haben) machen zukünftige Änderungen nur beschränkt vorhersehbar. Vielversprechender ist dagegen der Einsatz vereinfachter Repräsentationen anderer Module, die lediglich das für das untersuchte Modul relevante Verhalten abbilden. Änderungen anderer Module müssen dementsprechend nur ggü. diesem spezifizierten Modellverhalten abgesichert werden.
Mit Hilfe des Prüfstandsaufbaus lassen sich als Ursachen die Verwendung invalider Modelle anderer Module (B1-1-1) und speziell der Umgebung (B1-1-2) herleiten.
Für eine invalide Evaluation können auch eine unzureichende Metrik (B1-2-1), unzureichende zugeordnete Bestehens-/Versagenskriterien (B1-2-2) sowie unzureichende

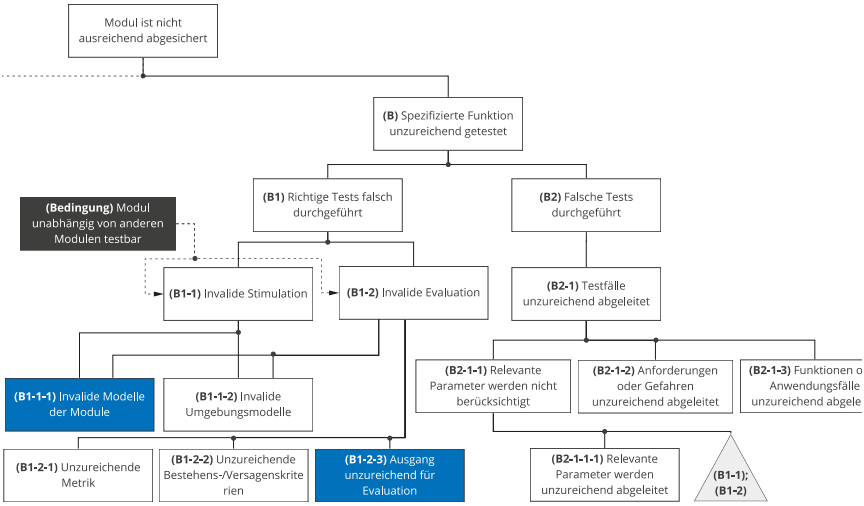

Abbildung 5: Fehlerbaum für Irrtum B.

Ausgaben am Modul (B1-2-3) als Ursachen identifiziert werden.
Der rechte Ast des Fehlerbaums listet lediglich falsche Tests aufgrund unzureichender Ableitung von Testfällen (B2-1) auf, da hier nur der Dekompositionsprozess, nicht aber die Testfallgenerierung auf Systemebene betrachtet wird. Daraus ergibt sich der potenzielle Irrtum, dass relevante Parameter nicht berücksichtigt werden (B2-1-1), da diese unzureichend abgeleitet wurden (B2-1-1-1) oder da diese über die Simulationsmodelle fehlerhaft oder unvollständig berücksichtigt werden, sodass die Stimulation bzw. Evaluation insgesamt als invalide zu bewerten ist (B1-1, B1-2). Ein derartiger Irrtum trat im Projekt UNICAR*agil* wie folgt auf. Ausgehend von der Annahme, den Regler durch hohe laterale Beschleunigungen herauszufordern, wurden Testdaten zur Abfahrt verschiedener Kreisbahnen erzeugt. Zur Initialisierung der Kreisfahrt wurde dabei zuerst geradeaus und dann in den Kreis eingefahren. Bei dieser Einfahrt zeigte sich eine größere Regelabweichung als während der gesamten Kreisfahrt. Zurückführen lässt sich dies auf die Gierbeschleunigung bei Einfahrt in die Kreisbahn, bei der der untersuchte Regler höhere Regelabweichungen erzeugt als bei lateraler Beschleunigung. Das Beispiel deckt keinen überraschenden Zusammenhang auf, illustriert jedoch wie Parameter unberechtigt vernachlässigt werden können.

4 Regeln für den Dekompositionsprozess

Anhand der zuvor entwickelten Irrtümer werden in Tabelle 1 Regeln definiert, die diese Irrtümer vermeiden. Hierzu werden die Irrtümer auf den niedrigsten Ebenen negiert und die Erkenntnisse aus deren Herleitung genutzt, um die Regeln zu konkretisieren. Die Einhaltung der Regeln im Entwicklungsprozess sind die Basis für einen erfolgreichen modularen Absicherungsprozess.

Tabelle 1: Regeln R1-R7 zur Vermeidung von Irrtümern im Dekompositionsprozess

Nr.	Regel
R1	Die funktionale Architektur muss Funktionen enthalten, die alle Anforderungen des Systems erfüllen. Somit müssen alle Funktionen für alle definierten Zustände auf die Anforderungen des Systems und vice versa rückverfolgbar sein.
R2	Die modularen Architekturen müssen die Schnittstellen zwischen den Modulen detailliert beschreiben, um sicherzustellen, dass die Funktionen gemeinsam die Anforderungen des Systems erfüllen. Die Schnittstellenbeschreibung muss hierzu eine Funktionsbeschreibung, die zulässigen Werte, ihr zeitliches Verhalten und mögliche Verläufe einschließlich eines Fehlermodells enthalten.
R3	Die modularen Architekturen müssen Abhängigkeiten zwischen den Modulen in allen möglichen Zuständen der Module bzw. des Gesamtsystems berücksichtigen.
R4	Die modularen Architekturen müssen auf unerwünschte Interaktion zwischen den Modulen untersucht werden.
R5	Die modularen Architekturen müssen Schnittstellen bereitstellen, die gültige Darstellungen anderer Module und der Umgebung in Bezug auf alle gewünschten Testfälle und deren Testziele ermöglichen. Daher sind Testfälle und Testziele vor der endgültigen Definition der modularen Architekturen zu definieren.
R6	Die modularen Architekturen müssen Schnittstellen bereitstellen, an denen alle erforderlichen Metriken und die zugehörigen Bestehens-/Versagenskriterien anwendbar sind. Hierfür können zusätzlich Modelle anderer Module oder der Umgebung verwendet werden. Daher sind Metriken und Bestehens-/Versagenskriterien vor der endgültigen Definition der modularen Architekturen zu definieren.
R7	Alle Informationen auf Modulebene (bspw. Funktionalitäten, Anforderungen, Anwendungsfälle, Parameter, Gefahren oder Testfälle) müssen durch mindestens eine Architektursicht auf die Systemebene und vice versa rückverfolgbar sein.

Regel 1 leitet sich insbesondere aus den Irrtümern A1 und A2 ab. Während die Notwendigkeit aller Funktionen zur Erfüllung der Anforderungen trivial erscheint, beinhaltet sie zusätzlich die Forderung nach Rückverfolgbarkeit von Funktionen und Anforderungen zwischen System- und Modulebene. Insbesondere für Systeme mit verschiedenen Entwicklungspartnern, die ggf. nur beschränkt Informationen untereinander austauschen, ist dies eine Herausforderung [12, S. 151]. Darüber hinaus ist die rein statische Betrachtung der Funktionen und Anforderungen eines dynamischen Systems nur zulässig, wenn die Funktionen und Anforderungen für alle dynamischen Zustände gleich formulierbar sind. Komplexe Systeme erfordern meist die Formulierung verschiedener Zustände (z.B. für verschiedene Betriebsmodi, Informationsverfügbarkeiten oder bei Richtungsabhängigkeiten), da sich ein Modul in diesen ggf. unterschiedlich verhält. Diese Zustände und zugehörige Zustandsübergänge sind bei der Spezifikation von Funktionen zu

berücksichtigen.
Regel 2 ist zu entnehmen, dass die Schnittstellen zwischen den Modulen so detailliert zu beschreiben sind, dass nicht nur die spezifizierte Modulfunktion dargestellt wird, sondern auch abgeleitet werden kann, dass alle Module im Verbund alle Anforderungen der Systemebene erfüllen. Obwohl die Beschreibung der Schnittstellen als wichtiger Meilenstein im Entwicklungsprozess gesehen wird, ist die Umsetzung einer technischen Schnittstellenbeschreibung in Entwicklungsprozessen oft lückenhaft [19, 221-224, 23, S. 314]. Derartige Lücken finden sich auch in Normen wie der ISO 26262 wieder, worin lediglich explizite Anforderungen an die Beschreibung der Hardware-Software-Schnittstellen gefordert werden [2, Teil 4]. Ein mögliches Konzept in Form einer detaillierten semantischen Schnittstellenbeschreibung wird in diesem Beitrag nach Vorstellung der weiteren Regeln erläutert.
Regel 3 erweitert Regel 2 durch die Berücksichtigung indirekter Abhängigkeiten über mehrere Module hinweg. Hierdurch werden Einflüsse von Modulen identifiziert, die keine Schnittstellen zum untersuchten Modul haben. Auf Grundlage von Sensitivitätsanalysen der jeweiligen Module lässt sich die Relevanz dieser Einflüsse quantifizieren. Der identifizierte Einfluss kann ebenfalls in einer semantischen Schnittstellenbeschreibung festgehalten werden, sodass der Einfluss bei der Absicherung des Moduls Berücksichtigung findet.
Regel 4 präzisiert Regel 3 für Einflüsse, die nicht erwünscht, ggf. aber nicht vermeidbar sind, wie bereits für Irrtum A4 beschrieben. Solche Einflüsse sind zu identifizieren und z.B. durch Sensitivitätsanalysen zu quantifizieren, um zu entscheiden, ob diese in Modultests zu berücksichtigen sind oder ob darauf verzichtet werden kann.
Aus den Regeln 5 und 6 lassen sich Anforderungen an die modularen Architektursichten ableiten. Regel 5 fordert, dass für alle notwendigen Modultests die Umgebung ausreichend valide repräsentiert werden kann und adressiert damit die möglichen Irrtümer B1-1-1 und B1-1-2. Im Rahmen einer modularen Absicherung muss diese Repräsentation mögliche Änderungen der repräsentierten Module berücksichtigen. Mögliche Änderungen sind hinsichtlich ihres Einflusses auf die Repräsentation bzw. das untersuchte Modul zu analysieren. Hierbei ist für jedes Modul ein Optimum zwischen Änderungskapazität und seiner spezifischen Konfiguration für einen bestimmten Anwendungsfall zu finden. Eine Erhöhung der Änderungskapazität bedeutet, dass umfangreichere Änderungen an Modulen zulässig sind, ohne dass andere unveränderte Module erneut abzusichern sind. Dies kann sich auf die Leistungsfähigkeit der Module auswirken, da diese mit vielfältigen Varianten/Konfigurationen kompatibel sein müssen und dazu weniger spezifisch konfiguriert werden.
Zur Einhaltung von Regel 6 und der Vermeidung der Irrtümer B1-2-1, B1-2-2 und B1-2-3 gelten ähnliche Randbedingungen für die Bestehens-/Versagenskriterien wie für die Modulrepräsentationen. Die Änderungskapazität verhält sich für Bestehens-/Versagens-kriterien ebenfalls gegenläufig dazu wie spezifisch ein Modul konfiguriert ist, sodass ein Optimum aus beiden Kriterien angestrebt wird. Grundsätzlich muss zur Einhaltung der Regel für jede Schnittstelle definiert werden, welche Kriterien erfüllt werden müssen, damit eine Ausgabe zulässig ist. Die Ausgabe kann dabei direkt oder indirekt bewertet werden. Für indirekte Bewertungen wird die eigentliche Ausgabe zuerst weiterverarbeitet. Hierfür sind ggf. Repräsentationen anderer Module notwendig. Eine Metrik bewertet die jeweilige

finale Ausgabe.
Regel 7 fordert die allgemeine Rückverfolgbarkeit von Informationen zwischen Modul- und Systemebene, um die Irrtümer B2-1-1-1, B2-1-2 und B2-1-3 zu vermeiden. Sie erweitert damit Regel 1, fordert jedoch nicht explizit die durchgehende Konsistenz zwischen den Informationen verschiedener Architektursichten, wie sie nach Regel 1 zwischen Funktionen und Anforderungen notwendig ist. Stattdessen wird für alle Architektursichten gefordert, dass die Informationen innerhalb einer Architektursicht zwischen den Hierarchieebenen rückverfolgbar sind.
In UNICAR*agil* werden Schnittstellen in einer dienstebasierten Architektur in funktionale Daten und Qualitätsdaten aufgeteilt [24]. Qualitätsdaten werden ergänzend zu den Daten zur Umsetzung der eigentlichen Funktion eingesetzt, um die aktuellen Fähigkeiten des Fahrzeugs zu bestimmen und durch einen Abgleich mit den geforderten Anforderungen zu prüfen, ob das Fahrzeug diese erfüllen kann (siehe z. B. [25]). Die Qualitätsdaten sind wie die funktionalen Daten in der Absicherung zu prüfen, sodass die Funktion der Selbstwahrnehmung sichergestellt wird. Qualitätsdaten werden aus Kenntnissen über potenzielle Irrtümer und daraus folgende Fähigkeitsdegradationen abgeleitet. Das Fehlermodell von Schnittstellen für die Absicherung von Modulen erfordert ergänzend dazu eine Beschreibung, welche Residuen an der Schnittstelle unter welchen Bedingungen auftreten. Die Herausforderung besteht jedoch weiterhin darin, das Sollverhalten an den Schnittstellen zu spezifizieren. Eine Betrachtung der Auswirkungen auf das Verhalten des Gesamtsystems ist dabei weiterhin erforderlich.
Aus den Regeln 2, 3, und 4 ergibt sich insbesondere die Notwendigkeit einer detaillierten Beschreibung der Schnittstellen. Schnittstellen lassen sich in syntaktischer Form und semantisch beschreiben. Die Syntax beschreibt dabei, wie Informationen repräsentiert und verarbeitet werden [22, S. 22]. Syntaktische Informationen der Schnittstelle bilden in der Entwicklung meist den Hauptfokus [26, S. 14], da ohne deren Abstimmung eine Kommunikation zwischen Modulen nicht möglich ist. Die Prüfung der Einhaltung an einer Schnittstelle kann durch einen manuellen Vergleich mit der Spezifikation erfolgen und durch Tests ergänzt werden, die versuchen, verschiedene Bereiche dieser Syntax anzusprechen. Neben der Prüfung, ob die Definition der Syntax eingehalten wird, lassen sich Schnittstellentests anwenden, in denen zur Stimulation des Moduls der mögliche Wertebereich systematisch angesprochen wird. Bei Schnittstellentests handelt es sich gleichzeitig um einen Funktionstest unter verschiedenen Eingaben. Hierbei wird durch die Systematik ein hohes Maß an Objektivität, üblicherweise jedoch eine geringe Effizienz bei der Aufdeckung von Fehlerzuständen erreicht. Der Grund hierfür ist, dass Fehlerursachen an einzelnen Stellen oder in einzelnen Zuständen [14, S. 153] und selten gleichverteilt über einen Wertebereich liegen. Daher werden Schnittstellentests durch Methoden wie Äquivalenzklassen- oder Grenzwerttests präzisiert.
In der Literatur und in Normen (z.B. AUTOSAR [27]) nur beschränkt adressiert ist die Semantik einer Schnittstelle. Die Semantik beschreibt als Ergänzung zur Syntax welche Informationen übertragen werden und welche logischen Abhängigkeiten existieren [22, S. 47]. Diese leiten sich üblicherweise aus den Anforderungen an das System bzw. aus den daraus dekomponierten und detaillierten Anforderungen an die Module ab. Ziel einer detaillierten semantischen Schnittstellenbeschreibung ist, die Erkenntnislücke aufgrund

fehlender Integrationstests zu schließen. Die Beschreibung sollte die Transformation bzw. Interpretation zwischen Systemstimulation und Modulstimulation bzw. Modulverhalten und Systemverhalten ermöglichen. Zur Abschätzung der Auswirkungen des Modulverhaltens auf Systemebene ist die Kenntnis des Sollverhaltens aller Module eines Regelkreises notwendig, womit Regel 2 und 3 berücksichtigt werden. Diese Verhaltensableitung ist keinesfalls an typisch prägnanten Stellen (z.B. an der Systemgrenze) zu stoppen, sondern muss immer bis zum Eingang des Moduls verfolgt werden. Damit werden auch Einflüsse auf die Modulteststimuli durch das eigene Modulverhalten berücksichtigt. Zusätzlich zu der jeweiligen Beschreibung sollten mögliche Anomalien, d.h. Abweichungen vom beschriebenen Soll-Verhalten an der Schnittstelle genannt werden. Diese können in der Modulentwicklung und in Modultests berücksichtigt werden und somit potenzielle Irrtümer aufdecken. Extrahierbar sind die Anomalien z.B. aus der Beschreibung der Qualitätsdaten oder anhand der 15 IQ(Informationsqualität)-Dimensionen [28, S. 26-27].
Ein Beispiel für die Notwendigkeit einer semantischen Beschreibung an den Schnittstellen ist das Verhalten des Kurswinkels in einer Trajektorie, wie sie in UNICAR*agil* vom Trajektorienplaner an den Trajektorienregler übermittelt wird. Der Kurswinkel kann für ein notwendiges Ausweichmanöver auf Fahrzeugebene sprunghaft verlaufen. Der Trajektorienregler muss für ein Gelingen des Ausweichmanövers den Kurswinkel möglichst schnell und präzise einregeln. In anderen Szenarien kann es dagegen vorkommen, dass aufgrund kurzzeitiger falsch-positiver Objekte oder Trajektorienkorrekturen der Soll-Kurswinkel sprungartig verläuft. Der Trajektorienregler kann dies jedoch nicht von einem Kurswinkelverlauf für ein Ausweichmanöver unterscheiden und würde ebenfalls mit hoher Querbeschleunigung reagieren. Der Trajektorienplaner muss solche sprungartigen Verläufe daher vermeiden, da aus unnötig hohen Beschleunigungen ein unsicheres Verhalten auf Systemebene erzeugt werden kann. In einer erweiterten semantischen Schnittstellenbeschreibung lassen sich solche Anforderungen an jede Schnittstelle bzw. jeden Parameter der Schnittstelle adressieren. Gleichzeitig können auch mögliche Anomalien wie ein möglicher sprungartiger Verlauf des Kurswinkels, aufgenommen und von allen Abonnenten der Schnittstelle abgenommen oder abgelehnt werden. In zukünftigen Arbeiten werden wir im Rahmen des Projekts UNICAR*agil* systematisch Schnittstellenbeschreibungen erarbeiten und deren Eignung hinsichtlich einer modularen Absicherung untersuchen.

5 Zusammenfassung

Der vorliegende Beitrag stellt das Konzept einer modularen Absicherung anhand eines strukturierten Vorgehens für Entwicklung und Testen vor, das die Unterschiede zwischen System und Modulen berücksichtigt. Als neuer Ansatz zur Identifizierung möglicher Irrtümer im Dekompositionsprozess vom System bis zum abgesicherten Modul wird eine Fehlerbaumanalyse zusammen mit der Analyse einer funktionalen Architektur und eines Prüfstandes vorgestellt. Zunächst wird die Systemebene mit der Modulebene im Hinblick auf die verfügbaren Informationen und Testmethoden verglichen. Die daraus resultierenden Schlussfolgerungen werden genutzt, um Irrtümer abzuleiten, die bei der Dekomposition dieser Informationen für die Entwicklung und den Test von Modulen auftreten können.

Daraus abgeleitete Regeln für den Dekompositionsprozess sind in dargestellter Form noch nicht verifizierbar, können in konkreten Entwicklungsprozessen jedoch als robuster Ausgangspunkt zur Ableitung spezifischer Regeln bzw. Anforderungen dienen.

Auf Systemebene bestehen generell weniger Ungewissheiten und Ungenauigkeiten als auf untergeordneten Modulebenen. Daher sind für eine Absicherung auf Systemebene weniger Analysen erforderlich als in diesem Beitrag für eine modulare Absicherung vorgestellt. Im Vergleich liegt der Analyseaufwand für eine modulare Absicherung initial höher. Allerdings verspricht dieser eine Reduzierung des zunehmenden Testaufwands für automatisierte Fahrfunktionen, da mit ihr ein baugleiches Modul für eine Vielzahl von Fahrzeugen verwendet werden kann, ohne nach Änderungen angrenzender Module Regressionstests zu erfordern. Hinzu kommt die Möglichkeit der funktionalen Dekomposition [29], die auch für Module, die weniger funktionale Ebenen beinhalten als das Gesamtsystem, eine Testaufwandsreduktion verspricht. Daher erwarten wir ein stark steigendes Interesse, sowohl in Bereichen der Forschung als auch der Durchführung, an dem vorgestellten und an weiteren neuen Ansätzen zur Umsetzung einer modularen Absicherung.

Danksagung

Diese Forschungsarbeiten wurden im Rahmen des Projekts "UNICARagil“ durchgeführt (FKZ 16EMO0286). Wir bedanken uns für die finanzielle Unterstützung des Projekts durch das Bundesministerium für Bildung und Forschung (BMBF).

6 Literatur

[1] I. Raghupatruni, S. Burton, M. Boumans, T. Huber und A. Reiter, „Credibility of software-in-the-loop environments for integrated vehicle function validation“ in *20. Internationales Stuttgarter Symposium*, Wiesbaden, 2020, S. 299–313.

[2] *Road vehicles - Functional Safety*, ISO 26262:2018, International Organization for Standardization.

[3] W. Wachenfeld und H. Winner, „The Release of Autonomous Vehicles“ in *Autonomous Driving: Technical, Legal and Social Aspects*, M. Maurer, B. Lenz, H. Winner und J. C. Gerdes, Hg., s.l.: Springer, 2016, S. 425–449, doi: 10.1007/978-3-662-48847-8_21.

[4] M. Wood et al., „Safety First for Automated Driving“, Aptiv, Audi, Baidu, BMW, Continental, Daimler, Fiat Chrysler Automob., Here, Infineon, Intel, Volkswagen, 2019. [Online]. Verfügbar unter: https://www.daimler.com/dokumente/innovation/sonstiges/safety-first-for-automated-driving.pdf. Zugriff am: 16. September 2021.

[5] *Road vehicles — Safety of the intended functionality*, ISO DIS 21448:2021, International Organization for Standardization (ISO).

[6] C. Amersbach und H. Winner, „Defining Required and Feasible Test Coverage for Scenario-Based Validation of Highly Automated Vehicles*“ in *The 22nd IEEE*

Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 2019, S. 425–430, doi: 10.1109/ITSC.2019.8917534.

[7] T. Woopen *et al.,* „UNICARagil - Disruptive Modular Architectures for Agile, Automated Vehicle Concepts“ in *27th Aachen Colloquium*, Aachen, 2018, doi: 10.18154/RWTH-2018-229909.

[8] D. W. Hubbard, *How to measure anything: Finding the value of "intangibles" in business,* 3. Aufl. Hoboken, New Jersey: Wiley, 2014.

[9] J. Göpfert, *Modulare Produktentwicklung: Zur gemeinsamen Gestaltung von Technik und Organisation*. Wiesbaden: Deutscher Universitätsverlag, 1998.

[10] D. D. Walden, G. J. Roedler, K. Forsberg, R. D. Hamelin und T. M. Shortell, Hg., *Systems engineering handbook: A guide for system life cycle processes and activities ; INCOSE-TP-2003-002-04, 2015,* 4. Aufl. Hoboken, NJ: Wiley, 2015.

[11] M. Steimle, T. Menzel und M. Maurer, „Toward a Consistent Taxonomy for Scenario-Based Development and Test Approaches for Automated Vehicles: A Proposal for a Structuring Framework, a Basic Vocabulary, and Its Application“, *IEEE Access*, Jg. 9, S. 147828–147854, 2021, doi: 10.1109/ACCESS.2021.3123504.

[12] J. Schäuffele, T. Zurawka und R. Carey, *Automotive Software Engineering: Grundlagen, Prozesse, Methoden und Werkzeuge Effizient Einsetzen,* 6. Aufl. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2016.

[13] H.-L. Ross, *Functional Safety for Road Vehicles*. Cham: Springer International Publishing, 2016.

[14] A. Spillner und T. Linz, *Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester – Foundation Level nach ISTQB®-Standard,* 6. Aufl. Heidelberg: dpunkt.verlag, 2019.

[15] International Electronical Commission, „International Electrotechnical Vocabulary: IEV ref 192-03-02 (error)“, 1. Feb. 2015. [Online]. Verfügbar unter: https://www.electropedia.org/iev/iev.nsf/display?openform&ievref=192-03-02. Zugriff am: 5. November 2021.

[16] A. Avizienis, J.-C. Laprie, B. Randell und C. Landwehr, „Basic concepts and taxonomy of dependable and secure computing“, *IEEE Trans.Dependable and Secure Comput.*, Jg. 1, Nr. 1, S. 11–33, 2004, doi: 10.1109/TDSC.2004.2.

[17] T. Stolte *et al.,* „A Taxonomy to Unify Fault Tolerance Regimes for Automotive Systems: Defining Fail-Operational, Fail-Degraded, and Fail-Safe“, *IEEE Transactions on Intelligent Vehicles*, 2021, doi: 10.1109/TIV.2021.3129933.

[18] M. Hillenbrand, *Funktionale Sicherheit nach ISO 26262 in der Konzeptphase der Entwicklung von Elektrik/Elektronik Architekturen von Fahrzeugen*. Zugl.: Karlsruhe, KIT, Dissertation, 2011. Hannover, Karlsruhe: Technische Informationsbibliothek u. Universitätsbibliothek; KIT Scientific Publishing, 2012.

[19] R. Sanchez, „Building real modularity competence in automotive design, development, production, and after-service“, *IJATM*, Nr. 3, S. 204–236, 2013, Art. no. 54918, doi: 10.1504/IJATM.2013.054918.

[20] R. Sanchez und T. Shibata, „Modularity Design Rules for Architecture Development: Theory, Implementation, and Evidence from Development of the Renault-Nissan

Alliance "Common Module Family" Architecture“, *Data Science and Service Research Discussion Paper*, Nr. 80, S. 1–44, 2018.
[21] T. Homolla, G. Gottschalg und H. Winner, „Verfahren zur Korrektur von inkonsistenten Lokalisierungsdaten in modularen technischen Systemen“ in *Uni-DAS 13. Workshop Fahrerassistenz und automatisiertes Fahren. FAS 2020*, 2021. [Online]. Verfügbar unter: http://tuprints.ulb.tu-darmstadt.de/18522/
[22] M. BROY und M. Kuhrmann, *Einführung in die Softwaretechnik*. Berlin, Heidelberg: SpringerVieweg, 2021.
[23] N. Leveson, *Engineering a safer world: Systems thinking applied to safety*. Cambridge, Massachusettes: MIT Press, 2012.
[24] A. Mokhtarian, A. Kampmann, B. Alrifaee und S. Kowalewski, „The Dynamic Service-oriented Software Architecture for the UNICARagil Project“ in *29th Aachen Colloquium Sustainable Mobility*, 2020, S. 275–284.
[25] M. Nolte, I. Jatzkowski, S. Ernst und M. Maurer, „Supporting Safe Decision Making Through Holistic System-Level Representations & Monitoring -- A Summary and Taxonomy of Self-Representation Concepts for Automated Vehicles“, 27.07.20. [Online]. Verfügbar unter: http://arxiv.org/pdf/2007.13807v2. Zugriff am: 26. Februar 2022.
[26] F. Bachmann *et al.,* „Documenting Software Architecture: Documenting Interfaces“, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA CMU/SEI-2002-TN-015, 2002. [Online]. Verfügbar unter: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5939. Zugriff am: 25. Februar 2022.
[27] *AUTOSAR: Generic Structure Template,* AUTOSAR consortium, Dez. 2017.
[28] K. Hildebrand, M. Gebauer und M. Mielke, Hg., *Daten- und Informationsqualität: Die Grundlage der Digitalisierung,* 5. Aufl. Wiesbaden: Springer Vieweg, 2021.
[29] C. T. Amersbach, „Functional Decomposition Approach - Reducing the Safety Validation Effort for Highly Automated Driving“. Dissertation, TU Darmstadt, Darmstadt, 2020.