Effiziente Diagnose von verteilten Funktionen automobiler Steuergeräte

Jens Martin Kohl

Technische Universität München

Institut für Informatik
der Technischen Universität München

Effiziente Diagnose von verteilten Funktionen automobiler Steuergeräte

Jens Martin Kohl

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr.-Ing. Alois Knoll
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. Dr. h.c. Manfred Broy
2.	Hon.-Prof. Dr.-Ing. Ulrich Heiden

Die Dissertation wurde am 17.08.2011 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 10.01.2012 angenommen.

Zusammenfassung

Software-Funktionen stellen einen großen Beitrag am Innovationsvolumen moderner Automobile. Der Entwicklungstrend geht hin zur Realisierung von Fahrzeugfunktionalitäten in Form von auf mehreren Steuergeräten verteilten, miteinander kommunizierenden Funktionen und löst somit die bisher vorwiegend komponentenzentrierte Entwicklungssichtweise ab. So bestehen moderne Automobile aus mehr als 70 Steuergeräten mit Hunderten Funktionen, die über bis zu fünf verschiedene Bussysteme kommunizieren.

Da verteilte Funktionen auch sicherheitsrelevante Anteile übernehmen, müssen mögliche Fehlverhalten rechtzeitig erkannt, gefährliche Auswirkungen beschränkt oder verhindert, sowie Hinweise zur Wiederherstellung des Systems bereitgestellt werden. Dies sind die Aufgaben der automobilen Diagnose.

Aufgrund des Verteilungsgrads der Funktionalitäten ergibt sich eine Steigerung der Komplexität der Systemarchitektur, die die Diagnose vor große Herausforderungen stellt. Die Beherrschung verteilter Funktionalitäten stellt somit eine Schlüsselkompetenz für Automobilhersteller dar. Die Wichtigkeit der Diagnose wird zudem erhöht durch die Tatsache, daß die Diagnose einen bedeutenden ökonomischen Faktor im Lebenszyklus des Fahrzeugs darstellt, unter anderem aufgrund von bis zu vierjährigen gesetzlichen Gewährleistungsverpflichtungen.

Die Dissertation befaßt sich mit der Fragestellung, wie die Effizienz der Diagnose über den gesamten Fahrzeuglebenszyklus gesteigert werden kann. In der Arbeit wird die Notwendigkeit der Diagnose sowie der inhärente Zielkonflikt zwischen Kosten und Qualität dargelegt. Basierend auf einer Analyse der gegenwärtigen Diagnose werden Kosten- und Qualitätspotentiale identifiziert.

Die Idee der Arbeit ist, durch eine Reduzierung der Kostenfaktoren Spielraum für eine Erhöhung der Diagnosequalität zu gewinnen. Zusätzlich werden durch die automatisierte Erstellung und Wiederverwendung der Diagnose langfristige Effizienzpotentiale erschlossen, die durch die automatisch unterstützte Wartung der Diagnoseelemente mittels statistischer Analyse der Lebenszyklusdaten erhöht werden.

In der Dissertation wird ein im Entwicklungsprozeß eingebetteter modellbasierter Ansatz für die on- *und* off-board Diagnose mit besonderem Fokus auf der Diagnose verteilter Funktionen vorgestellt, der sich über den gesamten Lebenszyklus der Diagnose erstreckt. Die Diagnose wird formal auf Abstraktionsebenen spezifiziert und die Diagnosefunktionen in ein formales Diagnosemodell umgewandelt. Die vorgestellten Abstraktionsebenen decken die Rollen des Handlungsfalls Diagnose ab und sind notwendig, um alle möglichen Fehler der Domäne zu erfassen. Das erstellte Modell bildet zusammen mit den zur Laufzeit erfaßten Beobachtungen der Diagnose ein boolesches Erfüllbarkeitsproblem, das sich effizient durch einen SAT-Solver lösen läßt. Schließlich ermöglicht die Einbindung des Zulieferers beginnend in den frühen Phasen des vorgeschlagenen Prozesses eine Validierung der Diagnose.

Der Nutzen des vorgestellten Ansatzes der Dissertation wurde anhand von Fallbeispielen evaluiert und aufgrund der Ergebnisse Basis für eine zukünftige Diagnosestrategie der BMW Group. Der vorgeschlagene Prozeß wurde in den BMW Standardprozeß für Diagnose integriert.

Abstract

Automotive software functions contribute greatly to a modern car's innovations. In the last years automotive software development has shifted from a component-centric view towards the realization of functions as distributed functions. Distributed functions are deployed on multiple electronic components called *electronic control units* (ECU) and communicate over bus-systems. For instance, modern cars consist of more than 70 ECUs hosting hundreds of functions connected over five different bus-systems.

With distributed functions taking over more and more safety-relevant functions, their possible faulty behaviour has to be detected and potential dangerous effects must be prevented or mitigated. Additionally, information about the fault's root cause has to be provided to support repairs in a garage. These are the tasks of the automotive diagnosis.

The distributed functionalities' many interconnections contribute to an increasing complexity of the car's system architecture and impose a huge challenge on the automotive diagnosis. Furthermore, diagnosis is an important economic factor in a car's life cycle as factors such as warranty regulations of up to four years demonstrate. Hence controlling the distributed functions through diagnosis is a key competence for automotive OEM.

The dissertation tackles the question of how the diagnosis' efficiency can be increased over the car's whole life cycle. The thesis points out the diagnosis' necessity in the automotive domain and its inherent conflict between cost and quality. Based upon an analysis of the current diagnosis' environment and process, cost and quality potentials are identified.

The thesis builds on the idea that a reduction of the diagnosis' cost factors offers economic margin to increase the diagnosis' quality. Additionally, focussing on automatically generating and reusing diagnosis functions helps leverage long-term efficiency potentials which can even be increased by an automatically supported maintenance of diagnosis data with life cycle data.

The contribution of the thesis is a model-based methodology for the on- *and* off-board diagnosis with special focus on diagnosing distributed functions, covering the whole diagnosis life cycle. The diagnosis is formally specified on abstraction levels and transformed into a formal diagnosis model. The introduced abstraction levels cover the roles of an use case diagnosis and are necessary to grasp all the domain's potential faults. The generated model and the observations taken at runtime form a boolean satisfiability problem which can be efficiently solved by a SAT-Solver. Additionally, integrating the suppliers from the early phases of the introduced process on enables the diagnosis' validation.

The thesis' methodology is evaluated in case studies and serves as basis for a future diagnostic strategy for the BMW Group. The proposed process has been integrated into BMW's standard diagnosis process.

Danksagung

Das Abschließen einer Dissertation ist nur mit der Hilfe und Unterstützung vieler Personen möglich.

Zuerst möchte ich mich bei Herrn Prof. Dr. Manfred Broy für die Übernahme der Betreuung der Arbeit bedanken. Mein Dank gilt ebenso Herrn Prof. Dr. Ulrich Heiden für die Übernahme des Zweitgutachtens. Beiden Gutachtern möchte ich zusätzlich für die vielen, langen und aufschlußreichen Diskussionen sowie Anregungen danken.

Die vorliegende Arbeit entstand während meiner Tätigkeit als Doktorand und Projektleiter innerhalb der BMW Group sowie im Rahmen eines über einjährigen Forschungsaufenthalt an der University of California San Diego.

Ich möchte mich bei der BMW Group und hier besonders bei Herrn Hubert Ströbel, Herrn Dr. Wolfgang Epple, Herrn Franz Gollmann sowie Herrn Prof. Dr. Ulrich Heiden für das Zustandekommen des Projekts sowie vor allem des Forschungsaufenthalts herzlich bedanken. Ohne diese Unterstützung wäre meine Dissertation in dieser Form so nicht möglich gewesen.

Mein ganz besonderer Dank gilt Herrn Prof. Dr. Ingolf Krüger für die Aufnahme und Betreuung des Forschungsaufenthalts an der UCSD sowie die vielen Diskussionen, die vor allem den formalen Aufbau des vorgestellten Ansatzes der Dissertation bedeutend beeinflußten und voranbrachten.

Weiterhin möchte ich mich auch bei den Mitarbeitern seiner Forschungsgruppe Frau Dr. Emilia Farcas, Herrn Massimiliano Menarini sowie Herrn Filippo Seracini vielmals für die Arbeit im gemeinsamen Projekt bedanken.

Danken möchte ich ebenso den Herren Dr. Andreas Bauer, Dr. Peter Braun, Michael Eder, Josef Kohl, Dr. Herbert Negele, Jan Philipps, Dr. Bernhard Schätz sowie Dr. Oscar Slotosch für das Gegenlesen dieser Arbeit sowie den Diskussionen und den daraus gewonnenen Erkenntnissen und Verbesserungen dieser Arbeit.

Meinen Eltern und Großeltern gebührt mein besonderer Dank für die langjährige Förderung, Ermöglichung und Unterstützung. Ihnen möchte ich diese Arbeit widmen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Dissertation befaßt sich mit der Frage, wie die Effizienz der Diagnose von (verteilten) Automobilfunktionen über den gesamten Automobillebenszyklus gesteigert werden kann. Ziel des Kapitels ist, den Leser in die Thematik der Dissertation einzuführen sowie eine kurze Übersicht über die Arbeit zu liefern.

In Abschnitt 1.1 wird gezeigt, daß der zunehmende Anteil an vernetzten Software-Funktionen am Automobil und seinen Innovationen zu einer Erhöhung der Komplexität der Gesamtfahrzeugarchitektur führt, deren Beherrschung die Automobilhersteller (OEM) vor große Herausforderungen stellt. In dieser Arbeit wird gezeigt, daß die Diagnose einen wichtigen Beitrag zur Beherrschung dieser Komplexität leistet.

Abschnitt 1.2 legt dar, daß die Diagnose im Automobil nicht nur aufgrund gesetzlicher Anforderungen verpflichtend ist, sondern auch einen wichtigen ökonomischen Faktor im gesamten Lebenszyklus darstellt. Die Bedeutung dieses Faktors wird anhand mehrerer ausgewählter Zahlen aufgezeigt. Die aufgeführten Zahlen dienen zur Verdeutlichung der Motivation der Arbeit, die Effizienz der Diagnose über den gesamten Fahrzeuglebenszyklus zu erhöhen.

In Abschnitt 1.3 wird die Problemstellung der Arbeit dargelegt.

Abschnitt 1.4 bietet eine Übersicht der Beiträge der Dissertation.

Der letzte Abschnitt liefert eine Darstellung des Aufbaus der Arbeit.

Übersicht

1.1 Steigende Komplexität des Automobils durch Software

In Automobilen der Luxusklasse sind heutzutage Hunderte von Funktionen auf mehr als 70 elektronischen Steuergeräten (ECU) gespeichert, die über bis zu fünf verschiedene Bussysteme kommunizieren. Diese Komplexität basiert zum großen Teil auf der zunehmenden Anzahl von Software-Innovationen und dem Vernetzungsgrad dieser Funktionen. Die Zunahme von Anzahl und Vernetzung der Software-Funktionen läßt sich zeitlich betrachtet in drei Abschnitte gliedern, wie Abbildung 1.1 zeigt.

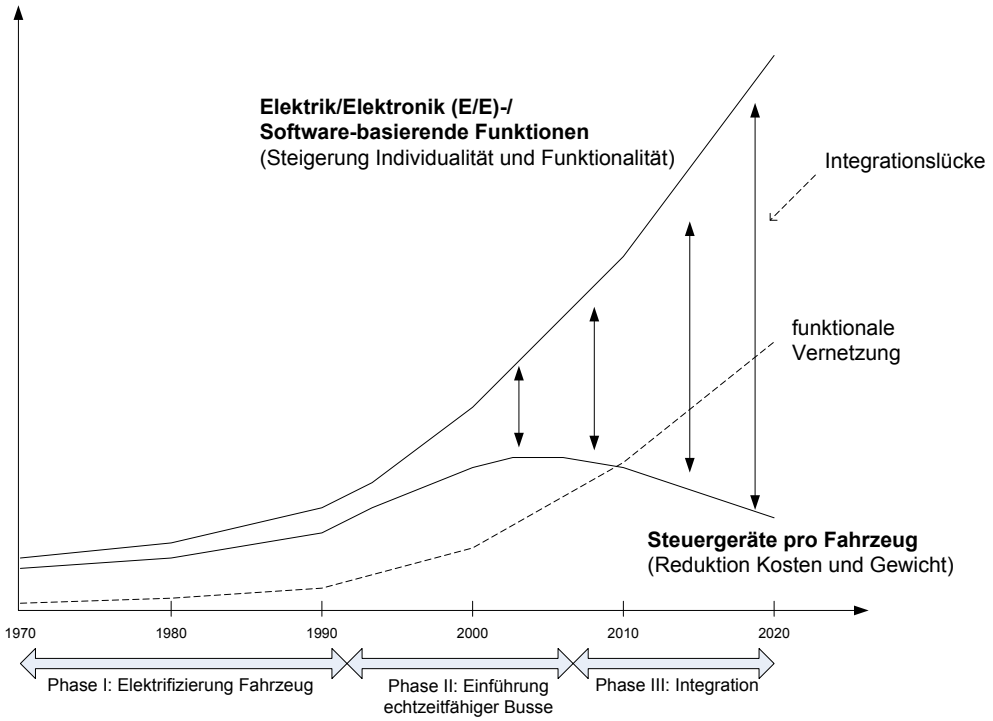


Abbildung 1.1: Steigende Integration und Vernetzungsgrad des Fahrzeugs durch E/E-Funktionen. Quelle: basierend auf [Neg06] und [Sch11, S. 2]

Die Einführung der elektronischen Steuergeräte (ECU) in der Automobilbranche Anfang der 1970er Jahre markiert den Startpunkt der Elektrifizierung und den Einzug von Software in das Automobil. Obwohl schon die ersten Steuergeräte in allen Bereichen des Fahrzeugs zu finden waren, agierten sie jedoch weitgehend autonom (vgl. [SZ10, S. 6ff]). Beispielhaft sind hier die Zentralverriegelung sowie die ersten elektronischen Einspritzsysteme zu nennen. Der technische Fortschritt der Steuergeräte, analog dem Mooreschen Gesetz für Transistoren [Moo65], führte zu einer kontinuierlichen Vervielfachung der Rechenleistung der Steuergeräte bei gleichzeitig sinkendem Preis und benötigter Verbaugröße. Diese anhaltende Entwicklung erlaubt es den Automobilherstellern, immer komplexere Funktionalitäten mit steigendem Rechenaufwand in die Fahrzeuge zu implementieren. Dies führte zur Entwicklung und Einführung erster Regelungssysteme wie der ersten Version des *Antiblockiersystem* (ABS) [Rob07b, S.840ff] im Jahre 1978.

Dennoch begann die Vernetzung des Fahrzeugs erst zunehmend mit der Einführung der ersten leistungsfähigen Bussysteme in den 1990er Jahren, wie beispielsweise

des CAN-Busses [Bos91]. Diese Bussysteme ermöglichen eine Kommunikation der Steuergeräte untereinander in Echtzeit und dadurch die Verteilung zeitkritischer Funktionalitäten auf mehrere Steuergeräte. Zudem können durch die gemeinsame Nutzung von Bauteilen wie Sensoren Kosteneinsparungen realisiert werden, da die Signale über die Busse sicher und verläßlich übertragen werden können. Ein Beispiel hierfür ist das Geschwindigkeitssignal, das im Fahrzeug am Hinterrad zentral erfaßt wird und über den CAN-Bus allen anderen Steuergeräten zur Verfügung gestellt wird. Ein weiterer wichtiger Treiber für den Anstieg der Verteilung ist die Realisierung von Kundenwünschen. Die Kundenwünsche erstrecken sich von der Erhöhung des Fahrkomforts durch Navigations- und Multimediasysteme im Fahrzeug bis hin zur Erhöhung der Fahrsicherheit durch Sicherheits- oder Fahrassistenzsysteme. Als Beispiele hierfür seien die *Adaptive Cruise Control* (ACC) [PSST00, WDS09] oder das durch den *Elch-Test* (vgl. [ISO00]) bekannt gewordene *Elektronische Stabilitätsprogramm* (ESP) [Rob07b, S.852ff] genannt.

Obwohl die Anzahl der Funktionen seitdem beständig ansteigt, existieren seit Anfang der 2000er Jahre Bestrebungen der Automobilhersteller, die Gesamtzahl der Steuergeräte eines Fahrzeugs konstant zu halten oder gar zu reduzieren, auch wenn dies gegenwärtig nur in beschränktem Maße gelingt. Dies wird durch die Integration verschiedener, oft auch unabhängiger, Funktionen auf einzelne Steuergeräte erreicht. Ein Auslöser für diese Entwicklung war neben der weiter zunehmenden Leistungsfähigkeit der Steuergeräte der zunehmende Kostendruck, da die Verringerung der Anzahl der verbauten Steuergeräte enorme Einsparungen ermöglicht. Die Integration führte zu einer deutlichen Vergrößerung der Funktionsanzahl pro Steuergerät und der Funktionsvernetzung und somit zu einer Erhöhung der Komplexität der Steuergeräte und der Systemarchitektur des Fahrzeugs insgesamt.

Anzahl und Umfang der Software-Funktionen sowie der Grad der Interaktionen untereinander werden in den nächsten Jahren weiter steigen (vgl. [Dai00, HHNZ06, BKPS07]). Wichtige Thementreiber hierfür sind die Entwicklung erweiterter Fahrassistenzfunktionen, die Einführung erweiterter Infotainment-Funktionen (vgl. [MI03, Abb. 7–9] sowie [DK04, S. 91]) oder die Vernetzung des Fahrzeugs mit dem Internet, vor allem aber die Erfüllung gesetzlicher Anforderungen an die Reduzierung von Verbrauch und Emissionen des Fahrzeugs. Hierbei sind besonders die Emissionsgesetzgebungen Kaliforniens [Cal10, S.2] und der Europäischen Union [Eur09, Artikel 1] zu nennen, die die Automobilhersteller zur Entwicklung emissionsärmerer und -freier Fahrzeuge zwingen. Neben der Einführung elektrischer Antriebe sehen die Automobilhersteller Elektronik und Software-Funktionen als wichtigen Faktor zur Erreichung dieser Vorgaben. Beispielhaft sind hier die Ersetzung der schweren mechanischen Systeme des gesamten Antriebsstrangs durch elektronische *X-by-Wire-Systeme* sowie die *Optimierung des Energiehaushalts des Gesamtfahrzeugs*.

Der beschriebene Zuwachs der Software-Funktionalitäten zeigt sich deutlich im Anstieg der Programmgröße der Steuergeräte. Verfügten die ersten Steuergeräte im Jahr 1980 über eine gesamte Programmgröße von wenigen Kilobytes, so stieg diese auf 100 Kilobyte im Jahre 1994. Mit der beginnenden Verteilung von Funktionen stieg der gesamte Speicherbedarf innerhalb von 6 Jahren auf 512 Kilobyte im Jahr 2000 [Dai00]. Das kontinuierliche Software-Wachstum führte zu einem Speicherbedarf von 100 MB im Jahr 2006 und wird für das Jahr 2011 auf 1 Gigabyte Programm-Code geschätzt [BKPS07].

1.2 Motivation der Dissertation

> „If automobiles had followed the same development cycle as the computer, a Rolls-Royce would today cost $100, get a million miles per gallon, and explode once a year, killing everyone inside“
>
> *(Robert Cringely)*

Im vorigen Abschnitt wurde der steigende Anteil der elektronischen und Softwarebasierenden Funktionen an den Fahrzeugfunktionalitäten und an sicherheitsrelevanten Fahrassistenzsystemen im Besonderen beschrieben. Da abnormales Verhalten oder gar Ausfälle dieser Funktionen kritische Folgen haben können, müssen diese Funktionalitäten auf fehlerhaftes Verhalten überwacht und Fehler falls möglich durch Eingreifen in die Funktion verhindert werden. Als fehlerhaftes Verhalten wird dabei von der Spezifikation der Funktion abweichendes Verhalten aufgefaßt. Gründe für den beschriebenen möglichen Eingriff in die Funktion liegen in der potentiellen Sicherheits- und Kundenrelevanz der Funktionen. Zusätzlich wird die Reparatur in einer Werkstatt durch Dokumentieren des Fehlers und Zuordnen der Fehlerursache zu einer austauschbaren Einheit unterstützt. Zusammengefaßt bilden diese Aufgaben den Funktionsumfang der automobilen Diagnose.

Die im vorigen Abschnitt 1.1 beschriebene Zunahme der Komplexität des Fahrzeugs hat große Auswirkungen auf die Diagnose, wie im weiteren Verlauf dieses Abschnitts gezeigt wird. Vor der Einführung der elektronischen Systeme beschränkte die Diagnose sich auf mechanische Umfänge.

Mit der Einführung der Elektronik im Fahrzeug bestand die Notwendigkeit, diese neuen Komponenten zu diagnostizieren. Die Bedeutung der Diagnose stieg durch die Einführung gesetzlicher Vorgaben in Kalifornien im Jahre 1988 (standardisiert 1994 als OBD II [Cal94]), die vorschreiben, daß die abgasrelevanten Funktionen des Autos zur Laufzeit elektronisch überwacht werden müssen. Die Elektrifizierung des Fahrzeugs bedeutet eine Steigerung der Komplexität der Diagnose, da elektronische Komponenten aufgrund beschränkten Einblicks nur innerhalb der Komponente mittels zusätzlicher Elektronik und Software-Funktionen diagnostizierbar sind und oftmals nicht mit bloßem Auge wie viele mechanische Komponenten. Aufgrund der vorwiegend autonom agierenden Komponenten war die Fehlerlokalisierung und somit die Diagnose dennoch relativ trivial.

Dies änderte sich durch die Einführung der echtzeitfähigen Bussysteme, die die zeitkritische Zusammenarbeit von Funktionen auf verschiedenen Steuergeräten ermöglichen. Die Zusammenarbeit zieht aber eine Erhöhung der Komplexität der Diagnose nach sich, da Ausfälle oder Fehler einzelner Funktionen nicht mehr lokal beschränkt sind, sondern - aufgrund der gegenseitigen Abhängigkeit - Effekte auf den Gesamtverbund haben können (vgl. [Bro03, Bro06, PBKS07]). Für die Diagnose bedeuten diese Abhängigkeiten die Notwendigkeit, für die Fehlererkennung und -lokalisierung zusätzlich zu der Komponentensicht eine Gesamtarchitektursicht zu entwickeln.

Die anhaltende Zunahme gemeinsam agierender Funktionen auf verschiedenen Steuergeräten sowie die Integration vieler, teils unabhängiger, Funktionen auf einzelne Steuergeräte bedeuten zusätzliche enorme Herausforderungen an die Diagnose. Der

steigende Verteilungsgrad führt zu einer Zunahme der transitiven Abhängigkeiten der Funktionen und somit zu einer Erhöhung der Komplexität der Diagnose der verteilten Architektur. Die Integration vieler Funktionen auf ein Steuergerät führt zu einer erhöhten Komplexität der lokalen Diagnose, da mehr und umfangreichere Fehlerbilder erkannt werden müssen.

Die Beherrschung der Komplexität des Fahrzeugsystemverbunds durch die Diagnose stellt somit eine Schlüsselkompetenz für die Automobilhersteller dar. Die Wichtigkeit wird erhöht durch die Tatsache, daß aufgrund von gesetzlichen Vorgaben wie einer Gewährleistungsdauer von zwei (EU) bis vier Jahren (USA) sowie der Möglichkeit der Wandlung im wiederholt auftretenden Fehlerfalle (das sogenannte *Lemon Law* in den USA, vgl. [MM75]) die Diagnose mit ihren erwähnten Aufgaben einen wichtigen Kostenfaktor im Lebenszyklus eines Fahrzeugs darstellt.

Die Wichtigkeit des Kostenfaktors Diagnose soll anhand von ausgewählten weiteren Kostenfaktoren vertiefend verdeutlicht werden. Abbildung 1.2 zeigt eine Einteilung der ausgewählten Kosten entsprechend ihres Auftretens im Lebenszyklus. Dabei werden der Qualität zugeordnete Kostenfaktoren in blau gehalten und Kostenfaktoren aufgrund mangelnder Qualität (Fehlerkosten) in roter Farbe. Eine genaue Darstellung des Lebenszyklus von Automobilen bietet Abbildung 2.1.

Unter dem Begriff Qualität wird im weiteren Verlauf der Arbeit die Definition der IS0 9000:2005 verwendet, die Qualität als „Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt“ [ISO05a] versteht. Der direkte Ertrag, den in die Qualität investiertes Geld erbringt, läßt sich schwer beziffern (vgl. [Stu04]) und wird deshalb im weiteren Verlauf dieser Arbeit nicht weiter betrachtet.

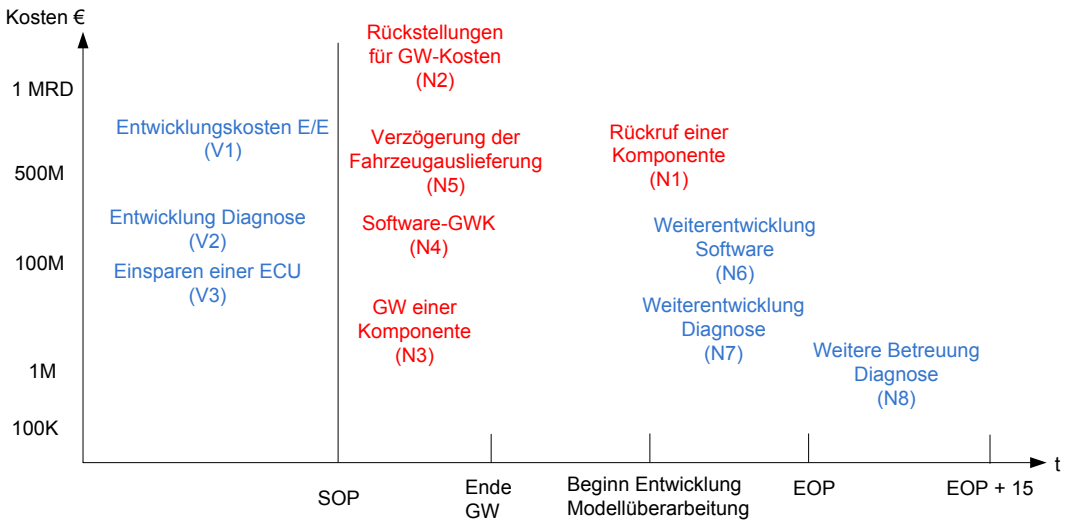


Abbildung 1.2: Beispielhafte Kosten Nichtbeherrschung der Komplexität der Diagnose. Quelle: basierend auf [MI03, HKK04, DK04, HHNZ06, Aud10, BMW10, Dai10]

Im Folgenden werden die in der Abbildung 1.2 dargestellten numerierten Kostenfaktoren detailliert. Dabei steht der führende Buchstabe *V* für einen Kostenfaktor in der Entwicklungsphase *vor* Serienanlauf des Fahrzeugs sowie *N* für einen Kostenfaktor *nach* Produktionsstart/ Serienanlauf.

Vor Serienanlauf/Produktionsstart (SOP):

V1 [MI03, Abb. 26] sowie [HKK04] schätzen den Anteil der Entwicklungskosten für Elektrik/ Elektronik an den Gesamtentwicklungskosten auf 40% für 2015. Daraus wurde ein geschätzter Anteil von 30% auf die Entwicklungskosten aus [BMW10, Dai10] abgeleitet.

V2 Aus einer eigenen Schätzung des 30-50% Anteils der Diagnose an der Software wurden die Entwicklungskosten für die Diagnose aus den Kosten von V1 abgeleitet.

V3 Das Einsparen einer ECU erbringt ca. 90 Mio. € bei einem Stückpreis der ECU von ca. 30 € bei 1'000'000 verkauften Fahrzeugen pro Jahr über 3 Jahre.

Nach Serienanlauf:

N1 Rückruf einer Komponente mit angenommenen durchschnittlichen Kosten von 500 € pro Vorfall * 1 Mio. betroffene Autos [HHNZ06].

N2 Für Gewährleistungs- und Kulanzkosten (GWK) sind vom Hersteller Rückstellungen gemäß [Int08] sowie §249 Abs. 1 des Handelsgesetzbuchs [Bun09] zu bilden. Falls diese in der Bilanz nicht direkt angegeben wurden, wurde ein Anteil von 20% an den Rückstellungen aus [Aud10, BMW10, Dai10, Toy09] geschätzt.

N3 Eigene Schätzung der summierten beispielhaften Gewährleistungskosten einer Komponente über den gesamten Gewährleistungszeitraum bei 100 € Kosten pro Fall bei angenommenen 5% Auftretenswahrscheinlichkeit über zwei Jahre GW-Zeitraum bei 1 Mio. betroffener Fahrzeuge.

N4 Durch Software verursachte GWK-Kosten reduzieren die *Earnings before Interest and Taxes* (EBIT) um 15 - 20% (aus [HHNZ06], angewandt auf [Aud10, BMW10, Dai10, Toy09]).

N5 Verzögerung der Auslieferung um drei Monate bei einer Jahresproduktion von 120'000 Fahrzeugen mit 10'000 € Profitmarge pro Fahrzeug [HHNZ06].

N6 Eigene Schätzung der Kosten für Weiterentwicklung der Software bei einer Fahrzeugmodellüberarbeitung.

N7 Eigene Schätzung der Kosten für Weiterentwicklung der Diagnose bei einer Fahrzeugmodellüberarbeitung.

N8 Paragraphen §§9, 133, 157 und 242 des BGB [Bun10] zwingen einen OEM zur Service-Betreuung des Fahrzeugs und seiner Komponenten. Gegenwärtig beträgt diese Zeitspanne bis 15 Jahre nach Produktionsende der betreffenden Baureihe.

1.3 Problemstellung

Die in Abbildung 1.2 dargestellten exemplarischen Kosten untermauern nicht nur die ökonomische Wichtigkeit der automobilen Diagnose, sondern auch den Einfluß der Qualität der Diagnose auf die Lebenszykluskosten eines Fahrzeugs.

Die Aufgaben der Diagnose sind das Entdecken von fehlerhaftem Verhalten, Eingrenzen bzw. Vermeiden des Fehlers durch Eingriff, falls dies möglich ist, sowie das Dokumentieren des Fehlers und die Zuordnung der Fehlerursache zu einer reparierbaren Einheit, um die Reparatur in der Werkstatt zu unterstützen. In der Dissertation wird unter dem Begriff *Qualität der Diagnose verstanden, wie die Diagnose diese Aufgaben erfüllt*. Der Begriff der Diagnosequalität in dieser Arbeit baut somit auf der Qualitätsdefinition der IS0 9000:2005 auf, die im vorigen Abschnitt dargelegt wurde.

Ist die Diagnose in der Lage, den Fehler einer möglichst kleinen reparierbaren Einheit zuzuordnen, lassen sich zudem die Reparaturkosten senken, da in solchen Fällen nur Teile anstelle der gesamten Einheit getauscht werden müssen. Dennoch sind der Investition von Ressourcen in die Qualität der automobilen Diagnose ökonomische Grenzen gesetzt, da die gegenwärtige Diagnose einem Konflikt zwischen Kosten und Qualität unterworfen ist: je mehr Ressourcen in die Qualität der Diagnose investiert werden, desto niedriger sind zwar die Fehlerkosten nach Serienanlauf des Fahrzeugs, da viele Fehler durch die Diagnose vermieden und behoben werden können. Gleichzeitig steigen aber die Entwicklungskosten der Diagnose.
Je weniger Ressourcen in die Diagnose investiert werden, um so geringer sind die Entwicklungskosten, desto höher aber die durch Fehler verursachten Kosten im Lebenszyklus des Fahrzeugs.

Die Diagnose befindet sich also in einem Zielkonflikt zwischen Kosten und Qualität. Dieser Zielkonflikt wird durch die ökonomisch angespannte Lage des Automobilmarktes sowie durch die in Abschnitt 1.2 beschriebene zunehmende Komplexität der Diagnose, die vor allem durch verteilte Software-Funktionen verursacht wird, zunehmend verschärft.

Die Fragestellung dieser Dissertation ist, wie dieser Zielkonflikt bewältigt werden kann. Da die wirtschaftliche Situation der Automobilbranche weiterhin angespannt bleiben wird (vgl. [MI03, Bec05, HHNZ06]), bietet eine Steigerung der Effizienz der Diagnose einen wirksamen Hebel zur Bewältigung des Zielkonflikts.

Im Kontext dieser Dissertation wird als Diagnoseeffizienz verstanden, aus den für die Diagnose eingesetzten Mitteln eine höhere Wirksamkeit zu erreichen. Sichtbar wird dies in Form einer höheren Qualität der Diagnose sowie einer Reduktion der durch Diagnose verursachten Kosten. Der Effizienzbegriff basiert also auf der Übersetzung von Effizienz in Wirksamkeit (vgl. [Wis99, Band 2]).

1.4 Beitrag der Dissertation

Im vorigen Abschnitt wurde dargelegt, daß die Diagnose sich in einem zunehmend verschärften Zielkonflikt zwischen Kosten und Qualität befindet.

In der Dissertation wird ein Ansatz vorgestellt mit der Zielsetzung, die Effizienz der Diagnose zu erhöhen. Hierzu gehört die Beherrschung der zunehmenden Komplexität der Diagnose bei gleichzeitiger Senkung der Kostenfaktoren der Diagnose. Durch die Reduktion wichtiger Kostenfaktoren sowie der Erschließung langfristiger Potentiale wird die Effizienz der Diagnose erhöht und zudem ökonomischer Spielraum für die Erhöhung der Qualität ermöglicht. Somit wird der in Abschnitt 1.3 beschriebene Zielkonflikt zwischen Kosten und Güte der Diagnose bewältigt.

Der Fokus der Arbeit liegt dabei auf der Diagnose von verteilten (Software-) Funktionen im Automobil aufgrund ihrer zunehmenden Wichtigkeit. Zudem liefern diese Funktionen einen wesentlichen Beitrag zur hohen Komplexität der automobilen Diagnose. Jedoch wird gezeigt, daß der vorgestellte Ansatz der Arbeit auch für nicht vernetzte und somit sämtliche Systeme in der Zieldomäne Automobil einsetzbar ist.

Die Beiträge der Dissertation lassen sich in vier Teile gliedern, die im weiteren Verlauf der Arbeit im Detail vorgestellt werden.

Ausgangspunkt der Arbeit ist eine *Analyse der gesamten gegenwärtigen automobilen Diagnose* auf Optimierungspotentiale. Die identifizierten Potentiale erstrecken sich dabei über den gesamten Lebenszyklus des Fahrzeugs und lassen sich klassifizieren in Kosten und Qualität. Beispielhaft für Qualitätspotentiale ist die getrennte Erstellung der Diagnose für die Werkstätten und das Steuergerät. Als Kostenpotential sei stellvertretend die geringe Wiederverwendbarkeit der Diagnose im Steuergerät genannt. Zusätzlich stellen vor allem verteilte Software-Funktionen die gegenwärtigen Diagnoseansätze vor große Herausforderungen. Hier ist vor allem das Durchsuchen von riesigen Zustandsräumen, wie sie bei verteilten Software-Systemen entstehen, zu nennen. Die Analyse ermöglicht die Bestimmung von Qualitäts- und Kostenpotentialen der automobilen Diagnose über den gesamten Lebenszyklus, die als Basis für den in der Dissertation vorgestellten Ansatz dienen.

In der Dissertation wird ein Diagnoseansatz vorgestellt, der den gesamten Lebenszyklus der Diagnose abdeckt. Es handelt sich dabei um einen *umfassenden, modellbasierten Diagnoseansatz*, der für alle Komponenten des Fahrzeugs sowohl in der Werkstatt als auch im Steuergerät einsetzbar ist. Die Diagnose wird formal auf drei Abstraktionsebenen spezifiziert. Die drei Abstraktionsebenen decken mit Kunde, Werkstatt und Entwicklung die Rollen eines Handlungsfalls Diagnose ab und sind notwendig, um alle möglichen Fehler, die in der Domäne auftreten können, zu erfassen. Zudem wird die Wiederverwendung von Diagnosefunktionen durch die Abstraktionsebenen erleichtert. Die formal spezifizierte Diagnose wird anschließend automatisiert in ein formales Diagnosemodell umgewandelt, das sowohl für die Werkstatt- als auch Steuergerätdiagnose einsetzbar ist. Das gespeicherte Modell ergibt zusammen mit den zur Laufzeit erfaßten Beobachtungen ein Erfüllbarkeitsproblem, das sich durch einen SAT-Solver (vgl. [GPFW96, MMZ+01, Zha03]) auch für Systeme mit großen Zustandsräumen, wie die genannten verteilten Software-Systeme, recht effizient lösen läßt (vgl. [GPFW96]). Zudem enthält der vorgestellte Ansatz ein multivariates, statistisches Verfahren, das die Wartung der Diagnosedaten über den gesamten Lebenszyklus über-

nimmt. Die statistischen Analysen ermöglichen neben der Wartung der Diagnosedaten auch die Entdeckung und Analyse von zum Entwicklungszeitpunkt unbekannten Fehlerbildern. Die vorgestellte Diagnosemethodik erschließt somit die identifizierten Qualitäts- und Kostenpotentiale.

Der vorgeschlagene Diagnoseansatz wird in einen *Diagnoseprozeß für (verteilte) Funktionen* eingebunden, um so langfristige Potentiale reproduzierbar zu erschließen. Der Prozeß umfaßt dabei sowohl den Entwicklungsprozeß der Diagnose als auch den Entwicklungsprozeß einer Fahrzeugmodellüberarbeitung sowie den Lebenszyklus der Diagnose. Der vorgestellte Prozeß ist sowohl geeignet für einen Einsatz bei einem Automobilhersteller als auch bei seinen Zulieferern. Die im Prozeß gestellten Anforderungen an die Diagnose beziehen die Diagnose in das Design eines Systems ein. Zusätzlich ermöglichen die Anforderungen zusammen mit der formalen Spezifikation des vorgestellten Ansatzes eine Validierung der Diagnosedaten und -funktionen im Entwicklungsprozeß beginnend in den frühen Prozeßphasen.

Als Datenschnittstelle des Prozesses zwischen Automobilhersteller und seinen Zulieferern dient eine Erweiterung der bekannten Qualitätsmethode FMEA. Das Einsetzen des strukturierten FMEA-Verfahrens hat den Vorteil, alle bekannten Fehler systematisch zu erfassen und mit Maßnahmen für die Vermeidung und Behebung des fehlerhaften Verhaltens zu belegen. Somit liefert die Anwendung einer FMEA einen wichtigen Beitrag zu einer hohen Diagnosequalität. Die *erweiterte FMEA* resultiert aus identifizierten Qualitäts- und Kostenpotentialen der FMEA und setzt auf dem vorgestellten Rollenansatz auf. Für die erfaßten Fehler wird untersucht, wie diese für die einzelnen Rollen sichtbar sowie mit Maßnahmen vermeidbar und behebbar sind. Die Maßnahmen werden mit statistischen Daten gewichtet und zusätzlich im Lebenszyklus gewartet.

Die Beiträge der Arbeit Prozeß, Datenmodell sowie erweiterte FMEA ermöglichen im Zusammenspiel eine Zuliefererersicht mit Schnittstelle zum OEM. Die Sicht berücksichtigt zudem, welche Daten die Prozeßpartner Zulieferer und OEM benötigen. Dadurch kann der Zulieferer beginnend in den frühen Entwicklungsphasen die Diagnose zielgenau mit höherer Qualität entwickeln. Die Sicht ist vor allem bei verteilten Systemen von Nutzen, da bei diesen Systemen die Zulieferer nur Teilumfänge des Gesamtsystems entwickeln. Die Zuliefererersicht ermöglicht dann durch die definierten Schnittstellen eine frühzeitige Prüfung der Diagnose der Teilumfänge des Zulieferers gegen die Schnittstellen des Gesamtsystems und erleichtert so die Integration.

Der vorgestellte Ansatz der Dissertation wurde anhand eines Fallbeispiel evaluiert und aufgrund der Ergebnisse Ausgangspunkt für eine zukünftige Diagnosestrategie für verteilte Funktionen der BMW Group. Der vorgeschlagene Prozeß wurde in den BMW Standardprozeß für Diagnose integriert.

Bereits veröffentlichtes Material. Teile dieser Arbeit wurden bereits veröffentlicht in [KB10] sowie [KKP+11].

1.5 Aufbau der Dissertation

In *Kapitel 1* wurde die Notwendigkeit der automobilen Diagnose dargelegt und belegt, daß sie einen bedeutenden Kostenfaktor darstellt. Zudem wurde das Ziel der Arbeit vorgestellt, die Effizienz der Diagnose über den gesamten Lebenszyklus nachhaltig zu steigern. Die Arbeit fokussiert dabei auf die Diagnose von verteilten Steuergerätefunktionen.

Kapitel 2 stellt die theoretischen Grundlagen der Arbeit vor. Da die elektronischen Komponenten des Automobils verteilte, eingebettete Systeme sind, wird eine Übersicht über Diagnoseansätze für diese Systeme dargelegt und die vorgestellten Ansätze auf einen Ansatz in der Zieldomäne diskutiert. Dies ist um so wichtiger, als daß die Zieldomäne über bestimmte Eigenschaften verfügt, die sie von anderen eingebetteten Domänen differenziert. Anschließend wird der gegenwärtige Entwicklungsprozeß der automobilen Diagnose dargelegt.

Das *Kapitel 3* stellt verschiedene Potentiale der Diagnose vor. Diese Ansatzpunkte wurden durch eine Evaluation des Diagnoseumfelds von Automobilhersteller und dessen Zulieferer identifiziert. Die Idee der Arbeit ist, daß ein Ansetzen an diesen Punkten eine deutliche Steigerung der Effizienz und eine Erhöhung der Qualität der Diagnose ermöglicht. Diese Steigerung erstreckt sich durch die Erschließung langfristiger Potentiale über den gesamten Lebenszyklus des Fahrzeugs und seiner Komponenten.

Kapitel 4 stellt den Ansatz der Arbeit im Detail vor. Es handelt sich hierbei um eine modellbasierte Methodik für die on- *und* off-board Diagnose, die auf Wiederverwendbarkeit getrimmt ist und die die automatische Generierung von Diagnoseelementen ermöglicht. Die Diagnoseelemente werden durch ein statistisches Verfahren über den gesamten Lebenszyklus gewartet.

Kapitel 5 zeigt die Einbettung der evaluierten Methodik in einen erweiterten Entwicklungs- und Lebenszyklusprozeß für die Diagnose.

Kapitel 6 veranschaulicht anhand von sowohl weitverbreiteten als auch vom Funktionsumfang leichtverständlichen Komponenten den Diagnoseansatz der Arbeit. Der Umfang der Fallbeispiele beschränkt sich zudem auf eine verständliche Untermenge der möglichen Fehler der gewählten Komponenten.

In *Kapitel 7* wird die Evaluation des Diagnoseansatzes durchgeführt. Die Ergebnisse des Ansatzes werden vorgestellt und auf ihre Auswirkungen hinsichtlich Kosten und Qualität untersucht.

Kapitel 8 liefert eine Zusammenfassung der Arbeit sowie einen Ausblick auf mögliche Anknüpfungspunkte.

In *Anhang A* werden die einzelnen Prozeßschritt der in Kapitel 5 eingeführten Prozesse im Detail vorgestellt.

Anhang B zeigt das Diagnosemodell für die gewählten Umfänge des Fallbeispiels der Dissertation.

Kapitel 2

Automobile Diagnose

In diesem Kapitel werden die Grundlagen der Diagnose automobiler Steuergeräte vorgestellt. Ziel ist es, den Leser mit der Thematik vertraut zu machen, bevor im nächsten Kapitel auf die Potentiale der automobilen Diagnose eingegangen wird.

Steuergeräte gehören zur Klasse der eingebetteten, reaktiven und verteilten Systeme. Abschnitt 2.1 verdeutlicht die Eigenschaften dieser Systeme. Zusätzlich wird die aus dem Zusammenspiel der einzelnen Steuergeräte entstehende Gesamtarchitektur des Fahrzeugs beschrieben.

Aufbauend auf der erstellten Klassifizierung der Steuergeräte wird im nächsten Abschnitt die Diagnose dieser Systeme vorgestellt. Hierzu notwendig ist eine Definition des Fehlerbegriffs. Zudem wird die Wichtigkeit der Diagnose durch ihren Beitrag zur Erhöhung der Verfügbarkeit eines Systems herausgearbeitet.

In Abschnitt 2.3 werden die in den Abschnitten zuvor eingeführten Konzepte in den Kontext der Arbeit gestellt. Weiterhin wird aufgezeigt, wie die speziellen Anforderungen und Charakteristika der Automobildomäne im Spannungsfeld OEM- Zulieferer Einfluß auf die Diagnose und ihre Konzepte nehmen.

Abschnitt 2.4 stellt die bekanntesten Ansätze für die Diagnose eingebetteter Systeme aus der Literatur vor. Fokus der Betrachtung liegt dabei vor allem auf der Identifizierung von Fehlverhalten und das Schließen auf die Fehlerursache(n). Eine Diskussion der Ansätze hinsichtlich eines Einsatzes im automobilen Umfeld erfolgt in Abschnitt 3.7.

Abgerundet wird das Kapitel durch einen Überblick auf verwandte Arbeiten.

Übersicht

2.1 Automobile Steuergeräte

Automobile Steuergeräte sind eingebettete, verteilte und reaktive Systeme, die zudem Echtzeitanforderungen unterliegen. In Abschnitt 2.1.1 werden zuerst die grundlegenden Begriffe und Eigenschaften dieser Systeme erklärt, bevor die Funktionsweise und der logische Aufbau der Steuergeräte im Automobil beschrieben werden. Anschließend werden spezielle Anforderungen an automobile Steuergeräte vorgestellt (Abschnitt 2.1.2) und mit verteilten Systemen in den Branchen Verbraucherelektronik sowie Luft- und Raumfahrttechnik verglichen. Abschnitt 2.1.3 zeigt die Funktionsweise der Steuergeräte. Der letzte Abschnitt wirft einen Blick auf verteilte Steuergerätefunktionen.

2.1.1 Verteilte, reaktive, eingebettete Echtzeit-Systeme

Eingebettete Systeme

Eingebettete Systeme lassen sich als Computer-Systeme definieren, die Teil eines größeren Systems sind und bestimmte Umfänge der Anforderungen des größeren Systems durchführen [ISO90].

Studien wie bspw. [Ten00] schätzen, daß der Marktanteil von eingebetteten Systemen an denen aller Computer-Systeme bei 98% liegt. Wichtige Branchen für eingebettete Systeme sind neben der Automobilbranche die Luft- und Raumfahrttechnik (Flugzeuge oder Satelliten) sowie die Verbraucherelektronik (DVD-/CD-Player, mobile Telefone, Fernseher, ...).

So besteht ein Automobil neben seinen eingebetteten Systemen aus mechanischen Teilen, mit denen zusammen die eingebetteten Systeme die geforderten Fahrzeugfunktionalitäten ermöglichen. Ein Beispiel hierfür ist der Fensterheber (vgl. Abschnitt 6.1), dessen mechanische Schalter Aktionen auf einem oder mehreren eingebetteten Systemen auslösen, die wiederum zur Erfüllung der angeforderten Aktion(en) ein Fenster ansteuern, beispielsweise über einen mechanischen Seilzug.

Zudem bilden die eingebetteten Systeme Kommunikationsverbindungen mit anderen eingebetteten Systemen zur Erledigung bestimmter Anforderungen an das Gesamtfahrzeug. Dadurch entsteht ein verteiltes (Gesamt-)System.

Verteilte Systeme

> „A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable“
>
> *(Leslie Lamport)*

Setzten sich in den Anfangsjahren der Computer-Systeme die Systeme vorwiegend aus einem zentralen, großen Rechensystem zusammen, so entstanden durch den im Mooreschen Gesetz [Moo65] beschriebenen technischen Fortschritt der Transistoren immer leistungsfähigere und kompaktere Prozessoren. Die Miniaturisierung der

Prozessoren ermöglichte zudem einen Einsatz der Prozessoren in neuen Domänen mit geringem bis sehr geringem Verbauraum wie der Verbraucherelektronik oder im Automobilbereich. Beleg für diese Entwicklung ist der in der Einleitung erwähnte Anstieg des durchschnittlichen Speicherbedarfs und Code-Umfang eines Steuergeräts im Automobil.

Diese technische Entwicklung zusammen mit dem Aufkommen leistungsfähiger Netzwerktechnologien führte zu einem Paradigmenwechsel weg von einer zentralisierten hin zu einer dezentralisierten Systemarchitektur. In einer solchen Architektur übernehmen viele kompakte Systeme per Kommunikation mittels fest definierter Protokolle die Aufgaben großer Zentralsysteme. Diese Architekturen werden als *verteilte Systeme* bezeichnet.

In der Literatur existieren mehrere Definitionen für verteilte Systeme. Stellvertretend werden an dieser Stelle drei häufig verwendete Definitionen aufgeführt. Tanenbaum und van Steen definieren ein verteiltes System als eine Ansammlung unabhängiger Computer, die dem Benutzer wie ein einzelnes, kohärentes System erscheinen [TS06]. Lamport versteht als verteiltes System eine Menge unterschiedlicher Prozesse, die flächendeckend getrennt sind und über Austausch von Nachrichten miteinander kommunizieren [Lam78]. Die für diese Dissertation verwendete Definition verteilter Systeme stammt aus Coulouris et al., die ein verteiltes System als System betrachten, in dem Komponenten, die sich auf vernetzten Computern befinden, ihre Aktionen nur durch die Übertragung von Nachrichten kommunizieren und koordinieren [CDK05].

Als wichtige Eigenschaften von verteilten Systemen werden das *Fehlen einer globalen Zeit* [Lam78], die Kommunikation und *Abstimmung des Systems über Nachrichtenaustausch* sowie die Möglichkeit des Auftretens von *unabhängigen Fehlern* der parallel handelnden Komponenten beschrieben [CDK05].

Im Automobil erfolgt die Kommunikation der einzelnen Steuergeräte über den Austausch von Nachrichten über spezielle Bus-Protokolle. Dadurch entsteht der Eindruck eines einzigen Systems. So besteht beispielsweise die automatische Abstandsregelung (ACC) aus vielen parallel ablaufenden Prozessen, die auf fünf Steuergeräten verteilt sind und mittels CAN-Bus kommunizieren (vgl. Abbildung 2.6 sowie [PSST00, IC93]). Eine kurze Übersicht über in der Automobildomäne verwendete Bussysteme findet sich unter [NHB05], mit speziellem Fokus auf den Sicherheitsaspekt unter [WWP04] oder ausführlich in [ZS08].

Reaktive Systeme

Harel und Pnueli definieren reaktive Systeme als Systeme, deren Zweck die Aufrechterhaltung der Interaktion mit ihrer Umgebung ist [HP85, Pnu86]. Der Zweck der Systeme bestehe nicht im Erhalten endgültiger Ergebnisse, sondern in der Interaktion mit ihrer Umgebung. Durch seine vielfachen Ein- und Ausgaben und Abhängigkeiten untereinander gleiche ein reaktives System bildlich betrachtet zudem eher einem Kaktus als einer Black Box-Einheit [HP85].

Halbwachs [Hal93] versteht zusätzlich unter reaktiven Systemen, daß sie kontinuierlich auf ihre Umwelt reagieren in einer Geschwindigkeit, die von dieser Umwelt bestimmt wird. Reaktive Systeme bestehen aus Hard- und Software. Wichtige Eigenschaften von reaktiven Systemen sind **Erfüllung von Echtzeitanforderungen**, ein

nebenläufiges und **deterministisches Verhalten** sowie das Ziel, die **Verläßlichkeit** des Systems zu garantieren. Diese Eigenschaften werden im Folgenden näher vorgestellt.

Echtzeitsysteme unterscheiden sich von Nicht-Echtzeitsystemen insofern, als daß die Spezifikation des Echtzeitsystems zeitliche Vorgaben (sogenannte Deadlines) für die Erfüllung der einzelnen spezifizierten Aufgaben enthält. Echtzeitsysteme haben also auf Eingaben in einer außerhalb des Systems definierten Zeitbedingung zu reagieren (vgl. [Ber89, Liu00]). Die Art, wie vorgegebene Zeitfenster eingehalten werden, definiert, ob ein System **harten** oder **weichen Echtzeitanforderungen** genügt.

Harte Echtzeitanforderung bedeutet, daß jede Nichteinhaltung eines spezifizierten Zeitfensters als Fehlverhalten aufgefaßt wird. Die Korrektheit des Systems hängt also davon ab, ob die geforderten Ergebnisse nicht nur logisch richtig sind, sondern auch, ob die Ergebnisse innerhalb des spezifizierten Zeitintervalls geliefert werden (vgl. [Sta88]). Harte Echtzeitsysteme garantieren also nicht nur die Erfüllung der spezifizierten Aufgaben, sondern auch ihre Abarbeitung innerhalb des vorgegebenen Zeitfensters. Alle sicherheitsrelevanten Systeme im Automobil oder der Luft- und Raumfahrttechnik müssen harten Echtzeitanforderungen genügen, definiert beispielsweise in [ISO09a].

Bei **weicher Echtzeitanforderung** wird das Nichteinhalten einer Zeitanforderung toleriert und nicht als Fehlverhalten gewertet. Beispielhaft hierfür sind Telekommunikationsanwendungen wie Videokonferenzen, bei denen das Nichteinhalten von Zeitanforderungen sich in einer Verminderung der Übertragungsqualität der Audio- und Videodaten auswirkt.

Nebenläufigkeit eines Systems besagt, daß das System aus einer Menge paralleler Komponenten besteht, die zusammenarbeiten, um ein beabsichtigtes Verhalten zu erreichen [Hal93]. Die gleichzeitige Ausführung der Komponenten setzt aber voraus, daß keine der Komponenten auf Berechnungen oder Ereignisse anderer Komponenten warten muß, da sonst ein Deadlock entstehen kann [CES71].

Weiterhin erwähnt Halbwachs zwei Gründe für die beabsichtigte Nebenläufigkeit: zum einen können solche Systeme somit parallel zu ihrer Umwelt laufen und zum anderen ermöglicht dies, den Einsatz reaktiver Systeme auf verteilten Architekturen aus Gründen der Geschwindigkeit oder Fehlertoleranz [Hal98].

Determinismus des reaktiven Systems besagt, daß die Ausgaben des Systems vollständig bestimmt sind durch die Eingabewerte des Systems und den Zeitpunkt der Eingabe [Hal93]. Dies bedeutet, daß gegeben ein System mit bestimmten Eingabewerten sowie dem Zeitpunkt der Eingabe die Ausgabewerte des Systems bestimmbar sind.

Die im späteren Abschnitt 2.2.1 definierte Fehlerterminologie baut auf der Definition des Determinismus auf, in dem von der erwarteten Ausgabe abweichende Ereignisse als fehlerhaftes Ereignis bzw. Versagen des Systems aufgefaßt werden.

Streben nach Zuverlässigkeit wird von Halbwachs als vielleicht wichtigste Eigenschaft von reaktiven Systemen angegeben, da reaktive Systeme Aufgaben in Domänen übernehmen, in denen Fehlverhalten teure oder für den Menschen gefährliche Folgen haben können [Hal93]. Deshalb seien die meisten kritischen Systeme entweder reaktiv oder enthielten reaktive Anteile [Hal98].

Der Begriff Zuverlässigkeit wird verschieden aufgefaßt. Eine im Bereich eingebetteter Systeme recht häufig verwendete Definition von Avižienis et al. versteht unter Zuverlässigkeit die Fähigkeit, Dienste zu liefern, denen zu Recht vertraut werden könne [ALRL04]. Zuverlässigkeit sei weiterhin die Fähigkeit eines Systems, Versagen von Diensten zu vermeiden, die häufiger oder schwerer sind als akzeptierbar [ALRL04]. Die Definitionen von Avižienis werden im späteren Verlauf der Arbeit auch für die mit der Zuverlässigkeit verbundene Fehlerterminologie verwendet.

Weitere, zum Teil unterschiedliche, Definitionen des Begriffes sowie eine genaue mathematische Definition der Zuverlässigkeitsfunktion finden sich in den Abschnitten 2.2.1 bzw. 2.5.

2.1.2 Anforderungen automobiler Steuergeräte

Tabelle 2.1 zeigt eine vergleichende Übersicht der Einflußfaktoren auf die Automobilbranche und die verwandten Branchen Verbraucherelektronik sowie Luft- und Raumfahrt. Wie die Tabelle darlegt, gibt es Gemeinsamkeiten der Automobildomäne mit den beiden anderen. Besonders hervorzuheben -und für die Dissertation relevant- ist jedoch, daß die Automobilindustrie sowohl sicherheitsrelevante Komponenten mit strengen Sicherheitsanforderungen besitzt, als auch einem enormen Kostendruck aufgrund der hohen Stückzahl der Steuergeräte unterliegt.

Einflußfaktor	Automotive	Avionik	Verbraucherelektronik
Wettereinfluß	mittel bis hoch	hoch	niedrig
Kostendruck	hoch	niedrig	hoch
Stückzahl	hoch	niedrig	hoch
Sicherheitsrelevanz	hoch	hoch	eher niedrig
erforderliche Verfügbarkeit	hoch	hoch	meist niedrig
Länge Lebenszyklus	hoch	hoch	kurz
Variantenvielfalt	hoch	niedrig	hoch
Systemkomplexität	hoch	hoch	niedrig bis hoch

Tabelle 2.1: Branchenübergreifender Vergleich wichtiger Einflußfaktoren.
Quelle: eigene Darstellung

Wettereinfluß

Aufgrund ihres Einsatzgebiets sind Automobile und ihre Komponenten dem Einfluß von Witterungsbedingungen wie Schnee, Regen oder Hagel sowie einem Temperaturbereich von -40° bis +140° Celsius ausgesetzt. Der Wettereinfluß auf das Auto ist somit zwar nicht größer als der in der Luft- und ganz besonders der Raumfahrttechnik, aber größer als der in der Verbrauchselektronik. Diese starken Umwelteinflüsse erfordern

den Einsatz robuster und somit teurere Bauteile als in der Verbraucherelektronik, um durch Wetter verursachte Schäden oder Fehler zu vermeiden.

Kostendruck und Stückzahl

Becker legt dar, daß seit Anfang der 2000er Jahre ein verschärfter Preis- und Kostenwettbewerb stattfindet, der alle Teilnehmer der automobilen Wertschöpfungskette betrifft. Wirtschaftliche Krisen, stagnierende oder sinkende Realeinkommen sowie Arbeitsplatzangst führten in den großen Automobilabsatzmärkten wie Europa oder USA zu sinkender Nachfrage nach Autos und somit zu rückläufigen Absatzzahlen und Ertragsrückgängen der Automobilhersteller. Die Automobilhersteller versuchten dies mit Preis-/ bzw. Rabattangeboten auszugleichen, was den Kostendruck auf die Hersteller erhöhte und den Wettbewerb nur mehr verschärfte [Bec05, Kapitel 2]. Da sich aber trotz Preiskämpfen die Absatzmenge nicht erhöhte, stieg der Margendruck für die Automobilhersteller und ihre Zulieferer. Dieser Druck zur Reduzierung der Stückzahlkosten betrifft auch stark die Steuergeräte.

Steuergeräte bestehen aus Soft- und Hardware. Da die Kosten für Vervielfältigung einmal erstellter Software recht gering und somit vernachlässigbar sind, dominieren die proportionalen Herstellkosten der Hardware den Stückpreis eines Automobilsteuergeräts, wie [SZ10, S.171ff] aufzeigt. Dies führt dazu, daß vorwiegend Druck auf die Reduzierung der proportionalen Hardware-Herstellkosten ausgeübt wird. Dies wirkt sich einerseits im Einsatz von Hardware mit begrenzten Ressourcen aus, andererseits aber auch darin, Bauteile komplett einzusparen oder gemeinsam zu nutzen.

Der Zwang zur Reduzierung der Herstellkosten hat starke Auswirkungen auf die Diagnose und ihren Umfang. Im Gegensatz zur Luft- und Raumfahrttechnik sind dem Einsatz von redundanter Hardware und zusätzlicher Meßtechnik für die Diagnose Kostengrenzen aufgrund der vielfach höheren Stückzahlen eines Automobils gesetzt.

Ein Vergleich der Auswirkungen der Einsparungen eines Sensors verdeutlicht den Unterschied. Ein Einsparen eines Sensors bei einem Automobilsteuergerät bei 1 Millionen Stückzahl ermöglicht Einsparungen von mehreren Millionen Euro. Diese Summe ist ein Vielfaches verglichen mit den Einsparungen eines Sensors bei einem Flugzeug. So hat beispielsweise das meistverkaufte Flugzeug der Firma Airbus, der A320, eine produzierte Gesamtstückzahl von 6681 [Air10].

Steigende Software-Innovationen

Die Entwicklung von (Software-)Innovationen für Automobile ist aufgrund der speziellen Anforderungen der Domäne kostspielig. Dennoch besteht trotz Kostendiktats der Zwang, Innovationen für das Fahrzeug zu entwickeln. [Bec05, Kapitel 2.2.2] legt dar, daß die Verweigerung zur stetigen Einführung von technischen Neuerungen zwar kurzfristige Einsparungen bringt, jedoch durch das Verpassen wichtiger technologischer Entwicklungen den langfristigen Unternehmenserfolg gefährde.

Becker [Bec05, Abb.29 auf S.80] zitiert aus [MI03, S.14ff] den auf die OEM einwirkenden Innovationsdruck und unterteilt die Technologien im Automobilbereich in *Must-have-*, *Nice-to-have-* sowie *Nischentechnologien*.

Must-have-Technologien sind meist stark kundennutzwertorientiert und erreichen schnell einen sehr hohen Markanteil. Beispiele hierfür sind Komponenten, die aufgrund gesetzlicher Anforderungen entwickelt wurden, wie Airbags, ABS oder ESP. Als *Nice-to-have-Technologien* werden hochpreisige Komponenten wie Navigationssysteme oder Klimaanlagen verstanden, die vor allem für Premiumhersteller von Interesse sind. Sie setzen sich zwar in den oberen Marktsegmenten durch, benötigen jedoch einige Zeit bis zu einem tieferen Marktanteil. „Diese beiden Innovationskategorien beeinflussen die ... [Automobil-]Industrie nachhaltig“ [MI03, S. 16]. Dagegen sei der Einfluß von *Nischentechnologien* zu vernachlässigen, da die Nachfrage danach „vom Anforderungsprofil eines kleinen Kundensegments“ [MI03, S. 16] abhänge.

[MI03, S. 15ff] stellt weiterhin fest, daß die Marktposition und die Wettbewerbsfähigkeit der Automobilhersteller stark von diesen beiden ersten Kategorien abhängen. In der Einleitung wurde der steigende Anteil der Software an den Innovationen des Fahrzeugs beschrieben. Software-Innovationen ermöglichen den Herstellern das Erreichen der beschriebenen Wettbewerbsvorteile durch Differenzierung von ihren Wettbewerbern sowie die Erfüllung von steigenden Kundenwünschen in den Kategorien *Must-Have* und *Nice-to-have*. Zudem sind Software-Innovationen um so höher einzustufen, da sie durch ihre deutlich geringeren Vervielfältigungskosten ökonomische Vorteile gegenüber mechanischen Neuheiten haben.

Für die Diagnose relevant ist ein weiteres Ergebnis der Studie, daß nämlich beginnend mit der steigenden Marktdurchsetzung von Innovationen aus beiden Kategorien Produktmerkmale wie Qualität und Zuverlässigkeit in den Vordergrund geraten ([MI03], [Bec05, S.81]).

Sicherheitsrelevanz

Die Übernahme von sicherheitsrelevanten Funktionen im Fahrzeug durch Steuergeräte birgt die Möglichkeit, daß Fehlverhalten des Steuergeräts zu gefährlichen Konsequenzen für die Insassen des Fahrzeugs oder für die Umwelt des Fahrzeugs führen können. Deshalb unterliegen Steuergeräte mit sicherheitsrelevanten Funktionen speziellen strengen Anforderungen, die in Standards wie [IEC98, ISO09a] definiert sind.

Hohe Verfügbarkeit

Die hohe Verfügbarkeit ist ein Faktor, der sich aus der Zuverlässigkeitsfunktion ableitet und für die Sicherheitseinstufung relevant ist. Je gefährlicher Ausfälle eines Systems sein können, desto höher ist die notwendige Zuverlässigkeit des Systems. Sowohl Automotive als auch Avionik sind Domänen mit höchsten Anforderungen an Verfügbarkeit.

Die in [ISO09a] definierten *Automotive Safety Integration Level* (ASIL)-Stufen definieren je nach Sicherheitseinstufung des Systems einen Mindestwert den die Zuverlässigkeitsfunktion erreichen muß. Dabei stellt ASIL A die geringsten Sicherheitsanforderungen und ASIL D die höchsten dar. Je nach ASIL-Stufe werden verschiedene Anforderungen gestellt, die das System erfüllen muß. Der Begriff Verfügbarkeit wird genauer in Abschnitt 2.2.1 definiert.

Langer Lebenszyklus

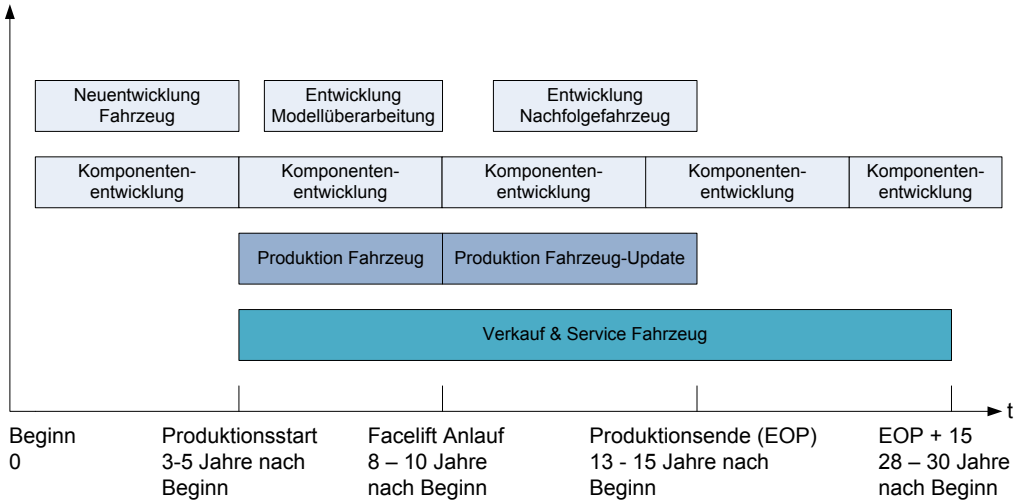


Abbildung 2.1: Übersicht über den Lebenszyklus eines Automobils.
Quelle: eigene Darstellung, basierend auf [SZ10, S.21]

Abbildung 2.1 zeigt den Lebenszyklus eines Automobils beginnend mit der Neuentwicklung eines Fahrzeugs durch den Automobilhersteller. In diesem Prozeß wird festgelegt, welche Anforderungen und Features das Fahrzeug erfüllen soll und aus welchen Komponenten das Fahrzeug bestehen wird. Gleichzeitig und teilweise unabhängig vom Fahrzeugentwicklungsprozeß verläuft der Komponentenentwicklungsprozeß, der vielfach vollständig von einem Zulieferer übernommen wird. Die teilweise Unabhängigkeit entsteht, da Komponenten fahrzeugübergreifend eingesetzt oder von einem Zulieferer auch an andere OEM verkauft werden.

Nach dem Entwicklungsprozeß beginnt der Produktionszeitraum mit Produktionsstart (SOP) und anschließend die Markteinführung des Fahrzeugs. Kurz nach Markteinführung und Produktionsanlauf startet die Entwicklung der Fahrzeugmodellüberarbeitung. Die Fahrzeugmodellüberarbeitung wird auch als Lebenszyklusimpuls oder Facelift bezeichnet (vgl. [Rau11, Kap. 3.2.3]). Neben Design-Änderungen des Fahrzeugs werden auch Verbesserungen wie beispielsweise technische Updates oder neue Versionen der verwendeten Komponenten eingesetzt. Hier bietet sich für einen OEM die Möglichkeit, aus den im Felde vorkommenden Fahrzeugerfahrungen zu lernen und so die Qualität der Modellüberarbeitung zu erhöhen. Der Nutzen dieser Daten erhöht sich zudem, da nach Serienanlauf der Modellüberarbeitung des Fahrzeugs der Entwicklungsprozeß für das Nachfolgefahrzeug startet.

Die mit dem Produktions- und Komponentenanlauf startende Serienbetreuung beinhaltet die Bereitstellung von Ersatzteilen, Wartungs- und Reparaturleistungen durch den OEM und (Vertrags-)Werkstätten. Diese Pflicht leitet sich aus dem *Leistung nach Treu und Glauben*-Paragraphen des Bürgerlichen Gesetzbuches (BGB) [Bun10, §242] ab. Die Verfügbarkeit von Ersatzteilen und die Betreuung ermöglicht die Erhaltung der Funktionsfähigkeit des Fahrzeugs für den Kunden über den erwarteten Lebenszyklus hinaus und wird vom Automobilhersteller auf bis zu insgesamt 15 Jahre

nach Produktionsende der betreffenden Baureihe gewährt. Dadurch ergibt sich ein Gesamtlebenszyklus von bis zu 30 Jahren für eine Fahrzeugproduktlinie.

Besonders wichtig ist, daß durch die ständige Weiterentwicklung der Komponenten sowie durch die Modellüberarbeitung und den Beginn der Entwicklung des Nachfolgefahrzeugs der Nutzen des Lernens aus den Vorfällen im Feld als sehr hoch eingeschätzt werden muß (vgl. [Edl01, Koh06]). Aufgrund der verschiedenen Produktlebenszyklen können diese Informationen aus dem Feld für den weiteren Verbau der Komponenten zeitnah eingesetzt werden. Dies ermöglicht die Erstellung von Qualitätsregelkreisen [Edl01, Kapitel 5.2] bei denen aus Fehlervorfällen gelernt wird, um so Fehler zu beheben und Wiederholfehler zu vermeiden.

Variantenvielfalt

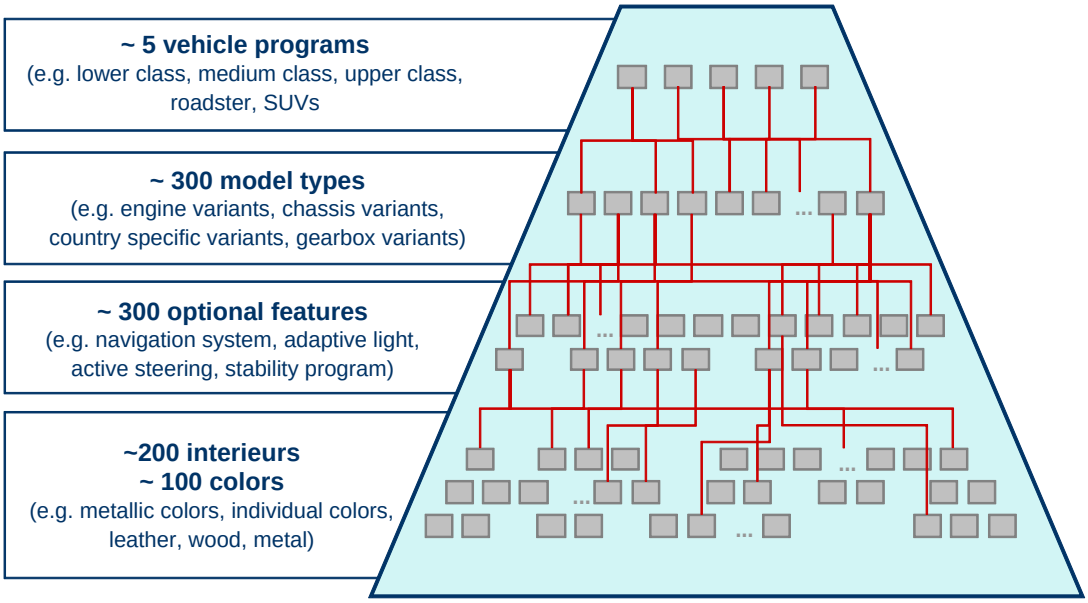


Abbildung 2.2: Hohe Anzahl möglicher Fahrzeugvariationen. Quelle: [Sch02, S. 19]

Im Gegensatz zur Luft- und Raumfahrttechnik verfügen Automobile über eine enorme Vielfalt an Varianten, wie Abbildung 2.2 beispielhaft zeigt. Diese Vielfalt resultiert aus verschiedenen angebotenen Baureihen/Modellen und ihren Varianten sowie vielfältigen Sonderausstattungen. Das Ziel der verschiedenen Sonderausstattungen ist eine Anpassung des Angebots der Automobilhersteller an die verschiedenen kundenspezifischen Wünsche, um somit eine möglichst große Marktabdeckung erreichen zu können (vgl. [Sch02, S. 12ff], [Kud04, Kap. 1]).

Obwohl sich viele Ausstattungen gegenseitig ausschließen (bspw. Cabrio und Panoramadach) entsteht doch eine enorme Anzahl an verschiedenen Versionen, die die Komplexität des Gesamtfahrzeugs enorm erhöhen. So ergab eine exemplarische Untersuchung des italienischen Automobilmarktes im Jahr 2006, daß im Durchschnitt 5,1 Modelle pro Hersteller mit wiederum durchschnittlich 12,2 Versionen pro Modell angeboten werden [VS08, Tabelle 4.4].

Hier sind besonders die Sonderausstattungen zu nennen, die bis kurz vor Produk-

tionsbeginn durch den Kunden geändert werden können oder sogar teilweise nach Produktion des Fahrzeugs nachgerüstet werden können. Je nach Sonderausstattungskombination können bei vorhandenen Komponenten zusätzliche Funktionsumfänge freigeschaltet werden. Beispielhaft hierfür ist die Variantenvielfalt des Schiebe- und Hebedachs, die vom Fahrzeugtyp (Coupé, Touring, Langversion, ...) und Sonderausstattungsumfang (Satellitenradio mit Dachantenne, Telematik-Dienste mit Antenne, Alarmanlage, ...) des Fahrzeugs beeinflußt wird.

Diese Erweiterungsmöglichkeiten müssen schon in der Entwicklungsphase der Komponenten berücksichtigt werden und im Steuergeräte-Code vorgehalten werden.

Die Variantenvielfalt bedeutet für die Diagnose, daß die Interaktionen des Fahrzeugs und all seiner möglichen Ausstattungen überwacht werden müssen. Hier ist besonders problematisch, daß viele Interaktionen nicht bekannt sind und diese aber Fehler auslösen können. Beispielhaft hierfür ist, daß ein Steuergerät A Nachrichten bzw. Werte innerhalb einer Nachricht von einem Steuergerät B empfängt, die A nicht kennt und deshalb nicht weiterverarbeiten kann, oder gar als ungültiges Signal wertet und einen Fehlerspeichereintrag setzt.

2.1.3 Funktionsweise automobiler Steuergeräte

Ein *Steuergerät* (ECU) ist ein eingebettetes, reaktives System, das aus einem Mikroprozessor mit Software-Funktionen, Sensoren, Aktuatoren sowie Sollwertgebern besteht. Durch die Verbindung und Kommunikation mit anderen Steuergeräten über Bussysteme ergibt sich ein verteiltes System. Beispielhafte Modelle für den Aufbau eines Steuergeräts finden sich in [EAD10, Abbildung 8] oder [SZ10, Kapitel 2.3].

Das Modell für den Aufbau des Steuergeräts wird für die Beschreibung der technischen Architektur von Steuergerätefunktionen (Abschnitt 2.1.5) benötigt, sowie für die in der *Failure mode and effects analysis* (FMEA, vgl. [FME80, DIN06]) durchgeführten Untersuchung der möglichen Hardware-Fehler (Abschnitt 4.3).

Zudem wird dieses Modell in Abschnitt 4.6 mit dem Deployment-Modell für die Speicherung der on-board Diagnoseumfänge gekoppelt. Im Folgenden werden die wichtigsten Bauteile kurz skizziert, für eine vollständige, genauere Darstellung der einzelnen Teile sei auf [Rei06, Rob07a, Rob07b, SZ10, WR11, Rei11] verwiesen, aus denen die folgenden Kurzbeschreibungen der Bauteile entnommen sind.

„**Sensoren** setzen eine physikalische oder chemische (meist nicht elektrische) Größe ... in eine elektrische Größe ... um“ [Rob07b, S.110]. Sensoren helfen also dem Steuergerät bei der Erfassung von Daten außerhalb des Gerätes und ermöglichen so zusammen mit den Aktuatoren die Interaktion mit der Umwelt. Durch die Sensoren kann das Steuergerät Werte aus der Umwelt erfassen, so beispielsweise die Temperatur oder Werte bzw. Zustände, die durch Aktuatoren anderer Steuergeräte entstanden. Im Automobilbereich wird eine Vielzahl von Sensoren eingesetzt, wie [Rob07a, S. 208 - 353] zeigt.

Aktuatoren sind elektromechanische Geräte wie beispielsweise Stellmotoren. Sie werden vom Steuergerät angesteuert und „setzen die Stellinformation tragenden Signale geringer Leistung in leistungsbehaftete Signale einer zur Prozessbeeinflussung notwendigen Energieform um“ [Rei11, S. 376], wie bspw. mechanische Arbeit. Mit

Hilfe der Aktuatoren ist das Steuergerät in der Lage, aktiven Einfluß auf seine Umwelt zu nehmen. Ein Beispiel hierfür ist der elektrische Stellmotor des Fensterhebers, der über ein Kommando angesteuert das Glasfenster bewegt. [Rob07a, S.353 - 365] sowie [Rei11, S. 376 - 387] stellen verschiedene Aktuatoren der Automobildomäne vor.

Benutzerschnittstelle und Sollwertgeber ermöglichen dem Fahrer direkten Einfluß auf das Steuergerät. Beispiele für in jedem Fahrzeug verbaute Sollwertgeber sind Fahrzeugstarter, Gaspedal und Handschaltknüppel. Bekannte Benutzerschnittstellen zum Fahrzeug sind zudem die Fensterheberschalter oder die Knöpfe und Drehregler der Fahrzeug-Entertainmentsysteme.

Bussignale. Die Einführung der ersten leistungsfähigen Bussysteme wie beispielsweise des CAN-Busses [Bos91, ISO94b] in den frühen 1990er Jahren ermöglichte eine zeitnahe Kommunikation der Steuergeräte sowie eine kostengünstige gemeinsame Verwendung von Sensoren. Bussysteme übertragen die physikalischen **Bussignale** als Nachrichten in Form eines spezifizierten Protokolls, das aus den Nutzdaten sowie Verwaltungsinformationen wie Prüfsumme oder Adressat besteht. Die zeitnahe Übertragung von Werten war vorher nur mittels teurer, einzelner **Stichleitungen** zwischen zwei Komponenten möglich. Eine kurze Übersicht über Bussysteme findet sich in [WWP04, NHB05], eine deutlich detailliertere in [ZS08].

Pin bezeichnet einen äußeren Anschluß eines Steuergeräts.

Sternkoppler wird verwendet, falls eine sternförmige, Punkt-zu-Punkt-Verbindung zwischen mehreren Komponenten gewünscht wird. Der Sternkoppler ist dabei als einzelne Einheit mit allen zu verbindenden Komponenten verbunden (Sterntopologie). Sternkoppler werden beispielsweise für den FlexRay-Bus eingesetzt (vgl. [WR11, S. 226] sowie [ZS08]).

2.1.4 Steuergeräte als steuerungs- und regelungstechnische Systeme

Die Regelungstechnik befaßt sich mit der Steuerung *dynamischer Systeme*, das heißt „Systeme, deren wichtigste Kenngrößen sich zeitlich ändern und deshalb als Funktionen der Zeit dargestellt werden. Dabei wird zwischen Eingangsgrößen, die auf das System einwirken und zeitliche Veränderungen innerhalb des Systems verursachen, und Ausgangsgrößen unterschieden, die das Verhalten des Systems als Reaktion auf die Eingangsgrößen beschreiben...Unter *Steuerung* wird dabei die zielgerichtete Beeinflussung eines dynamischen Systems bezeichnet“ [Lun10a, S. 2].

„Das *Regeln* ist ein Vorgang, bei dem fortlaufend eine Größe, die Regelgröße (die zu regelnde Größe) erfaßt, mit einer anderen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflußt wird. Kennzeichen für das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungskreis des Regelkreises fortlaufend sich selbst beeinflußt“ [DIN94].

Abbildung 2.3 stellt ein Blockschaltbild für ein Regelungssystem dar. Aufgabe der Regelung ist es, die zu regelnde Größe **X** ständig an einen durch **W** vorgegebenen Wert anzugleichen. Diese Aufgabe wird erschwert durch Störgrößen **Z**. Um diese Aufgabe zu erfüllen, wird **X** fortlaufend überwacht, beispielsweise durch Sensoren, und mit der Führungsgröße **W** verglichen. Häufig handelt es sich bei den Führungsgrößen durch vom Fahrer vorgegebene Sollwerte **W***, die über Sollwertgeber übertragen werden.

Abhängig von diesem Vergleich muß **X** an die Führungsgröße angeglichen werden. Die Angleichung wird von der Regelung gesteuert, indem Befehle in Form von Ausgangsgrößen **U** an die Aktuatoren weitergeleitet werden. Die Aktuatoren setzen dies über die Änderung von Stellgrößen **Y** durch. Das Ergebnis der Angleichung wird wieder über Sensoren in Form von Meß- oder Rückführungsgrößen **R** erfaßt, wodurch sich ein geschlossener Regelungskreis ergibt [SZ10, S.38ff.].

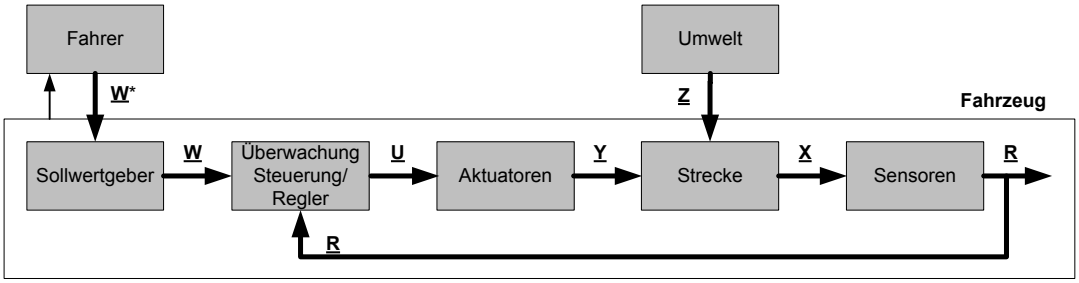


Abbildung 2.3: Steuergeräte als Steuerungs- und Regelungssystem in Fahrzeugen. Quelle: [SZ10, Abbildung 2-1]

Regelungssysteme werden im Automobilbereich sehr häufig eingesetzt. Hierbei übernehmen Steuergeräte vollständig und autonom die Aufgaben der Überwachung und der Regelung durch Kontrolle der Regler. Im Folgenden wird dies nur kurz anhand eines Fallbeispiels dargelegt, für eine genauere und tiefere Darstellung sei beispielsweise auf [Lun10b, Teil III] verwiesen.

Beispielhaft sei hier das ACC erwähnt, das den Abstand des Fahrzeugs zu Vorderfahrzeugen durch Regelung der Fahrzeuggeschwindigkeit (**X**) konstant hält. **W*** stellen die Einflußgrößen des Fahrers auf das ACC dar. Hierbei handelt es sich um den gewünschten Abstand zum Vorderfahrzeug sowie die vom Fahrer selbst bestimmte Geschwindigkeit des Fahrzeugs. Die Steuerung des ACC erfaßt die aktuelle Geschwindigkeit des Fahrzeugs, den per Radar bestimmten Abstand zum Vorderfahrzeug sowie andere Daten wie eingelegter Gang. Anhand des gemessenen Abstands und der aktuellen Geschwindigkeit wird vom Regelungssystem berechnet, wie schnell das Fahrzeug fahren muß (**U**), um den vom Fahrer gewünschten Abstand zu halten. Dieser Wert wird an die Aktuatoren weitergeleitet, die das Fahrzeug auf die erforderliche Geschwindigkeit bremsen oder beschleunigen (**Y**). Bei Fahrzeugen mit Automatikgetriebe wird zudem ein passender Gang eingelegt. Die angepaßte Geschwindigkeit wird über Sensoren neu erfaßt (**R**) und an die Überwachung weitergeleitet. Mögliche Größen, die sich störend (**Z**) auf das Regelungssystem auswirken können, sind beispielsweise Windverhältnisse sowie die unabhängige Geschwindigkeit der Vorderfahrzeuge.

Das ACC-Fallbeispiel zeigt auf, daß Steuergeräte sicherheitsrelevante Umfänge des Fahrzeugs überwachen und regeln. Da die Regelung vollständig und vom Fahrer unabhängig erfolgen kann, kommt der Überwachung solcher Systeme eine enorm wichtige Aufgabe zu. Hierzu leistet die Diagnose einen wichtigen Beitrag (vgl. Abschnitt 2.4.4).

2.1.5 Verteilte Steuergerätefunktionen

In diesem Abschnitt wird die Anwendung des Konzepts der verteilten Funktion auf die Automobildomäne dargestellt. Nach grundlegenden Begriffsdefinitionen wird auf die Architektur von verteilten Funktionen eingegangen und anschließend die Vor- und Nachteile dieser Funktionen dargestellt.

Grundlegende Definitionen

Oft wird unter dem Begriff der Funktion eine Spezifikation einer Funktionalität verstanden, die in Form einer Komponente umgesetzt wird. In dieser Arbeit wird jedoch unter dem Begriff einer *Funktion* eine Entität verstanden, die Eingaben zu Ausgaben verarbeitet. Somit haben auch die Hardware-Bauteile ein funktionales Verhalten. Das Verhalten der Funktion wird durch die *Spezifikation* beschrieben. Eine Funktion, wie beispielsweise eine Software-Funktion, stellt somit eine Umsetzung bzw. *Implementierung* der Spezifikation dar.

Unter dem Begriff einer *Komponente* wird in dieser Arbeit die Hardware verstanden, auf der die Funktion gespeichert wird.

Eine *Funktion* ist dann *verteilt*, falls sie verteilt erbracht wird. Dies erfolgt durch die durch Kommunikation mehrerer Teilfunktionen, die auf verschiedenen Komponenten gespeichert sind (vgl. Definition verteilter Systeme in Abschnitt 2.1.1).

Logische und technische Architektur verteilter Funktionen

In Abschnitt 2.1.1 wurde die Steigerung der Leistungsfähigkeit der in Automobilen eingesetzten Prozessoren beschrieben. Der vermehrte Einsatz dieser leistungsstarken Prozessoren zusammen mit der Einführung echtzeitfähiger Bussysteme bietet die Möglichkeit, Funktionalitäten steuergeräteübergreifend zu implementieren.

Hierzu wird eine Funktion in Teilfunktionalitäten zerlegt, die auf mehreren Steuergeräten verteilt sind und über Bussysteme kommunizieren. Als Ergebnis entsteht aus vorher einzelnen, weitgehend autonom agierenden ECU ein Zusammenspiel mehrerer Steuergeräte in einem als Bordnetz bezeichneten Systemverbund. Zur Erledigung ihrer zugewiesenen Aufgaben bedienen sich verteilte Funktionen auch der Funktionsumfänge verbundener Systeme.

Durch die komponentenübergreifende Verteilbarkeit von verteilten Funktionen muß zwischen der logischen und der technischen Architektur unterschieden werden. Die *logische Architektur* beschreibt dabei das funktionale Zusammenspiel der einzelnen Funktionsanteile. Die logische Architektur wird auf die *technische Architektur* abgebildet. Diese Aufgabe übernimmt das sogenannte Deployment-Modell (vgl. Abschnitt 4.6).

Abbildung 2.4 zeigt den technischen Aufbau der Systemarchitektur und den Einfluß von Fahrer und Umwelt auf das Gesamtsystem. Dabei bezeichnet Umwelt alle Einflüsse in der Umgebung des Fahrzeugs, die nicht vom Fahrer ausgehen. Hierbei kann es sich um die Mitfahrer handeln, um elektronische Systeme wie Mobiltelefone oder -wie auch später aufgezeigt- die Testsysteme der Werkstatt. Der Fahrer kann direkt über die erwähnten Sollwertgeber Einfluß auf die gesamte Architektur nehmen. Die

Systemarchitektur besteht aus vielen interagierenden Steuergeräten. Die Steuergeräte können dabei direkt miteinander über Bussysteme verbunden sein oder indirekt über Aktuatoren und Sensoren agieren. Dies geschieht, falls Aktuatoren eines Steuergeräts eine Handlung auslösen, die von den Sensoren eines anderen Steuergeräts eingelesen werden. Die Steuergeräte können aber auch Bauteile gemeinsam nutzen.

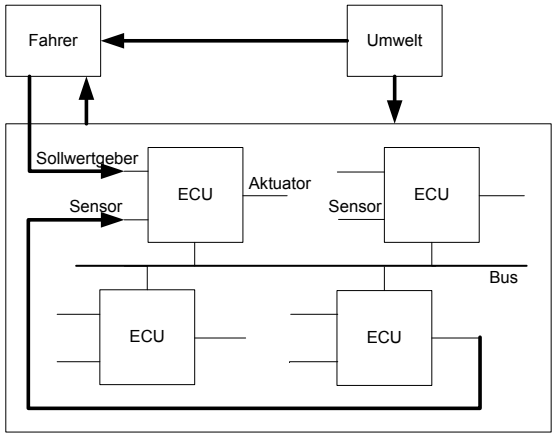


Abbildung 2.4: Übersicht technische Systemarchitektur des Fahrzeugs.
Quelle: basierend auf [SZ10, S.17]

Im Folgenden soll anhand der automatischen Abstandsregelungskontrolle (ACC) die logische und technische Architektur von verteilten Funktionen aufgezeigt werden. Die logische Architektur aus Abbildung 2.5 besteht aus mehreren Teilfunktionen, die sich zur Durchführung der Abstandsregelung über Nachrichtenaustausch koordinieren. Die logischen Teilfunktionen werden auf die technische Architektur in Abbildung 2.6 abgebildet und bedienen sich der Sensoren, Aktuatoren, Sollwertgeber und Bussysteme zur Durchführung ihrer Funktionalitäten. Aufgrund der enormen auszutauschenden Datenmengen sowie Sicherheitsrelevanz des Gesamtsystems werden CAN-Busvarianten verwendet (vgl. [ZS08]). Obwohl es sich insgesamt um fünf Steuergeräte handelt, erscheint das ACC dem Fahrer als einzelnes System. Eine ausführliche Darstellung des ACC und seiner Funktionalitäten bietet [PSST00, WDS09].

Verteilte Funktionen im Automobil verfügen über eine inhärent höhere Komplexität als autonome Funktionen. Dies liegt stark darin begründet, daß verteilte Funktionen sich aus mehreren *nebenläufigen* Funktionen zusammensetzen, die *parallel* ausgeführt werden, voneinander *abhängig* sind und aufgrund *örtlicher Verteiltheit* über Nachrichtenaustausch innerhalb harter Echtzeitanforderungen kommunizieren. Verteilte Funktionen erfüllen also die Eigenschaften verteilter, reaktiver, eingebetteter Echtzeitsysteme (vgl. Abschnitt 2.1.1).

Vorteile verteilter Funktionen

Verteilte Funktionen ermöglichen die **räumliche Verteilung** einzelner Systeme an definierbaren Orten innerhalb des Fahrzeugs. Da die Komponenten miteinander kommunizieren, können durch das Übertragen von Sensorenwerten, bspw. des am rechten Hinterrad erfaßten Geschwindigkeitssignals, andere Steuergeräte diesen Wert

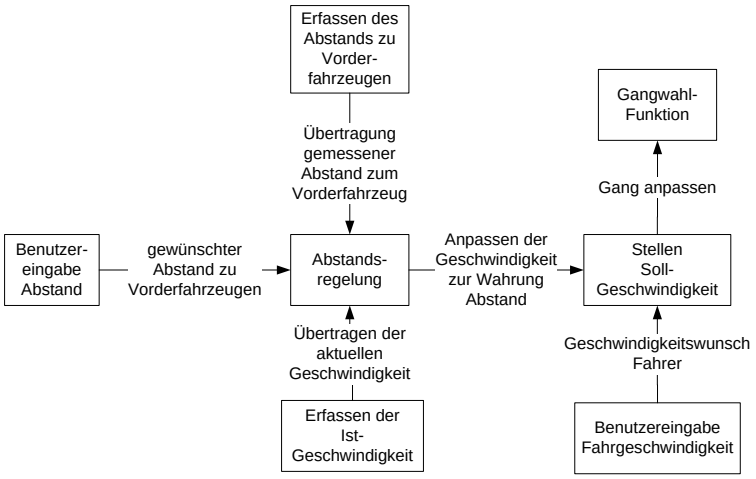


Abbildung 2.5: Logische Architektur der verteilten Funktionalität ACC.
Quelle: eigene Darstellung, basierend auf [WDS09, Kap. 32.4]

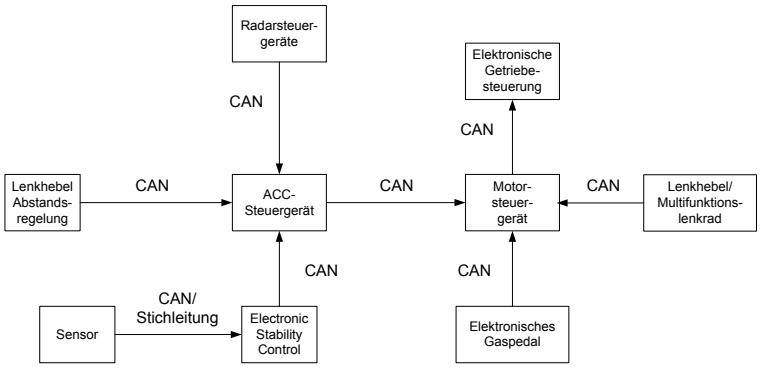


Abbildung 2.6: Technische Architektur der verteilten Funktionalität ACC.
Quelle: eigene Darstellung, basierend auf [PSST00, WDS09]

nutzen, ohne ihn selber mit einem zusätzlichen Sensor zu erfassen. Dies ermöglicht hohe Kosteneinsparungen.

Die in Abschnitt 2.1.2 beschriebene hohe Anzahl an verschiedenen Sonderausstattungen ist viel einfacher durch verteilte Funktionen erreichbar. Besonders die Möglichkeit, das Fahrzeug nach Produktion zu erweitern, läßt sich mittels modularer Funktionen, die über fest definierte Schnittstellen interagieren, erreichen. Das Gesamtsystem ist also durch den Einsatz verteilter Systeme **skalierbarer** und leichter **erweiterbar**.

Wie das Fallbeispiel des ACC zeigt, kann das ACC zur Wahrung des Abstands zu Vorderfahrzeugen das Fahrzeug beschleunigen oder bremsen sowie zusätzlich -bei Automatikgetrieben- den rechnerisch idealen Gang zur Reduzierung des Spritverbrauchs wählen. Um diese Funktionsumfänge durchführen zu können, müssen aber Teilumfänge mit verschiedenen Systemen interagieren. Dies zeigt, daß sich gewisse Funktionsumfänge des Fahrzeugs nur durch die Verteilung von Funktionen erreichen lassen.

Modularer Aufbau und Kommunikation über definierte Schnittstellen in festen Zeitrastern bietet zudem für die Automobilhersteller eine **höhere Unabhängigkeit**

gegenüber Zulieferern, da durch die auf Teilumfänge reduzierte Entwicklungsleistung mehr Zulieferer diese implementieren können und die OEM nicht von einigen wenigen Zulieferern abhängig sind.

Verteilte Systeme ermöglichen durch ihre Topologie eine **höhere Ausfalltoleranz**, da sicherheitsrelevante Funktionen auf mehreren Komponenten verteilt werden können und redundant ausgelegt werden können.

Nachteile verteilter Funktionen

Nachteilig bei verteilten Funktionen ist die inhärent höhere Komplexität des zu betrachtenden Gesamtverbunds, die sowohl die **Entwicklung** als auch die **Koordination zur Laufzeit** der einzelnen Funktionen und des Gesamtverbunds deutlich erhöht.

Die Entwicklung wird erschwert, da schon in der Design-Phase die mögliche Erweiterbarkeit beachtet werden muß und somit die gesamte verteilte Funktionalität modular zu entwickeln ist. Weiterhin müssen zur Erfüllung der vorgesehenen Funktionen die **zeitlichen Anforderungen** der einzelnen Funktionalitäten und des Gesamtsystems definiert werden (vgl. [Sch11]). Die nötige Entwicklungsleistung wird zudem zusätzlich erschwert durch die Tatsache, daß die verteilten Funktionen oft von verschiedenen Lieferanten entwickelt werden.

Während der Laufzeit stellen die **transitiven Abhängigkeiten** der einzelnen Systeme eine große Herausforderung dar. Diese Abhängigkeiten können durch eine **Fehlerpropagation** gefährliche Auswirkungen auf das Gesamtfahrzeug haben. Diese drohende Gefahr für das Gesamtsystem ist um so höher einzuschätzen, als daß diese Abhängigkeiten vielfach unbekannt sind.

2.2 Grundlagen der Diagnose verteilter Systeme

Diagnose stammt ab vom griechischen Wort $\delta\iota\alpha\gamma\nu\omega\sigma\iota\varsigma$ und bedeutet wörtlich übersetzt „durch Erkenntnis“ ($\delta\iota\alpha$ „durch“ und $\gamma\nu\omega\sigma\iota\varsigma$ „Erkenntnis“ [Men93]). Heute wird Diagnose als eine „unterscheidende Beurteilung“ [HDS95] aufgefaßt, so bspw. in der Medizin die Identifizierung einer Krankheit anhand von Merkmalen und Symptomen [Saf92, S. 246].

In der Domäne verteilter Systeme kann die Diagnose auch als die Aufgabe gesehen werden, zu klären, ob ein System sich so verhält wie es spezifiziert wurde. Von der Spezifikation abweichendes Verhalten wird als fehlerhaft gewertet. Abschnitt 2.2.1 stellt die im Rahmen dieser Dissertation verwendete Fehlerterminologie vor und zeigt auf, wie Fehler in verteilten Systemen entstehen können.

Aufgrund einer möglicherweise eingeschränkten externen Sicht auf das System muß die Diagnose als Bestandteil des Systems das Erfassen der Beobachtungen übernehmen. Die Diagnose generiert eine Menge von Beobachtungen, um vorhandene Unterschiede zwischen dem erwünschten spezifizierten Verhalten und dem gemessenen Systemverhalten zu erkennen. Dieser Vorgang wird Fehlererkennung genannt. Die zusätzliche Aufgabe der Diagnose ist dann das als *Inferenz* bezeichnete Schließen auf die Ursache vorhandener Fehler anhand von hinterlegtem Wissen in Form von

Regeln oder Modellen. Die Art der Erkennung des Fehlverhaltens, der Speicherung des Wissens über das System und sein erwartetes spezifiziertes Verhalten sowie das als *Inferenz* bezeichnete Schließen auf die Fehlerursache differenziert die verschiedenen Diagnoseansätze.

2.2.1 Der Fehlerbegriff in verteilten Systemen

„Der Grund war nicht die Ursache, sondern der Auslöser“

(Franz Beckenbauer)

In diesem Abschnitt werden die einzelnen Begriffe der Fehlerterminologie der Arbeit dargelegt. Wie Abschnitt 2.5 vertiefend zeigt, existieren verschiedene Auffassungen hinsichtlich dieser Begriffe.

Verwendet werden im weiteren Verlauf der Arbeit die Fehlerbegriffe aus der Domäne *Dependability* bzw. *Verläßlichkeit*. Die maßgeblichen Begriffe dieser Domäne *failure*, *error* und *fault* mitsamt ihren Definitionen wurden in [Avi67] eingeführt und im Laufe der Zeit mehrfach überarbeitet und aktualisiert, so bspw. in [ALR01, ALRL04]. Im deutschen Sprachraum werden sehr häufig die im Anhang von [Lap92] aufgeführten Übersetzungen dieser Begriffe *Versagen*, *Fehlerzustand* sowie *Fehlerursache* verwendet.

Weiterhin werden in diesem Abschnitt die Zuverlässigkeit und Fehlerbegriffe mathematisch definiert. Die Formeln werden später für die Einordnung der Lebenszyklusdaten in die Methodik der Dissertation benötigt. Da die Lebenszyklusdaten in der Mengengröße differieren, wird in diesem Abschnitt auch das empirische Herleiten der Formeln gezeigt. Eine ausführlichere Definition und Herleitung der in diesem Abschnitt dargestellten Formeln findet sich in [Bir99, Kapitel 1.2.3] sowie in [Pha03].

In Abschnitt 2.1.1 wurde die *Zuverlässigkeit eines Systems* definiert als die Fähigkeit eines Systems, Versagen von Diensten zu vermeiden, die häufiger oder schwerer sind als akzeptierbar [ALRL04]. Dienst bedeutet in diesem Kontext das Verhalten des Systems, so wie es von den anderen Systemen wahrgenommen wird, die mit dem System kommunizieren. Fehler stellen also eine Abweichung von der Zuverlässigkeit dar.

Basierend auf dieser Aussage läßt sich die Zuverlässigkeitsfunktion über den Zeitraum t mit der Menge der funktionsfähigen Einheiten $\overline{\nu}(t)$ aus der Größe der Gesamtmenge N empirisch definieren als:

$$\widehat{R}(t) = \frac{\overline{\nu}(t)}{N}$$

Der Übergang von $\widehat{R}(t)$ zur stochastischen Zuverlässigkeitsfunktion $R(t)$ entsteht durch Bildung des Grenzwerts $N \to \infty$. Mathematisch gesehen läßt sich somit die Zuverlässigkeitsfunktion stochastisch als Komplement der Anzahl der ausgefallenen Einheiten bzw. der Dichtefunktion $F(t)$ der Ausfallrate definieren als:

$$R(t) = 1 - F(t)$$

Auch die Ausfallrate über den Zeitraum t wird empirisch definiert als die Differenz der Anzahl der Einheiten, die zum Zeitpunkt t ausfielen und derer, die bis dahin noch

nicht ausfielen, also:

$$\widehat{\lambda} = \frac{\overline{\nu}(t) - \overline{\nu}(t+\delta t)}{\overline{\nu}(t)\delta t} = \frac{-dR(t)}{dtR(t)}$$

Bildet man den Grenzwert für die Anzahl der Einheiten n, so ergibt sich mit weiteren Umformungen für die Zuverlässigkeit folgende Formel:

$$R(t) = e^{-\int_0^t \lambda(x)dx}$$

Zur Vereinfachung des Terms wird oft angenommen, daß die Fehlerrate über den angenommenen Zeitraum konstant und somit zeitunabhängig bleibt, also daß $\lambda(t) = \lambda$. Daraus ergibt sich für die Zuverlässigkeit folgende Formel:

$$R(t) = e^{-\lambda t}$$

Für die Verteilungsfunktion der Zuverlässigkeitsfunktion $R(t)$ ergibt sich bei konstanter Ausfallrate die aufgrund ihrer Form sogenannte *Badewannenkurve* aus Abbildung 2.7.

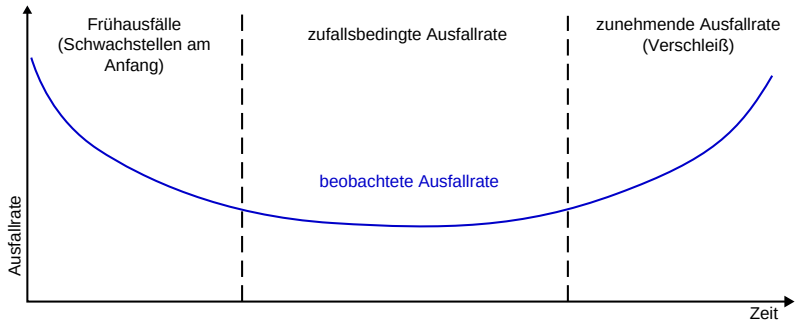


Abbildung 2.7: „Badewannen"-Kurve der Ausfallrate λ als Funktion der Zeit t.
Quelle: [Bir99, Kap. 1.2.3]

Failure bzw. Versagen

Ein Versagen liegt vor, falls das auftretende Verhalten eines Systems oder Dienstes nicht das Verhalten ist, das in der Spezifikation definiert wurde [Lap92]. Diese Definition impliziert, daß das System in Betrieb oder im Starten inbegriffen sein muß und Versagen nur während diesen Zuständen entdeckt werden kann. Falls zudem kein Einblick in das System möglich ist, kann das Fehlverhalten des Systems auch nur mit Versagen beschrieben werden [Bre01, S.11ff].

In der deutschen Sprache kann ein Versagen anhand der Zeitdauer des Versagens unterteilt werden (vgl. [Bre01, S. 11]). Handelt es sich um ein permanentes Versagen, wie beispielsweise aufgrund eines Durchbrennens eines Motors, so kann man von *Ausfall* sprechen. Ein Wackelkontakt hingegen ist reparabel und stellt somit keinen permanenten Fehler dar. Kurzfristiges Versagen wird als *Fehlfunktion* bzw. *Malfunction* bezeichnet.

Der Begriff Versagen ermöglicht eine Metrik zur Bewertung der Fehleranfälligkeit eines Systems, die durch die *Mean-Time-To-Failure* (MTTF)- Formel ausgedrückt werden

kann. Zudem bauen auf dem Begriff Versagen weitere im späteren Verlauf der Arbeit wichtige Größen auf, die im Folgenden neben MTTF dargestellt werden.

Mean Time to Failure (MTTF) bzw. Mittelwert der ausfallfreien Arbeitszeit wird häufig verwendet, um die Verläßlichkeit eines Systems anzugeben. Die MTTF läßt sich für gegebene Größen der Menge der Betrachtungseinheiten N und beobachteten fehlerfreien Zeiteinheiten T_i der Betrachtungseinheiten wie folgt definieren:

$$MTTF = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} T_i$$

Für $N \to \infty$ sowie unter Ausnutzung des Zusammenhangs zwischen der Dichtefunktion des Fehlerauftretens und der Verbindung zur Zuverlässigkeitsfunktion $R(t)$ läßt sich die MTTF wie folgt schreiben:

$$MTTF = \int_0^\infty R(t)dt$$

Die Annahme einer konstanten Ausfallrate λ ergibt dann folgende Vereinfachung:

$$MTTF = \frac{1}{\lambda}$$

Die MTTF gibt also die durchschnittliche bzw. erwartete Zeit bis zum Auftreten eines ersten Fehlers bei nicht reparierbaren Systemen an und somit gleichzeitig die erwartete Zeit bis zum Ableben des Systems ab.

Mean Operating Time between Failures (MTBF) bzw. mittlere Betriebszeit zwischen Versagen unterscheidet sich von der MTTF durch die Annahme, daß Systeme sich reparieren lassen. Ist ein System also wiederherstellbar durch eine Reparatur, wie beispielsweise ein Fahrzeug in der Werkstatt, so spricht man von der **MTBF**, der mittleren Betriebszeit eines Systems zwischen zwei Versagen.

Aufgrund der Beschreibung des gleichen Phänomens werden MTTF und MTBF oft synonym verwendet. Da die Steuergeräte reparabel sind, wird im weiteren Verlauf dieses Abschnitts der Begriff MTBF verwendet.

Mean Time to Recover (MTTR) bzw. mittlere Zeit zur Wiederherstellung beschreibt eine Kennzahl, die für reparierbare Komponenten von Interesse ist. Durch Reparaturen können Komponenten wiederhergestellt werden. Die mittlere Zeit der Wiederherstellung gibt an, wie lange ein Ausfall im Mittel zeitlich dauert, bis das System wieder in einen fehlerfreien Zustand zurückversetzt wird.

Mit der Verteilungsfunktion der Reparaturdauer $G(t)$ läßt sich dann die MTTR wie folgt definieren ([Bir99, S.305]):

$$MTTR = \int_0^\infty (1 - G(t)dt)$$

Verfügbarkeit

[ALRL04] definiert als Verfügbarkeit bzw. *Availability* eines Systems seine Bereitschaft zur Ausführung korrekter Dienste. [Bir99] versteht unter Verfügbarkeit die Wahr-

scheinlichkeit, daß ein Gegenstand unter gegebenen Bedingungen und zu einem bestimmten Zeitpunkt seine verlangte Funktion ausführt.

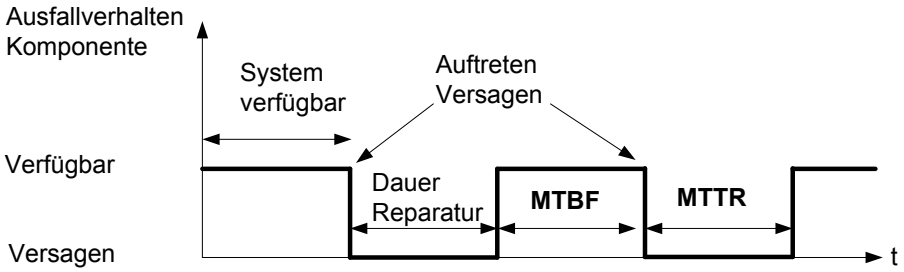


Abbildung 2.8: Zusammenhang zwischen MTBF und MTTR für reparaturfähige Komponenten. Quelle: basierend auf [SZ10, S. 96]

Abbildung 2.8 zeigt den Zusammenhang zwischen Verfügbarkeit, mittlere ausfallfreie Zeit und der mittleren Ausfallzeit. Die MTBF gibt den gemittelten Zeitpunkt zwischen zwei aufeinanderfolgenden Versagen an und die MTTR, wie lange dieses Versagen im Mittel anhält, bis es durch eine Reparatur beseitigt wird.

Die Zeit zwischen diesen Intervallen ist die Zeit, in der das System verfügbar ist. Rechnerisch läßt sich die Verfügbarkeit A also wie folgt bestimmen:

$$A = \frac{MTBF}{MTBF + MTTR}$$

Anhand des Bruches und der Abbildung läßt sich ersehen, daß die Verfügbarkeit eines Systems durch die Steigerung der Zuverlässigkeit (also Vergrößerung der gemittelten Zeit zwischen Fehlern) und/oder durch die Verkürzung der gemittelten Reparaturzeiten erhöht werden kann.

Im nächsten Kapitel wird gezeigt, daß die Diagnose hierzu einen entscheidenden Beitrag zur Erhöhung der Verfügbarkeit leisten kann: einerseits durch Früherkennung und Vermeidung von Fehlern und somit einer Erhöhung des MTBF-Wertes, andererseits durch eine Reduktion der Reparaturzeit mittels einer genauen Identifizierung der Fehlerursache(n) für die anschließende Reparatur.

Fehlerzustand bzw. Error

Das Verhalten eines Systems läßt sich in eine Sequenz mehrerer externer, für die Umwelt sichtbare Zustände des Systems unterteilen. Daraus folgt, daß bei einem Versagen des Systems mindestens einer oder mehrere dieser Zustände des Systems von ihren spezifizierten abweichen [Bre01, S. 12]. Diese Abweichung wird *Error* oder *Fehlerzustand* genannt [ALRL04].

[Bre01, S.12] weist darauf hin, daß aufbauend auf dieser Definition ein Fehlerzustand nur bei zustandsbehafteten Systemen zutreffen kann und sinnvoll sei. Ein Fehlerzustand kann zu einem Versagen führen oder dieses verursachen, der Übergang sei jedoch nicht zwingend.

Ein Beispiel hierfür stellt eine Variable im Programmcode dar. Hat diese Variable einen falschen Wert, kann dieser Wert durch Weiterverarbeitung zu einem Verhalten

führen, das als Versagen aufgefaßt wird. Es kann aber auch sein, daß diese Variable mit einem neuen Wert überschrieben wird, bevor es zu einem Versagen kommen kann.

Fehler bzw. Fault

Als *Fault* bzw. *Fehlerursache* wird die zugesprochene oder angenommene Ursache eines Fehlerzustands bezeichnet [ALRL04]. „Die Fehlerursache ist der eigentliche Unterschied zwischen einem fehlerhaften System und seiner korrekten Fassung“ [Bre01, S. 12].

Für den weiteren Verlauf der Arbeit sind vier Arten von Fehlern interessant, die im Folgenden kurz aufgelistet werden. Je nach Aktivität wird zwischen einem *schlafenden* oder *aktiven* Fehler unterschieden. Verursacht der Fehler einen Fehlerzustand, spricht man von einem aktiven Fehler. Der Fehler kann *intern* im System entstehen oder durch *externen* Einfluß der Umwelt auf das System [ALRL04, Kap. 3.5].

Ein Beispiel für diese Faktoren findet sich in der Auswirkung eines vorhandenen Fehlers in einem Kühlsystem. Solange das Kühlsystem nicht benötigt wird, ist der vorhandene Fehler schlafend. Wird nun das Kühlsystem benutzt, wird der Fehler aktiv und verursacht einen Fehlerzustand. Der Grund für den Fehler des Kühlsystems kann entweder intern sein (beispielsweise durch Verschleiß) oder durch einen externen Einfluß wie beispielsweise einer hohen Umgebungstemperatur, für die das Gesamtsystem nicht ausgelegt war (vgl. [ALRL04, Abb. 12]).

Beispielhaft für Software-Funktionen ist eine fehlerhafte Programmierung einer Programmschleife. Die Fehlerursache der falschen Programmierung der Schleifenterminierungsbedingung führt zu dem fehlerhaften Zustand einer Endlosschleife und zu einem Versagen des Systems, das aufgrund der Endlosschleife nicht in der Lage ist, eine Anfrage fertig abzuarbeiten [ISO09a, Teil X, Abbildung 5].

Ein weiteres Software-Beispiel ist das Verglühen des Mars Climate Orbiter der NASA in der Atmosphäre des Mars. Die Fehlerursache war die verschiedene Verwendung von Meßeinheiten im Navigationssystem und in der Antriebssteuerung, die aber ihre Werte ohne Umrechnung miteinander austauschten. Die Antriebssteuerung rechnete in der europäischen Maßeinheit Newton pro Sekunde, während im Navigationssystem die etwa viermal so große imperiale Einheit Pound-force (lb_f) pro Sekunde verwendet wurde. Dies führte dazu, daß der Climate Orbiter sich dem Mars zu sehr näherte und in der Atmosphäre verglühte [Nat99].

Kausaler Fehlerzusammenhang

Ein kausaler Zusammenhang zwischen den vorgestellten Fehlerbegriffen besteht darin, daß eine Fehlerursache einen fehlerhaften Zustand als Folge haben kann, der wiederum zu einem Versagen des Systems führen kann. Abbildung 2.9 zeigt den Zusammenhang als Graphik.

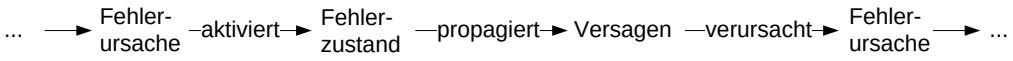


Abbildung 2.9: Zusammenhang Fehlerbegriffe. Quelle: [ALRL04, S.21]

2.2.2 Allgemeine Funktionsweise der Diagnose eingebetteter Systeme

Abbildung 2.10 zeigt eine Übersicht des Ablaufs der Diagnose eingebetteter Systeme.

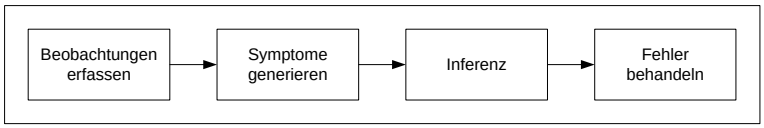


Abbildung 2.10: Übersicht Diagnoseablauf. Quelle: eigene Darstellung

Ein verteiltes System besteht aus Komponenten genannten Bauteilen, auf denen wiederum Funktionen gespeichert sind, die sich zur Erfüllung der Aufgaben des Systems per Nachrichten koordinieren und austauschen (vgl. [CDK05] sowie Abschnitt 2.1.5). Die Funktionen verarbeiten Eingaben zu Ausgaben, die wiederum Eingaben für andere Funktionen darstellen. Das gewünschte Sollverhalten der Funktion wird durch eine Spezifikation beschrieben. Funktionales Verhalten, das von der Spezifikation abweicht, wird als fehlerhaft gewertet.

Die erste Aufgabe der Diagnose von eingebetteten Systemen ist der Vergleich des Ist-Zustands des eingebetteten Systems mit seinem spezifizierten Sollverhalten. Um diese Aufgabe erfüllen zu können, erstellt die Diagnose *Beobachtungen*, die den gegenwärtigen Ist-Zustand des Systems widerspiegeln. Existieren Diskrepanzen zwischen Soll- und Ist-Verhalten, dienen diese Beobachtungen zur Erkennung eines gegenwärtigen **Fehlerzustands**. Dies wird auch als Fehlererkennung bezeichnet. Verschiedene Techniken zur Fehlererkennung wie Replikation, Plausibilisieren oder Paritätsprüfungen werden in [LA90, Kap. 5] beschrieben.

Basierend auf den erstellten und ausgewerteten Beobachtungen bzw. *generierten Symptomen* versucht die Diagnose auf die **Fehlerursache(n)** zu schließen. Dies wird als *Inferenz* bezeichnet. Anschließend erfolgt die *Fehlerbehandlung*. Da Versagen von eingebetteten Systemen auch hohe Kosten oder kritische Auswirkungen auf ihre Umwelt haben können, wird der Diagnose zudem die Aufgabe übertragen, ein **Systemversagen** zu vermeiden. Dies geschieht, indem die Diagnose entweder beim Auftreten des Fehlers oder sogar bei Erkennen (und somit vor Auftreten des Fehlers) Vermeidungsmaßnahmen einleitet.

Da sowohl das Auftreten des Fehlers als auch die Vermeidung des Fehlers das Systemverhalten ändern kann, wird das Eingreifen der Diagnose in das System in dieser Arbeit als Teil der Diagnose betrachtet. Im weiteren Verlauf dieser Arbeit wird dieses Eingreifen als *präventive Diagnose* bezeichnet. Zudem ist darauf zu achten, das durch die präventive Diagnose geänderte Verhalten im Fehlerfall auch in der Spezifikation als Sollverhalten im Fehlerfalle zu erfassen.

Die Diagnose leistet somit einen wichtigen Beitrag zur Erhöhung der Verfügbarkeit des Systems. Zum einen ermöglicht sie durch Hilfestellung zur Problembeseitigung

eine schnellere Reparaturzeit und Reduktion der **MTTR**, zum anderen erhöht die präventive Diagnose durch Fehlervermeidung den **MTBF**-Wert sowie die Sicherheit des Gesamtsystems und potentiell gefährliche Auswirkungen auf die mit dem System interagierende Umwelt.

2.3 Automobile Diagnose

In Abschnitt 2.3.1 wird gezeigt, daß der Funktionsumfang der automobilen Diagnose den der Diagnose eingebetteter Systeme aus Abschnitt 2.2 erweitert. Abschnitt 2.3.2 stellt dar, daß für verteilte automobile Systeme zudem verschiedene Charakteristiken (vgl. Abschnitt 2.1) existieren, die unmittelbaren Einfluß auf die automobile Diagnose sowie ihre Funktionsweise haben. Abschnitt 2.3.3 legt die für den weiteren Verlauf der Arbeit notwendige Klassifikation des Fehlerbegriffs dar. Die Funktionsweise der gesamten automobilen Diagnose zur Laufzeit wird in Abschnitt 2.3.4 detailliert. Abschließend folgt eine Beschreibung des gegenwärtigen Entwicklungsprozeß der automobilen Diagnose in Abschnitt 2.3.5.

2.3.1 Definition und Grundlagen der automobilen Diagnose

Die *allgemeine Diagnose* befaßt sich mit der Erkennung von fehlerhaftem Verhalten von Systemen oder Funktionen und dem Schließen auf die Ursache des Fehlers. Als fehlerhaftes Verhalten wird von der Spezifikation des Systems oder der Funktion abweichendes Verhalten gewertet.

Aufgrund des autonomen Verhaltens von eingebetteten Regelfunktionen bzw. -systemen sowie der Möglichkeit von sicherheitskritischen und kundenrelevanten Auswirkungen möglicher Fehler müssen die Systeme in der Lage sein, entstehende Fehlverhalten durch einen Eingriff in das System zu vermeiden. Da sich durch einen solchen Eingriff das beobachtbare Verhalten des Systems ändern kann, wird in dieser Arbeit die Fehlervermeidung (präventive Diagnose) als Teil der *Diagnose eingebetteter Systeme* gesehen (vgl. Abschnitt 2.2.2).

Fehlerhafte Systeme des Fahrzeugs lassen sich vorwiegend a posteriori in Werkstätten reparieren oder wiederherstellen. Hinzu kommt, daß die Einsicht der Werkstätten in elektronische Systeme beschränkt ist, wie schon in Abschnitt 1.2 erwähnt. Aufgrund dieser eingeschränkten Reparaturfähigkeit ist es notwendig, daß die *automobile Diagnose* die Fehlerbeseitigung in der Werkstatt unterstützt. Dies geschieht indem erkannte Fehler im automobilen System in Form von Fehlerspeichereinträgen (DTC, vgl. [ISO05b]) dokumentiert werden und durch die Zuweisung der Fehlerursache auf eine sogenannte austauschbare Einheit, einer für die Werkstatt zugänglichen Komponente. Die Reparatur in den Werkstätten führt zu einer Unterteilung der automobilen Diagnose in Anteile auf einem Steuergerät sowie den Anteilen der Werkstätten, wie im nächsten Abschnitt detailliert dargestellt wird.

2.3.2 Anforderungen an die automobile Diagnose

Unterteilung des Fahrzeugs in elektronische und mechanische Teile. In Kapitel 2.1.1 wurde beschrieben, daß Automobile aus elektronischen und mechanischen Teilen bestehen, die zusammen agieren, um die Funktionalitäten des Fahrzeugs zu ermöglichen. Die Aufgabe der Diagnose ist die Überwachung der gesamten Anteile des Fahrzeugs.

Die Diagnose elektronischer Teile des Fahrzeugs unterscheidet sich stark von der der mechanischen Teile. Fehler mechanischer Teile können oft sichtbar sein, wohingegen die Black Box-Sicht auf elektronische Teile das Erkennen von Fehlern deutlich erschwert.

Unterteilung der Diagnose in on- und off-board Anteile. Die Diagnose unterteilt sich in einen on-board und einen off-board Anteil. Die genauere Funktionsweise der unterteilten Diagnose wird im nächsten Abschnitt dargestellt.

In diesem Abschnitt wird deshalb nur kurz begründet, wieso die Diagnose aus diesen beiden Anteilen besteht. Die on-board Diagnose kann mechanische Fehler des Fahrzeugs, wie beispielsweise defekte mechanische Bauteile, nicht direkt erkennen. Diese Fehler können nur indirekt von der on-board Diagnose erkannt werden, falls diese Bauteile von Sensoren ausgewertet werden. Zusätzlich gibt es Fehler, die von der on-board Diagnose erkannt, nicht jedoch behoben werden können.

Die beiden erwähnten disjunkten Fehlerklassen können nur off-board in einer Werkstatt behoben werden. Um den Reparaturvorgang für elektronische Teile zu beschleunigen, ist die Werkstatt auf präzise Hinweise der on-board Diagnose angewiesen. Ein gutes Zusammenspiel zwischen on- und off-board Diagnose ermöglicht somit die mittlere Zeit zur Wiederherstellung (MTTR) zu senken.

Eine genauere Klassifikation der möglichen Fehler der Automobildomäne findet sich in Abschnitt 2.3.3.

Gesetzliche Vorgaben. Automobilhersteller haben aufgrund gesetzlicher Vorgaben eine *Gewährleistungspflicht* für ihre verkauften Fahrzeuge mitsamt seinen Komponenten von *zwei Jahren* in der europäischen Union [Eur99] bis hin zu *vier Jahren* in den USA, in der die Werkstätten das Fahrzeug kostenfrei für den Kunden wieder instandsetzen müssen. Der Zeitraum der Gewährleistung wird erweitert durch die Verpflichtung des OEMs und seiner (Vertrags-)Werkstätten bis zu 15 Jahre nach Ende der Serienproduktion des Fahrzeugs Ersatzteile, Wartungs- und Reparaturleistungen bereitzustellen.

In den USA ist die Diagnose im Steuergerät seit der Einführung der *OBD II* 1994 [Cal94] gesetzlich vorgeschrieben und zulassungsrelevant. Die OBD schreibt vor, daß zur Laufzeit des Fahrzeugs die emissionsrelevanten Komponenten des Fahrzeugs auf Funktionsfähigkeit untersucht werden müssen und im Falle eines Fehlers dieser Komponenten der Fahrer mit Hilfe sogenannter OBD-Lampen im Armaturenbrett darauf hingewiesen werden muß.

Die in der ISO 26262 definierten **ASIL**-Richtlinien für die Entwicklung sicherheitsrelevanter Komponenten schreiben nicht direkt eine Diagnose vor. Die ASIL-Richtlinien

geben einen Mindestwert für die Verfügbarkeit des Systems vor. In [ISO09a, Teil II Kapitel 7.4.5] wird gefordert, daß die Hersteller für sicherheitsrelevante Komponenten einen „Field Monitoring"-Prozeß für die funktionale Sicherheit etablieren müssen. Dieser Prozeß schreibt ein Reporting vor über Vorfälle von sicherheitsrelevanten Steuergeräten inklusive der ergriffenen Maßnahmen durch Hersteller und/oder Werkstätten. Weiterhin schreibt [ISO09a, Teil VII Kapitel 6.4.2.1] eine Analyse der gesammelten Daten vor, um sicherheitsrelevante Probleme zu entdecken. Bei Steuergeräten können diese Anforderungen nur durch die on-board Diagnose erfüllt werden.

Bei diesen auftretenden Mängeln oder einem Versagen des Fahrzeugs übernimmt die Diagnose die zusätzliche Aufgabe, die Fehlerursache des aufgetretenen Versagens oder Mangel einer möglichst kleinen reparierbaren Fahrzeugeinheit zuzuweisen, der sogenannten *austauschbaren Einheit*. Diese Zuweisung ist dem Kostengedanken geschuldet, da der Kunde im Gewährleistungszeitraum aufgrund §437 BGB Anspruch auf Wandlung oder auf Nacherfüllung (§439 BGB) in Form von Reparatur des Fahrzeugs hat [Bun10].

Hohe Stückzahl. Wie in Abschnitt 2.1.2 erwähnt, ergeben sich aus der hohen Stückzahl der Steuergeräte Kostengrenzen und ein Bestreben, die proportionalen Herstellkosten zu reduzieren. Der Zwang zur Reduzierung der Herstellkosten hat starke Auswirkungen auf die Diagnose und ihren Umfang. Im Gegensatz zur Luft- und Raumfahrttechnik sind dem Einsatz von redundanter Hardware und zusätzlicher Meßtechnik für die Diagnose Kostengrenzen aufgrund der vielfach höheren Stückzahl des Automobils gesetzt, wie anhand des Vergleichs des Airbus A320 in Abschnitt 2.1.2 gezeigt wurde.

Variantenvielfalt. Die Variantenvielfalt bedeutet für die Diagnose, daß die Interaktionen des Fahrzeugs und aller seiner möglichen Ausstattungen überwacht werden müssen und schon in der Entwicklung der Diagnose im Programm-Code vorgehalten werden müssen. Hier ist besonders problematisch, daß viele Interaktionen nicht bekannt sind und Fehler auslösen können. Beispielhaft hierfür ist, daß Steuergerät A Signale von Steuergerät B empfängt, die A nicht kennt und somit nicht weiterverarbeiten kann, oder gar als ungültig wertet und einen Fehlerspeichereintrag setzt.

Komplexität Gesamtverbund. Die Auswirkungen der steigenden Komplexität des automobilen Gesamtsystems wurden in Abschnitt 1.2 gezeigt. Dies resultiert aus dem Anstieg der funktionalen Vernetzung des Fahrzeugs sowie der Integration vieler Funktionen auf einzelne Steuergeräte. Der Anstieg der Verteilung bedeutet für die Diagnose aufgrund der funktionalen Abhängigkeiten die Notwendigkeit, eine Gesamtarchitektursicht für die Fehlererkennung und -lokalisierung zu entwickeln. Zusätzlich bewirkt die Integration vieler Funktionen auf ein Steuergerät eine Steigerung der Komplexität der Diagnose, da mehr und umfangreichere Fehlerbilder erkannt werden müssen.

2.3.3 Fehlerklassifikation in der automobilen Diagnose

In diesem Abschnitt wird eine für die automobile Domäne relevante Fehlerklassifizierung vorgestellt, die auf der Fehlerterminologie aus Abschnitt 2.2.1 aufbaut und alle potentiellen Fehler der Automobildomäne erfaßt.

1. **Im Steuergerät entdeckbare Fehler**
   - **1.1 Entdeckbare und im Steuergerät verhinderbare Fehler.** Die on-board Diagnose im Steuergerät erkennt anhand von Beobachtungen und Symptomen, daß sich ein Fehler anbahnt und verhindert den Übergang vom Fehlerzustand in ein Versagen des Systems. Beispiele hierfür sind das Erkennen eines überlaufenden Zählers oder das Erkennen des Überhitzens eines Fenstermotors, der daraufhin abgeschaltet wird, bevor ein Versagen eintreten kann. Es findet also ein Eingreifen oder Einschränken von Funktionalitäten des Fahrzeugs statt, auch ohne zwingende Benachrichtigung des Fahrers.
   - **1.2 Entdeckbare und nicht im Steuergerät verhinderbare Fehler.** Die on-board Diagnose erkennt wie im vorherigen Fall das Anbahnen eines Fehlers, kann aber das Versagen nicht vermeiden, sondern nur Hinweise für eine spätere Reparatur speichern. Beispielhaft hierfür ist das Erkennen von falsch angeschlossenen Bauteilen oder das Erkennen von defekten Kommunikationspartnern durch Auswertung von Busnachrichten.
2. **Nicht im Steuergerät entdeckbare Fehler**
   - **2.1 Mechanische Fehler.** Mechanische Fehler des Fahrzeugs sind nicht direkt durch das Steuergerät entdeckbar. Diese Fehler können nur in der Werkstatt bearbeitet werden.
   - **2.2 Unbekannte Fehler.** Diese Fehler waren zum Zeitpunkt der Entwicklung der Diagnose nicht bekannt. Für diese Fehler gilt, daß aus dem Feedback der Werkstätten gelernt werden muß, sodaß der Fehler erfaßt und mit Reparaturmaßnahmen versehen werden kann (vgl. [Edl01, Koh06]). Ein Beispiel hierfür ist ein Fehler, der nur im Zusammenspiel verschiedener Systemausstattungen oder nur sporadisch auftritt.
3. **„Fehler", die eigentlich Features sind.** Hierunter wird ein Verhalten des Fahrzeugs verstanden, das vom Kunden irrtümlich als fehlerhaft aufgefaßt wird. Bekannte Beispiele hierfür sind das Reversieren eines Fensters im Falle eines erkannten Einklemmens sowie ein (zu frühes) Eingreifen einer Fahrsicherheitsfunktion in das Fahrverhalten. Kunden, die mit dem Verhalten des Fahrzeugs nicht vertraut sind, können dies als Fehler auffassen und in einer Werkstatt das Fahrzeug als fehlerhaft beanstanden, obwohl das Fahrzeug sich wie spezifiziert verhielt. Hier ist es nötig, den Kunden durch die Werkstätten über das Verhalten zu informieren. Dies hilft bei der Vermeidung unnötiger Reparaturen und steigert die Kundenzufriedenheit, da der Kunde das Verhalten des Fahrzeugs besser versteht [MPEE10].

Die Fälle **1.2** und **2.1** zeigen, daß nicht alle möglich auftretenden Versagen des Fahrzeugs vom Steuergerät erkannt werden können. Deshalb ist die Werkstattreparatur notwendig, die idealerweise vom Steuergerät durch gespeicherte Symptome unterstützt wird (Fall **1.2**). Es ist zudem notwendig, den Kunden/Fahrer einzubinden, um

unbekannte Fehler genauer erfassen zu können, wie Fall **2.2** zeigt.

Sporadisch auftretende Fehler sind häufig sehr schwierig reproduzierbar. Um solche Fehler dennoch erfassen zu können, werden für die sporadischen Fehler erweiterte Methoden wie Statistik (vgl. Abschnitt 4.8) sowie das Zusammenspiel aller in der Diagnose involvierten Personen (vgl. Abschnitt 4.2) eingesetzt.

2.3.4 Funktionsweise der automobilen Diagnose

Wie die Fehlerklassifizierung in Abschnitt 2.3.3 zeigt, bedarf es aufgrund der verschiedenen potentiellen Fehler einer Zusammenarbeit zwischen der Diagnose im Steuergerät und der Werkstatt. Diese Zusammenarbeit zeigt Abbildung 2.11.

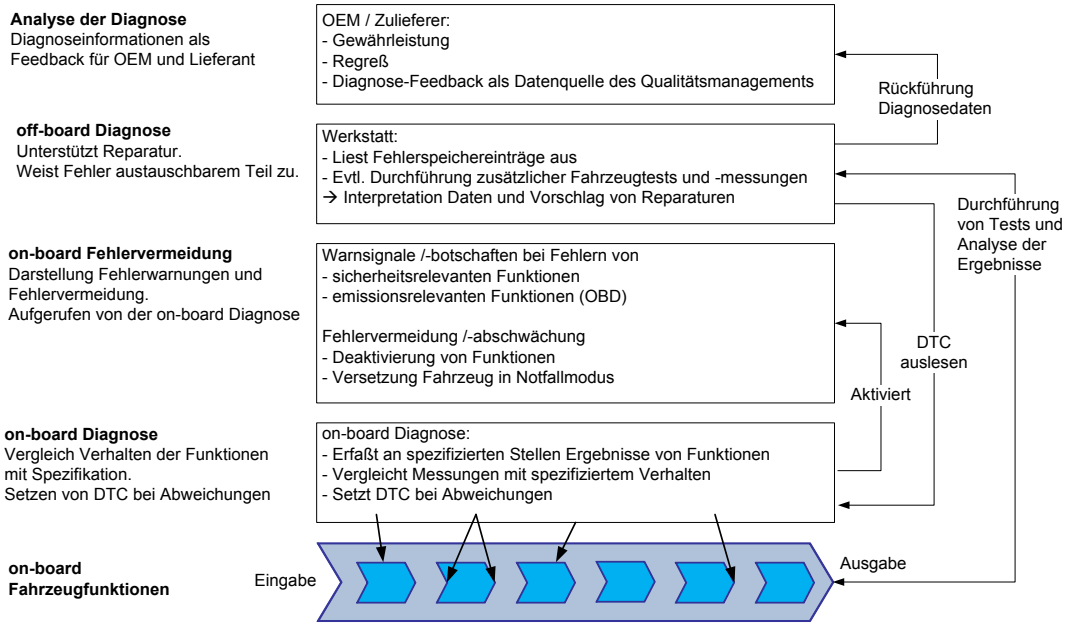


Abbildung 2.11: Übersicht Gesamtprozeß automobile Diagnose. Quelle: basierend auf [BMW08]

Die Funktionen des eingebetteten Systems Steuergerät werden durch die Diagnose im Steuergerät überwacht. Die Diagnose im Steuergerät entspricht dabei dem in den Abschnitten 2.2.2 sowie 2.3.1 dargestellten Ablauf.

Die on-board Diagnose erstellt Beobachtungen und vergleicht diese mit dem Sollverhalten des Systems, um fehlerhaftes Verhalten der Steuergerätefunktionen zu entdecken. Findet die Diagnose Diskrepanzen, startet die Fehlererkennung die Inferenz und anschließend die Behandlung der erkannten Fehlerursachen, die das Auftreten der erkannten Fehlerursache mit einem *Diagnostic Trouble Code* (DTC) [ISO05b] im Fehlerspeicher dokumentiert.

Abbildung 2.12 zeigt den Ablauf der zyklisch durchgeführten Fehlererkennung, wie sie in heutigen Steuergeräten durchgeführt wird. Die Abbildung zeigt, daß die im Bild in blau dargestellten „Normalfunktionen“ des Steuergeräts, das Einlesen, Verarbeiten und Ausgeben von Werten von der Diagnose umschlossen werden. Das Steuergerät prüft, ob es über seinen Busanschluß Nachrichten empfangen hat. Obwohl die

Kommunikation im Automobil meist ereignisgetrieben ist, kommunizieren verteilte Teilfunktionen von sicherheitsrelevanten Funktionen regelmäßig untereinander, um die Funktionsfähigkeit zu prüfen und sicherzustellen. Liegen keine Nachrichten der Teilfunktionen auf dem Bus, so besteht ein Netzwerkfehler, der von den einzelnen kommunizierenden Steuergeräten mit einem DTC gespeichert wird. Ein solcher DTC wird oft auch als sogenannter *Netzwerk-DTC* bezeichnet (vgl. [BMW08]). Sind hingegen die empfangenen Nachrichten abnormal im Sinne von nicht gültig gemäß der Spezifikation, so nimmt das Steuergerät an, daß das sendende Steuergerät einen Defekt hat und setzt einen sogenannten *Ereignis-DTC*. Der Ereignis-DTC besagt, daß das Steuergerät einen Fehler bemerkt hat, der nicht von ihm selber ausgelöst wurde (siehe auch Abschnitt 4.5.3). Nachdem das Steuergerät die Prüfung seiner Kommunikationspartner abgeschlossen hat, überprüft es seine internen Bestandteile auf abnormales Verhalten. Dies beinhaltet die gesamten Bauteile, die zudem weitere Eingangssignale für die internen Funktionen liefern. Die Überprüfung geschieht innerhalb der Funktionen des Steuergeräts, die selber auch wieder auf abnormales Verhalten überwacht werden. Stellt das Steuergerät einen Fehler fest, setzt es einen *Komponenten-*/ bzw. *Funktions-DTC*.

Die Fehlerbehandlung kann zusätzlich erweitert werden durch Fehlerabschwächungs- oder Fehlervermeidungsmaßnahmen. Die Aufgabe dieser Maßnahmen ist die Vermeidung eines Versagens des Systems nach Erkennen eines Fehlerzustands. Dies wird als präventive Diagnose bezeichnet. Zur Erfüllung ihrer Aufgaben kann die präventive Diagnose Fahrzeugfunktionalitäten deaktivieren bzw. abschalten oder das Fahrzeug in einen Notlaufmodus versetzen. Weiterhin kann der Fahrer in Form von Fehlerbenachrichtigungen auf ein Versagen seines Fahrzeugs bzw. im Falle eines Eingreifens der Diagnose auf die bewußte Änderung des Verhaltens des Systems hingewiesen werden.

Aufgabe der off-board Diagnose bzw. Tester (siehe auch [SZ10, Abb. 6-1 und Kap. 6.1]) ist die Unterstützung der Werkstatt bei der Fehlerbehebung durch Identifizierung der kleinsten austauschbaren Einheit (vgl. Abschnitt 2.3.2), die für das Versagen des Systems und seinen fehlerhaften Zustand verantwortlich ist. Zur Identifizierung dieser Einheit baut die Werkstattdiagnose zum einen auf den gespeicherten Symptomen der on-board Diagnose auf (Fehlerklasse 1.2 aus Abschnitt 2.3.3), kann zum anderen aber auch selbständig Tests und Messungen durchführen und analysieren. Die gespeicherten DTC und Beobachtungen der Steuergeräte liest der Tester aus durch Senden von Befehlen an die einzelnen Steuergeräte, die meist im *Unified Diagnostic Services*-Protokoll [ISO06], KWP2000 [ISO99a] oder CAN [ISO04] verschickt werden. Bei mechanischen Fehlern hingegen sind keine Hinweise aus den vorigen beiden Schichten möglich. Hat der Tester die bestimmte Fehlerursache einer austauschbaren Einheit zugeordnet, schlägt er der Werkstatt Reparaturmaßnahmen für diese Einheit vor.

Die in der Werkstatt erfaßten Daten werden an den OEM über spezielle Protokolle zur Auswertung der auftretenden Diagnosefälle im Feld übermittelt (vgl. [Koh06]). Diese Daten liefern dem Qualitätsmanagement des OEM und seinen verantwortlichen Lieferanten aufgrund des Datenumfangs wichtige Feedback-Informationen über die (häufigsten) Probleme im Feld. Zusätzlich helfen die enormen Datenmengen OEM und Komponentenhersteller bei der Optimierung der Reparaturmaßnahmen sowie bei der Bearbeitung und Lösung unbekannter (Fehlerklasse 2.2 aus Abschnitt 2.3.3)

sowie sporadisch auftretender Fehler im Feld.

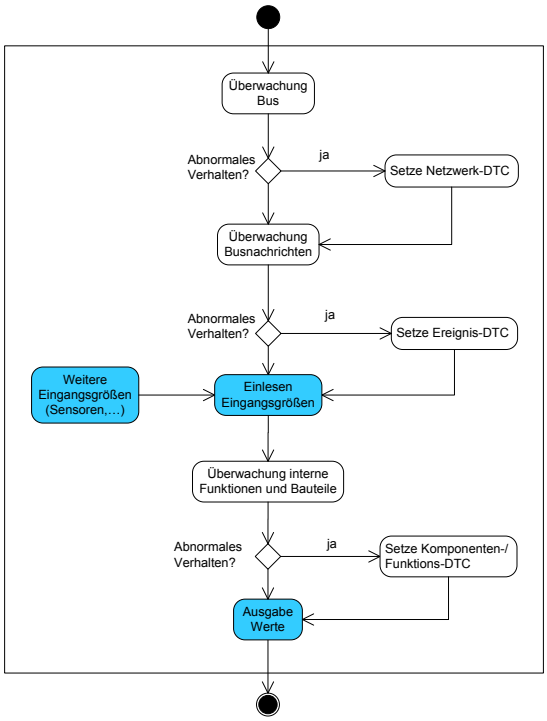


Abbildung 2.12: Ablauf Fehlererkennung im Steuergerät. Quelle: eigene Darstellung

2.3.5 Entwicklungsprozeß automobile Diagnose

Abbildung 2.13 zeigt den gegenwärtigen Diagnoseentwicklungsprozeß mitsamt seinen wichtigsten Prozeßschritten, der sich am V-Modell [Koo92] orientiert. Wie in Abschnitt 2.3.4 dargelegt, bedarf es des Zusammenspiels zwischen on- und off-board Diagnose für die Erkennung und Bearbeitung aller möglichen Fehler.

Da Steuergeräte sehr häufig von Zulieferern entwickelt werden, der OEM jedoch für den *Bordnetz* genannten Gesamtverbund aller Steuergeräte verantwortlich ist, spielt sich die Diagnoseentwicklung des OEM vorwiegend auf der Ebene Systemarchitektur ab. Die Diagnose in der Komponente unterscheidet sich dabei in eine *Systemfunktionsdiagnose* und *Kundenfunktionsdiagnose* (vgl. [BMW08]).

Die Systemfunktionsdiagnose enthält dabei allgemeingültige Diagnosefunktionen, die jedes Steuergerät enthalten muß. Dies beinhaltet zum einen Funktionen, die für die Diagnose selber relevant sind, wie bspw. das Setzen eines DTC oder das UDS-Protokoll [ISO06] für die Kommunikation mit dem Tester in den Werkstätten, sowie die Diagnose von grundlegenden Software-Bausteinen und Bauteilen. Diese Funktionen werden meist den Zulieferern vom OEM in Form von Standardbibliotheken zur Verfügung gestellt.

Die Kundenfunktionsdiagnose betrifft die Diagnose von Funktionen, die für den Kunden sichtbar sind und oft von Lieferanten erstellt werden. Da es sich hierbei um verteilte, komponentenübergreifende Funktionen handeln kann, muß sichergestellt werden, daß die Funktionalitäten in das Diagnosekonzept für das Gesamtfahrzeug

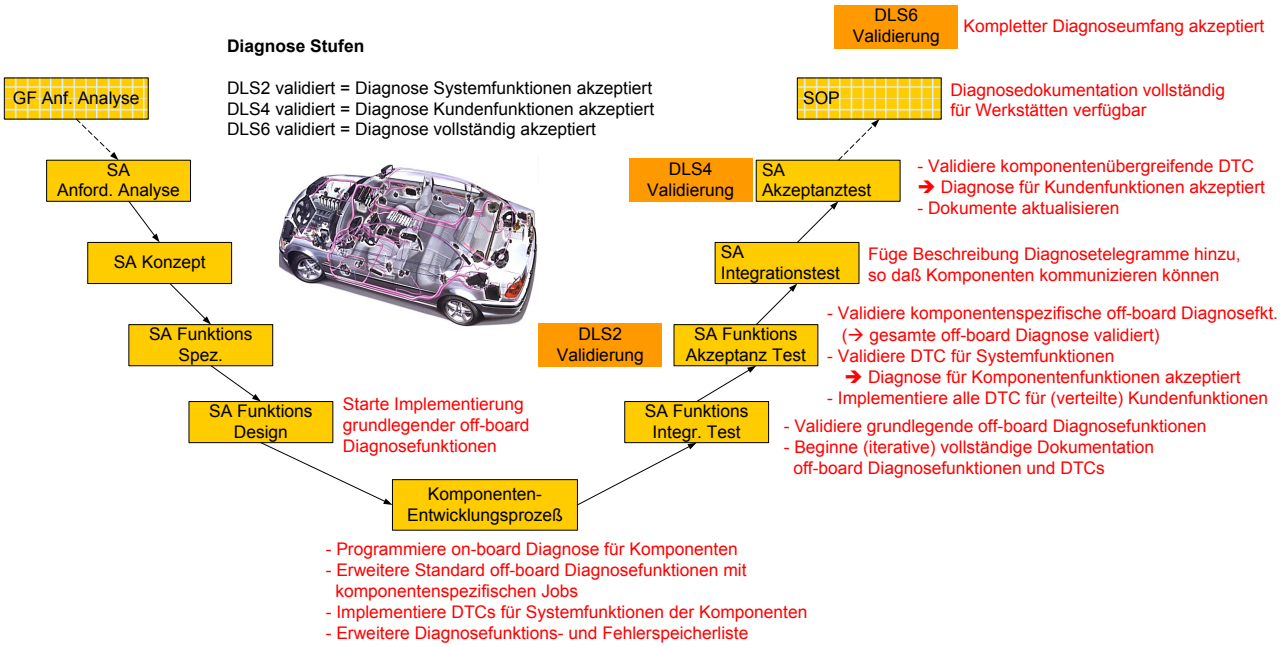


Abbildung 2.13: Ist-Stand Diagnoseentwicklung. Quelle: basierend auf [BMW06, BMW08]

passen. Dies wird durch die in Tabelle 2.2 aufgelisteten generellen Diagnoseanforderungen sichergestellt, deren Erfüllung vom OEM geprüft werden.

Das Wissen um mögliche Fehler und wie diese bzw. das sich durch diese Fehler ändernde Systemverhalten erkannt werden kann, basiert auf Erfahrungswissen von ähnlichen bzw. Vorgängersystemen oder wird durch Analyse des Systems gewonnen. Eine bewährte Methodik zur systematischen Erfassung möglicher Fehler stellt dabei die *Failure Mode and Effects Analysis* (FMEA) dar, auf die im späteren Verlauf der Arbeit genauer eingegangen wird. Die Datenstruktur des erfaßten Fehlerwissens sowie der Ablauf des Schließens auf die Fehlerursache anhand der ausgewerteten Beobachtungen sind abhängig vom verwendeten Diagnoseansatz. Im nächsten Abschnitt werden die wichtigsten Diagnoseansätze der Automobildomäne vorgestellt.

2.4 Übersicht Diagnoseansätze in der Literatur

In diesem Abschnitt werden bekannte Diagnoseverfahren aus der Literatur vorgestellt und hinsichtlich eines Einsatzes in der in Abschnitt 2.3.4 beschriebenen Domäne Automobil diskutiert. Alle Ansätze werden in dem Prozeß aus Abschnitt 2.3.5 entwickelt und folgen dem gleichen, in Abbildung 2.10 beschriebenen Ablauf, unterscheiden sich aber deutlich in der Verarbeitung der erfaßten Symptome sowie der Inferenz auf die Fehlerursache, wie Tabelle 2.3 zusammenfassend am Ende dieses Abschnitts zeigt.

Zuerst werden Diagnoseansätze dargelegt, die auf in Regeln kodiertem *Erfahrungswissen* basieren, bevor dann die *Discrete Event System-Diagnose* vorgestellt wird. Abgeschlossen wird das Kapitel mit *modellbasierten Diagnoseverfahren*.

Allgemeine Anforderungen Komponentendiagnose
Versagen von Steuergerätefunktionen müssen entdeckbar sein und im Steuergerät dokumentiert werden
Kann ein Versagen im Steuergerät rechtzeitig vermieden oder abgeschwächt werden, so ist dies zu tun. Hierzu darf die Funktion auch eingeschränkt oder geändert werden (vgl. safety mechanism [ISO09a, Teil I, Kap. 1.107]).
Da durch ein solches Eingreifen das Verhalten der Funktion geändert wird, ist dieses Sollverhalten im Fehlerfall in der Spezifikation des Systems zu dokumentieren.
Versagen oder Einschränkungen von kundenerlebbaren oder -sichtbaren Funktionen müssen mit Fehlerspeichereinträgen gesichert werden
Fehlerursachen müssen zu austauschbaren Einheiten zuweisbar sein
Reparaturmaßnahmen müssen aus Fehlerspeichereinträgen ableitbar sein
DTC-Setzbedingungen müssen validiert werden
Jeder DTC muß vollständig dokumentiert sein
Dokumentation für on- und off-board Diagnosefunktionen muß vollständig sein

Tabelle 2.2: Grundlegende Lastenheftanforderungen an die automobile Diagnose. Quelle: basierend auf [BMW08]

Eine Diskussion der vorgestellten Ansätze auf ihre Vor- und Nachteile für die Automobildomäne findet sich aufgrund der damit verbundenen Qualitäts- und Kostenpotentiale in Abschnitt 3.7.

2.4.1 Expertenwissen-basierte Diagnose

Ein auf Expertenwissen basierendes System, oft kurz Expertensystem genannt, bezeichnet ein System, das ein Experte in einer existierenden menschlichen Kunst ist und das die Rolle eines beratenden Experten übernimmt [BS84, Vorwort]. Die ersten Expertensysteme erschienen in den 1970er Jahren und stellten einen Paradigmenwechsel in der künstlichen Intelligenz dar. Der bisherige Ansatz eines *general problem-solvers* (vgl. [NS63]) wurde als zu schwach angesehen, um als Basis für die Erstellung hochperformanter Systeme eingesetzt zu werden. Feigenbaum legt weiter dar, daß der beste generelle Problemlöser ein Mensch sei in Gebieten, in denen er ein Spezialist ist und über Spezialistenwissen und -methoden verfügt [FBL70, S. 37]. Dieses Wissen bestehe vorwiegend aus heuristischem Wissen, Erfahrungen sowie aus bewährten Vorgehensweisen anstelle von Fakten. Zudem sei dieses Wissen dem Experten exklusiv vorbehalten, da er nicht in der Lage sei, es zu teilen [Fei77, S. 8].

Expertensysteme setzen an den erwähnten Punkten durch das Kodifizieren von Expertenwissen an. Sie bestehen aus einer *Wissensbasis*, die die kodierten *Regeln* des zu diagnostizierenden Systems enthält. Diese werden in Zusammenarbeit zwischen einem Experten für die Domäne und den für das System verantwortlichen Ingenieuren erstellt. Die kodierten Regeln definieren zudem die Plazierung der Beobachtungen zur Erfassung des Ist-Zustand des Systems und werden zur Laufzeit von einer Inferenz-Engine benutzt, um zusammen mit den nun ausgewerteten Beobachtungen auf die

Lösung (bspw. eines Diagnoseproblems) zu schließen, wie Abbildung 2.14 zeigt. Die verschiedenen auf Expertenwissen basierenden Diagnoseansätze unterscheiden sich durch das für die Inferenz-Engine eingesetzte Verfahren, wie im Folgenden dargestellt wird.

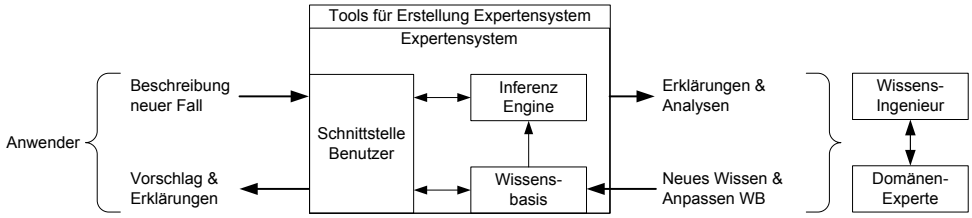


Abbildung 2.14: Aufbau Expertensystem. Quelle: [BS84, S.7]

MYCIN

MYCIN [BS84] ist eines der ersten und zugleich eines der wichtigsten Fallbeispiele für ein Expertensystem. MYCIN wurde entwickelt, um die richtige Dosis von Antibiotika für Patienten mit bakteriellen Krankheiten zu bestimmen. Abbildung 2.15 stellt den Aufbau sowie die Erstellung der Wissensbasis von MYCIN dar.

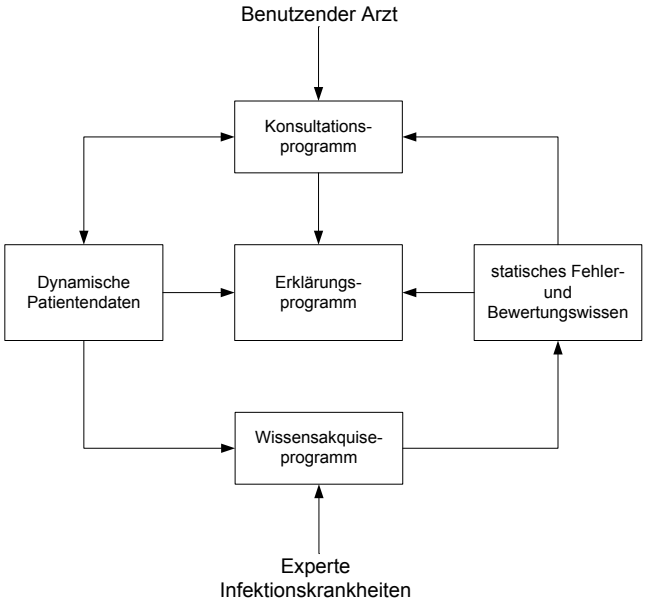


Abbildung 2.15: Aufbau Mycin. Quelle: [BS84, Kapitel 4.1]

MYCIN verwendet eine *zielgetriebene* bzw. *backward chaining* (vgl. [RN10, Kap. 7.5 und 9.4]) Diagnosestrategie für die Inferenz. Ausgehend von einem zu erreichenden Ziel bzw. einer Hypothese wird rückwärts gehend durch Inferenzregeln geprüft, ob Daten vorliegen, die das Ziel erfüllen oder die Hypothese unterstützen. Gesamthaft betrachtet stellt sich das in der Wissensbasis hinterlegte Wissen mitsamt der Inferenzregeln in folgender beispielhafter Form dar:

ERMITTLE C (*ZIEL*)
FALLS B, dann C (*Regel 1*)
FALLS A, dann B (*Regel 2*)
$\therefore$ FALLS A, DANN C (*Implikation*)

MYCIN prüft dann, ob A vorhanden oder wahr ist, um das Ziel zu erreichen. Sind mehrere erreichbare Ziele vorhanden oder besteht ein Unsicherheitsfaktor in einer der Regeln, so werden die Ziele mittels Wahrscheinlichkeiten gewichtet. In MYCIN wird die Wahrscheinlichkeit in Form von *certainty factors* (CF) angegeben.

Regelbasierte Systeme

Im Gegensatz zu dem in MYCIN verwendeten *backward chaining*, gibt es auch die Möglichkeit des *datengetriebenen Schließens* bzw. *forward chaining* (vgl. [RN10, Kap. 7.5 und 9.3]). Der Unterschied zwischen beiden Verfahren ist, daß das zielgetriebene Schließen versucht, eine Annahme zu beweisen, während das datengetriebene Schließen versucht, anhand erfaßter Daten eine Fehlerdiagnose zu bestimmen. Im Automobilbereich werden aufgrund der vielen möglichen Fehler und vieler erfaßbarer Daten vorwiegend regelbasierte Systeme mit *forward chaining* verwendet.

Die Regelbasis und Inferenz stellt sich dann wie folgt beispielsweise dar (vgl. [Hay85], [RN10, Kap. 9.3]):

Regel1: WENN (Temperatur $>$ 200 Grad) DANN Symptom S_1 erkannt
Regel2: WENN (Symptom S_1 erkannt) DANN messe Öldruck
Regel3: WENN (Öldruck $>$ X Pascal) DANN Symptom S_2 erkannt
Regel4: WENN (Symptome S_1 und S_2 erkannt) DANN Ursache U_1 erkannt
Regel5: WENN (Ursache U_1 erkannt) DANN speichere DTC 0x50
Regel6: WENN (U_1 erkannt oder DTC 0x50 gesetzt) DANN Fehler F_1
Regel7: WENN F_1 DANN wende Reparaturmaßnahme *Tausche XX* an

Auch hier besteht die Möglichkeit, Unsicherheitsfaktoren einzubeziehen (vgl. [RN10, Kap. 14.7.1]).

Bayes'sches Netzwerk

Eine Weiterentwicklung des regelbasierten Systems mit Unsicherheitsfaktoren stellt das Bayes'sche Netzwerk dar, das auch unter anderen Namen wie beispielsweise Causal Network, Belief Network oder Ursachen-Wirkung-Netzwerk bekannt ist. Die Notwendigkeit der Bayes'schen Netzwerke läßt sich aus der Tatsache herleiten, daß regelbasierte Systeme Wahrscheinlichkeitsfaktoren nicht richtig einbauen. Russel und Norvig merken an, daß mit Vergrößerung der Regelbasis unerwünschte Interaktionen zwischen den Regeln anstiegen und somit die *certainty factors* von Hand optimiert werden mußten. Weiterhin wurde häufig beobachtet, daß die *certainty factors* durch Übergewichtung von Evidenzen falsche Wahrscheinlichkeitswerte ausgaben [RN10, S.549]. Dennoch sind die Ansätze miteinander verwandt. Heckerman zeigte, daß eine Abwandlung der *certainty factors* äquivalent zu der Bayes'schen Inferenz auf Bäumen ist [Hec85].

Aufbau des Netzwerks. Ein Bayes'sches Netzwerk ist ein gerichteter Graph, in dem jeder Knoten mit einem Wahrscheinlichkeitswert versehen ist. Das Bayes'sche Netzwerk baut sich wie folgt auf ([Pea86], [RN10, Kap. 14]):

1. Eine Menge von Knoten, von denen jeder mit einer Zufallsvariable in Verbindung steht, die diskret oder kontinuierlich sein kann.
2. Eine Menge von gerichteten Pfeilen, die die Knoten verbinden. Führt ein Pfeil von X zu Y, so ist X ein Elternteil von Y. Der Graph ist zyklenfrei.
3. Jeder Knoten X_i verfügt mit $Pr(X_i|Parents(X_i))$ über eine bedingte Wahrscheinlichkeitsverteilung, die die Auswirkungen der Eltern auf den Knoten X_i bestimmt.

Das gesamte Netzwerk stellt dabei eine Menge von bedingten Unabhängigkeitsaussagen dar (vgl. [RN10, Kap. 14.2]), denn Knoten und ihre Zufallsvariablen sind voneinander unabhängig, falls sie im Netz nicht miteinander verbunden sind. Diese Tatsache wird später für die Inferenz benötigt.

Die Konstruktion des Netzwerkes und der Wahrscheinlichkeitsverteilung der einzelnen Knoten basiert auf der *Kettenregel* ([RN10, Kap. 14.2.1], [Pea86, S. 244]):

$$Pr(x_1, ..., x_n) = P(x_n|x_{n-1}, ...x_1)Pr(x_{n-1}, ..., x_1) = \prod_{i=1}^{n} Pr(x_i|x_{i-1}, ..., x_1)$$

Die Konstruktion des Netzes erfolgt dann wie folgt ([RN10, S. 514]):

1. [Knoten.] Bestimme die Variablen $X_1, ..., X_n$, die für das Netz benötigt werden.

2. [Verbindungen.] Für jeden Knoten X_i:

- Bestimme aus $X_1, ...X_{i-1}$ die minimale Anzahl von Eltern für X_i, so daß die Kettenregel erfüllt ist.
- Füge für jedes Elternteil einen Verweis vom Elternteil zu X_i ein
- Trage in die Wahrscheinlichkeitstabelle des Netzes den Wert ein für $Pr(X_i|Parents(X_i))$

Abbildung 2.16 zeigt den beispielhaften Aufbau eines Bayes-Netzwerk für den Zusammenhang zwischen Symptomen S_i, Fehlerursachen U_i sowie Reparaturmaßnahmen M_i.

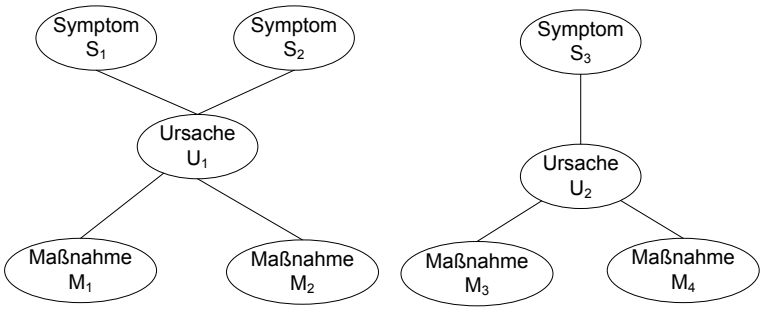


Abbildung 2.16: Beispiel für ein Bayes'sches Netz.
Quelle: eigene Darstellung, basierend auf [Koh06, Abb. 4.2]

Inferenz. Die Inferenz in einem Bayes'schen Netzwerk erfolgt durch Bestimmung der Wahrscheinlichkeitsverteilung für eine Menge von Anfragevariablen X gegeben bestimmte beobachtbare Ereignisse $E_1, ..., E_n$ (Evidenzvariablen). Dabei ist die hierarchische Struktur des Netzes zu berücksichtigen, indem die zwischen den einzelnen Anfragevariablen des Netzes liegenden, nicht beobachteten Variablen y, einbezogen werden. Diese werden oft als *hidden variables* bezeichnet für die gilt $y \in E$. Für die komplette Menge aller Anfragevariablen X ergibt sich also dann $X = \{X\} \cup E \cup Y$.

Der Algorithmus zur Interpretation des Netzwerkes verläuft grob betrachtet wie folgt ([RN10, Kap. 14.4]):

1. Bestimme nächste Anfragevariable X
2. Werte Berechnungsvorschrift $Pr(X|e) = \alpha Pr(X, e) = \alpha \sum_y P(X, e, y)$ aus
3. Generiere Verdacht durch Propagieren der Ergebnisse durch Fehlernetzwerk
4. Fehler gefunden? Falls ja, dann fertig; sonst zurück zu 1.

Für das Fallbeispiel könnte sich beispielsweise die Anfrage ergeben, welche der Maßnahmen M_1, M_2 mit größerer Wahrscheinlichkeit die Komponente reparieren kann, gegeben Symptome S_1, S_2. Für M_1 würde sich beispielsweise folgender Wert ergeben, der mit der Kettenregel berechnet werden kann:

$$Pr(M_1 = true|S_1 = true, S_2 = true) = \alpha Pr(m_1, s_1, s_2, u_1)$$

Beispiele für die Anwendung der kausalen Netzwerke in der Diagnose sind der Windows-Druckerassistent [BH96] und der Office Assistant [HBH+98] sowie im Automobilbereich bspw. [HMDJ08].

Prüfplan bzw. fallbasiertes Schließen

Der Prüfplan unterscheidet sich insofern von anderen Diagnoseansätzen, als daß weder die möglichen Fehler mitsamt Symptomen- wie beim regelbasierten Schließennoch das Verhalten des Systems modelliert werden. Der Prüfplan wird von Experten erstellt und enthält nur bekannte, erarbeitete Diagnosevorfälle mitsamt ihren Lösungen. Das Verfahren entspricht dem **fallbasierten Schließen** bzw. Case-based reasoning (CBR). CBR ist eine Diagnosemethode, bei der die Diagnose auf bereits geschehen, erfaßten Fällen basiert. Für die erfaßten Fälle werden Symptome, die Diagnose(n), eine Erklärung des Zusammenhangs sowie ggf. eine Reparaturmaßnahme gespeichert. Bekannte Ansätze hierfür wurden definiert von [AW91, AP94].

Aufbau. Abbildung 2.17 zeigt, daß der Aufbau eines Prüfplans einem *UML-Aktivitätsdiagramm* (vgl. [Rum05, Kap. 8]) entspricht.

Inferenz. Die „Inferenz" erfolgt dann durch Verfolgen des geplanten Ablaufs des Plans. Wie Abbildung 2.17 zeigt, können im Prüfplan neben Vergleichen auch Messungen oder Reparaturen enthalten sein. Der Prüfplan ist ein Ansatz, der in der Automobilbranche vorwiegend in den Werkstätten verwendet wird. Der Prüfplan für Werkstätten wird meist so erweitert, daß die kompletten Aktivitäten der Werkstätten

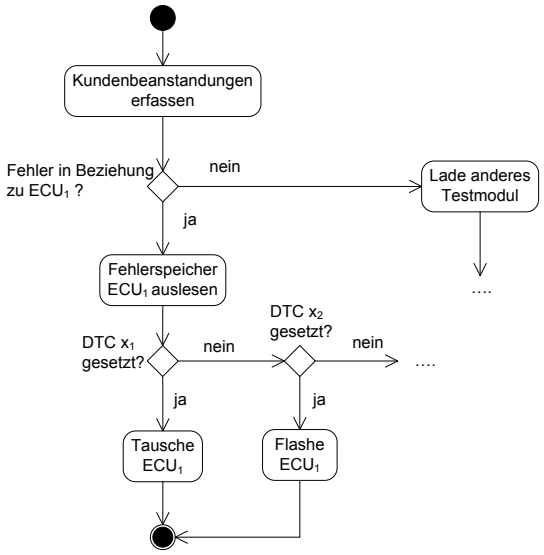


Abbildung 2.17: Beispielhafter Ablauf eines Prüfplans. Quelle: eigene Darstellung

im Fehlerfalle enthalten sind. Dies kann neben Fehlerlokalisierung und -behebung auch die für die finanzielle Abrechnung der Behebung notwendige Befundung umfassen.

2.4.2 Discrete Event System-Diagnose (DES-D)

Discrete Event Systems ist ein hierarchischer, komponierbarer Formalismus zur Modellierung von (dynamischen) Systemen. Der Formalismus basiert auf der Annahme, daß viele Prozesse oder Systeme *ereignisabhängig* sind (bspw. Drücken eines Schalters löst eine Aktion aus) sowie aufgrund ihrer Steuerung durch Computersysteme mit endlichen, *diskreten* Werten arbeiten. Cassandras und Lafortune definieren ein Discrete Event System als ein System, dessen Zustandsraum sich durch eine endliche, *diskrete* Menge beschreiben läßt und bei dem Zustandsübergänge nur zu endlichen Zeitpunkten beobachtet werden [Cas08, Kap. 1.3]. Beispiele für ein Discrete Event System stellen eine Ampelsteuerung einer Kreuzung sowie ein Überwachungssystem eines Prozesses dar. Eine genauere Einführung in Discrete Event Systems findet sich in [Cas08, Kap. 1.3] sowie [Zei00].

Aufbau. Die *Discrete Event System*-Diagnose (DES-D) setzt sich für eine Komponente i wie folgt zusammen (vgl. [SSL+94]):

$$G_i = (X_i, \sum_i, \delta_i, x_{i_0})$$

Hierbei stellt X_i den diskreten, endlichen Zustandsraum dar, δ_i die Zustandsübergangsfunktion, x_{i_0} den Startzustand sowie $\sum_i$ die Vereinigungsmenge der beobachtbaren und nicht beobachtbaren Ereignisse $\sum_o$ und $\sum_{uo}$. Die Ereignismenge und der Zustandsraum beinhalten sowohl normales als auch fehlerhaftes Verhalten der Komponente i. Hierbei wird die Menge der fehlerhaften Ereignisse als $\sum_f$ bezeichnet. Die

Menge der fehlerhaften Ereignisse wird in disjunkte Fehlermengen $\sum_{f_1} \cup \ldots \cup \sum_{f_n}$ aufgeteilt und mit beobachtbaren Ereignissen in Verbindung gesetzt. Die Diagnose besteht dann darin, in einer Spur der beobachteten Ereignisse des Systems die Ereignisse zu finden, die mit Fehlern in Verbindung stehen.

Im Folgenden wird anhand eines relativ einfachen Fallbeispiels das Konzept der DES-Diagnose gezeigt, für eine detailliertere Darstellung sei auf [SSL+94, SSL+96] verwiesen. Das Fallbeispiel modelliert das Plausibilisieren eines Werts. Der Wert wird von einer sicherheitsrelevanten Komponente über ein Bus-System an zwei weitere Komponenten geliefert, die den Wert überprüfen und auf das gleiche Ergebnis gelangen müssen. Das Modell des Gesamtsystems wird im weiteren Verlauf mit M bezeichnet.

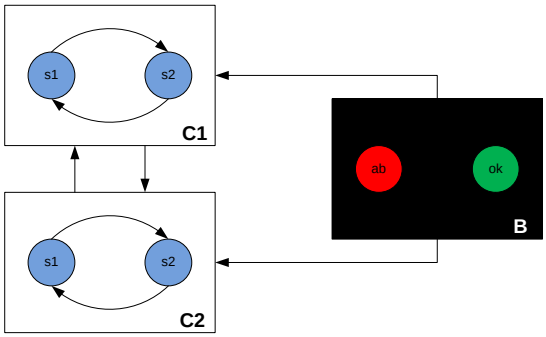


Abbildung 2.18: Fallbeispiel DES-Diagnose

Abbildung 2.18 zeigt eine Darstellung des Systems, das aus den beobachtbaren Komponenten $C1$ und $C2$ sowie aus der nicht beobachtbaren Komponente mit Black Box-Sicht B besteht, deren Ergebnis von $C1$ und $C2$ überprüft wird. $C1$ und $C2$ sind nicht in der Lage, in das Innere von B zu sehen, sondern können nur anhand der über den Bus gesendeten Ausgaben feststellen, ob B in einem normalen ok- oder abnormalen Zustand ist. Ok-Zustand bedeutet, daß die Komponente sich gemäß ihrer Spezifikation verhält und somit plausible Werte sendet. Für das Fallbeispiel in diesem Abschnitt wird angenommen, daß weder das Bussystem noch die Funktionen und Bauteile, die für das Einlesen des Buswertes verantwortlich sind, Fehler verursachen können.

Weiterhin bestehen $C1$ und $C2$ aus Unterzuständen, die mit den logischen Variablen $s1, s2$ gekennzeichnet werden. Basierend auf der Annahme daß $C1$ und $C2$ beobachtbare Komponenten sind, sind sowohl $C1$ als auch $C2$ in der Lage, sich selbst zu diagnostizieren. Für das Fallbeispiel wird weiterhin angenommen, daß sich das Gesamtsystem in einem fehlerfreien Anfangszustand befindet. Dann ergibt sich für den Trace zum Startpunkt 0 folgende mögliche Belegungen:

$$\{[s1, s1, ok], [s2, s2, ok]\}_0,$$

Die weiteren möglichen Belegungen des Traces ergeben sich nun aus den aufgelisteten, logischen Werten der einzelnen, vorhandenen Zustände, beispielsweise

$$\{[s1, s1, ok]\}_0, \{[s2, s2, ok]\}_1, \{[s2, s2, ok]\}_2, \ldots$$

Fehlerentdeckung und -inferenz. Die modellierte Komponente stellt einen Plausibilitätsprüfer dar. Dies bedeutet, daß sowohl $C1$ als auch $C2$ den Wert von B prüfen und zu dem gleichen Ergebnis kommen *müssen*, also jeweils den gleichen Unterzustand haben müssen. Verfügen $C1$ und $C2$ über einen verschiedenen Wert, so befindet sich die Komponente B in einem abnormalen Zustand. Aufgrund der Black Box-Sicht stellt dies auch zugleich die einzige Möglichkeit dar, den Zustand von B zu prüfen. Für diesen Fehlerfall können folgende mögliche globalen Zustände im Trace beobachtet werden:

$$\{[s1, s1, ok]\}_0, \{[s2, s2, ok]\}_1, \{[s2, s1, ab]\}_2, \ldots$$

Die Diagnose des Systems M besteht nun in einer Analyse der *Spur* (Trace) der (globalen) Zustände. Ziel ist die Identifizierung des Auftretens von beobachtbaren Fehlern gegeben eine Spur von beobachtbaren Ereignissen. Dazu wird geprüft, ob der Trace Zustände enthält, die inkonsistent zum Normalverhalten sind. Dies kann in linearer Zeit erfolgen.

2.4.3 Modellbasierte Diagnose (MBD)

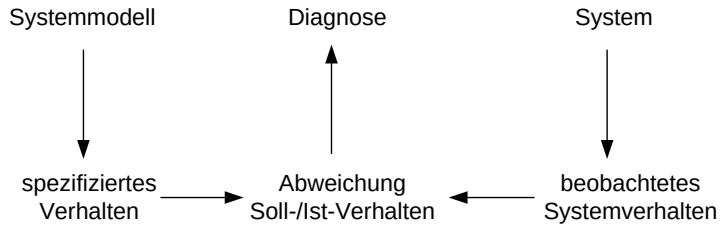


Abbildung 2.19: Aufbau der modellbasierten Diagnose. Quelle: [KW87]

Die in dieser Arbeit vorgestellten modellbasierten Diagnoseansätze setzen das in Abbildung 2.19 beschriebene prinzipielle Verfahren ein. Es wird ein Modell des Systems erstellt, das das gewünschte, spezifizierte Verhalten mittels prädikatenlogischer Variablen beschreibt. Zur Laufzeit wird dann das Modell mit dem tatsächlichen Verhalten verglichen. Dies geschieht durch die Erstellung und Auswertung von Beobachtungen. Werden Diskrepanzen zwischen dem Ist- und Sollverhalten des Systems festgestellt, ist die Aufgabe der Diagnose, die Ursachen für diese Abweichungen zu bestimmen und die für den Fehler verantwortliche(n) Komponente(n) zu identifizieren.

Aufbau. In diesem Abschnitt wird die prinzipielle Vorgehensweisen für die modellbasierte Diagnose dargelegt, wie von Reiter [Rei87] und de Kleer [KW87] 1987 vorgestellt. Beide Ansätze modellieren ein System und sein Verhalten mit Hilfe prädikatenlogischer Formeln.

Die Grundstruktur der modellbasierten Diagnose wird von [Rei87] als Triple $(SD, COMPS, OBS)$ definiert, bei dem
(1) *SD* die Systembeschreibung auf Basis von prädikatenlogischen Formeln,
(2) *COMPS* die mit Konstanten beschriebenen Komponenten des Systems sowie
(3) *OBS* die Menge der Beobachtungen als prädikatenlogische Aussagen
darstellt.

Auf dieser Definition aufbauend stellt die Diagnose für ein System die minimale Menge Komponenten $\Delta \in COMPS$ dar, für die folgende Formel gilt ([Rei87, S. 63]):

$$SD \cup OBS \cup \{AB(c)|c \in \Delta\} \cup \{\neg AB(c)|c \in COMPS - \Delta\}$$

Dabei bezeichnet *SD* das Systemmodell, das aus einer Menge funktionierenden Komponenten $COMPS$ sowie nicht funktionierenden, *ab*normalen Komponenten $AB(c)$ besteht. Zur Laufzeit werden Beobachtungen *OBS* erstellt, um das Verhalten des Systems festzustellen. Diese Beobachtungen dienen als Basis für die Identifizierung der fehlerhaften Komponenten $AB(c)$. Für das Prädikat $AB(c)$ ergibt sich der boolesche Wahrheitswert wahr, falls die Komponente c ein beobachtbares Verhalten zeigt, das nicht seinem spezifizierten entspricht. Alle fehlerhaften Komponenten ergeben zusammen die Menge der fehlerhaften Komponenten Δ.

Fallbeispiel. Im Folgenden wird die modellbasierte Diagnose anhand des Fallbeispiels aus Abbildung 2.20 veranschaulicht. Es handelt sich dabei um ein aus einem Addierer und zwei Multiplikatoren bestehendes Schaltgitter mit fehlerhaftem Verhalten, das an das bekannte Beispiel aus [Rei87, KW87] angelehnt ist.

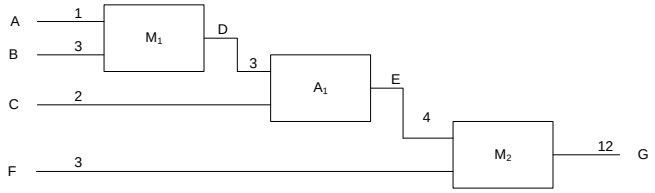


Abbildung 2.20: Fallbeispiel modellbasierte Diagnose. Quelle: eigene Darstellung, basierend auf [KW87, Rei87].

Für das Fallbeispiel ergibt sich folgendes Systemmodell (vgl. [Bau06, S. 108]):

$$SD = \left\{ \begin{array}{l} Mult(x) \wedge \neg AB(x) \Rightarrow (output(x) = input_1(x) \cdot input_2(x)), \\ Add(x) \wedge \neg AB(x) \Rightarrow (output(x) = input_1(x) + input_2(x)), \\ Mult(M_1), Mult(M_2), Add(A_1), \\ input_1(M_1) = A, \\ input_2(M_1) = B, \\ D = output(M_1), \\ input_1(A_1) = D, \\ input_2(A_1) = C, \\ E = output(A_1), \\ input_1(M_2) = E, \\ input_2(M_2) = F, \\ G = output(M_2) \end{array} \right\}$$

Die ersten 2 Zeilen stellen eine formale Beschreibung eines funktionierenden Multiplikators bzw. Addierers dar. Diese Darstellung wird auch oft *Behavioural model* oder *Verhaltensmodell* bezeichnet. Die restlichen Zeilen beschreiben den strukturellen Aufbau des Modells.

Das Fallbeispiel besteht aus den Komponenten $COMP = M_1, M_2, A_1$, für die folgenden Beobachtungen erfaßt wurden:

$$OBS = \left\{ \begin{array}{l} A = input_1(M_1) = 1, \\ B = input_2(M_1) = 2, \\ C = input_2(A_1) = 2, \\ D = output(M_1) = 3, \\ E = output(A_1) = 4, \\ F = input_2(M_2) = 3, \\ G = output(M_2) = 12 \end{array} \right\}$$

Inferenz. In diesem Paragraph wird anhand zweier Algorithmen vorgestellt, wie in der modellbasierten Diagnose das Schließen auf die Fehlerursache anhand der Beobachtungen erfolgt. Es wird dabei sowohl auf die Möglichkeit eines einzelnen Fehlers im Schaltgitter, als auch auf die eines Mehrfachfehlers eingegangen. Im ersten Fehlerfalle arbeitet Addierer A_1 nicht wie erwartet. Im zweiten Fehlerfall sind sowohl der Addierer A_1 als auch der Multiplikator M_1 defekt.

Basis für die Inferenz sind die von de Kleer eingeführten **Conflict Sets** (vgl. [KW87, Kapitel 2.3]). Ein Conflict Set C ist eine Menge von Komponenten $\{c_1, ...c_n\}$ mit $c_i \in COMPS$, so daß gilt

$$SD \cup OBS \cup \neg AB(c_i) | c_i \in C \text{ ist inkonsistent.}$$

Ein Conflict Set C ist genau dann minimal, falls es keine Untermenge von C gibt, die auch ein Conflict Set darstellt. Zur Bestimmung der Konflikte wird ein *hitting Set* eingeführt, für das gilt:
Sei C eine Menge von Mengen. Ein *hitting set* für C ist eine Menge $H \subseteq \bigcup_{S \in C} C$, so daß $H \cap S \neq \emptyset$ für alle $S \in C$. Ein hitting set für C ist genau dann minimal, falls es keine echte Untermenge von C gibt, die ein hitting set darstellt. Dann ist $\Delta \subseteq COMP$ eine minimale Diagnose genau dann wenn Δ ein (minimales) hitting set für die Sammelmenge aller Conflict Sets darstellt [Rei87, Theorem 4.4].

Reiters Diagnosealgorithmus. Sei C eine Menge von Conflict Sets. Es wird dann eine Breitensuche (vgl. [Knu97]) auf C durchgeführt, die beim Knoten $\emptyset$ startet (aus [Bau06, S.111]):

D1. [Nächster Knoten.] Sei C der aktuelle Knoten der Breitensuche.

D2. [Conflict Set?] Prüfe, ob $COMP \setminus C$ ein Conflict Set ist. Falls ja, gehe zu D3. Falls nein, zu D4.

D3. [Eliminiere Nicht-Diagnosen.] Eliminiere alle Knoten C' aus C für die gilt $C' \cap (COMP \setminus C) = \emptyset$, da C' keine minimale Diagnose sein kann.

D4. [Eliminiere Nachkommen.] C stellt eine minimale Diagnose dar. Eliminiere alle Kinder von C.

Das Fallbeispiel kann mit folgender Formel beschrieben werden: $((A \cdot B) + C) \cdot F) = G$. Für die gewählte Variablenbelegung $A = 1, B = 3, C = 2, F = 3$ müßte sich $G = 15$ ergeben. Bei einem funktionierenden System ergäbe sich dann für die Diagnose mit

$\emptyset$ die leere Menge. Für das Fallbeispiel wird aber für G 12 gemessen, das System ist also fehlerhaft. Erste Kandidaten für die Diagnose sind $[M_1, M_2, A_1]$ (oberste Knoten in Abb. 2.21).

Für das erste Fehlerbeispiel wurde angenommen, daß nur der Addierer A_1 defekt ist. Der Algorithmus würde M_1, dann M_2, dann A_1 prüfen. Für A_1 ergibt sich folgende Formel: $Add(A_1) \wedge \neg AB(A_1) \rightarrow output(A_1) = input_1(A_1) + input_2(A_1)$.
Mit den Werten des Fallbeispiels aus Bild 2.20 ergibt sich dann:
$(Add(A_1) \wedge \neg AB(A_1) \rightarrow 4 = 5)$, was einen Widerspruch darstellt. Da A_1 nur eine Komponente darstellt, ist sie gleichzeitig die minimale Diagnose.

Im zweiten Fallbeispiel sind sowohl der Addierer A_1 als auch der Multiplikator M_1 defekt. Es wird angenommen, daß für D der Wert 4 gemessen wurde. Auch hier ergibt sich durch Anwendung des oben beschriebenen Algorithmus zuerst ein Conflict Set mit $[M_1, M_2, A_1]$. Durch das Anwenden der Formeln ergibt sich, daß zwar M_2 funktioniert (da $output(M_2) = E \cdot F$) und M_2 somit aus dem Conflict Set genommen wird, jedoch sowohl die Belegung für M_1 als auch für A_1 einen Widerspruch ergibt. Deshalb stellt $[M_1, A_1]$ das minimale Conflict Set und somit die minimale Diagnose dar.

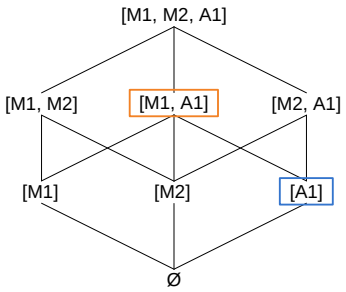


Abbildung 2.21: Suchraum Diagnosekandidaten bei der modellbasierten Diagnose für das Fallbeispiel in Abb. 2.20. Quelle: basierend auf [Rei87]

General Diagnosis Engine. Im Folgenden wird der Algorithmus aus [Bau06, S. 126ff.] dargestellt, der eine Zusammenfassung der textuellen Erklärung der GDE in [KW87] darstellt.
Sei C die Menge der Conflict Sets für ein System $(SD, COMP, OBS)$. Der Algorithmus geht dann schrittweise über alle Variablen oder Parameter aus SD für die Variablenbelegungen bestimmt werden müssen. Im Algorithmus wird ein *assumption-based truth maintenance system* (ATMS) [Kle86] verwendet, dessen Aufgabe es ist, Abhängigkeiten zwischen beobachtbaren Parametern festzuhalten.

D1. [Schließe aus Variablenbelegung.] Stelle für den aktuellen Parameter seine Variablenbelegung gemäß seines durch SD definierten Normalverhaltens fest (bspw. Ein-/Ausgabewerte einer Komponente).

D2. [Erstelle Annahme.] Belege den ermittelten Wert mit Annahmen aus der Systembeschreibung (einschließlich AB-Prädikaten über $COMP$) und füge die Annahmen der ATMS hinzu.

D3. [Vergleiche Werte.] Vergleiche den ermittelten Wert mit dem tatsächlich beobach-

teten Wert aus OBS.

D4. [Diskrepanz erkannt?] Falls eine Diskrepanz zwischen vorhergesagtem und tatsächlichem Wert vorliegt, wird die ATMS verwendet, um vorherige Annahmen bezüglich $\neg AB$-Prädikaten rückgängig zu machen. Füge die betroffenen Komponenten zu C hinzu und aktualisiere die ATMS

D5. [Nächster Wert.] Wiederhole beginnend mit Schritt **D1** bis alle Werte bestimmt sind.

D6. [Bestimme Diagnosen.] Berechne die minimalen hitting sets für die gesammelten Conflict Sets in C.

2.4.4 Fault Detection and Isolation (FDI) in dynamischen Systemen

„Sed quis custodiet ipsos custodes?“

(Juvenal - Satira 6.346-348)

In Abschnitt 2.1.4 wurde aufgezeigt, wie Steuergeräte durch Überwachung und Steuerung Umfänge des Fahrzeugs regeln können. Da dies autonom im Steuergerät geschieht, kommt der Überwachung dieser Umfänge durch Diagnose eine enorm wichtige Bedeutung zu. Die Aufgabe der Diagnose für Regelsysteme ist also den „Wächter zu überwachen“.

Basis für den hier vorgestellten Diagnoseansatz ist die Erfassung des Regelsystems und seines Verhaltens mit einem mathematischen Modell. Wie auch in anderen modellbasierten Systemen werden gemessene Werte mit spezifizierten verglichen, um so Fehlersymptome zu generieren. Abbildung 2.22 zeigt eine graphische Darstellung des Verfahrens. Unterschiedlich zu den bisher dargelegten modellbasierten Ansätzen ist, daß das in diesem Abschnitt dargestellte Verfahren *dynamische Systeme* diagnostiziert (vgl. Abschnitt 2.1.4), deren Verhalten von zeitveränderlichen Werten beeinflußt wird. Somit sind auch Fehlverhalten des Systems über den Zeitverlauf zu überwachen.

Die Diagnose läßt sich also wie folgt mathematisch darstellen [Ise94, Kap. 1.3]:

$$\underline{Y} = f(\underline{U}, \underline{N}, \underline{\theta}, \underline{X})$$

Hierbei bezeichnet **U** die Eingangssignale, **Y** die Ausgangs- bzw. Stellgrößen sowie gegebenenfalls Führungsgrößen **W** und Rückführungsgrößen **R**, **N** im allgemeinen nicht meßbare Störsignale des Prozesses und seiner Stell- und Meßeinrichtungen, **X** interne Zustandsgrößen (Signale) sowie $\underline{\theta}$ Prozeßparameter.

Fehler werden in diesem Ansatz nach zeitlichem Auftreten (abrupt, einsetzend oder aussetzend) sowie abhängig vom Prozeßmodell klassifiziert. Fehler können hierbei *additiv* oder *multiplikativ* sein. Additive Fehler, bspw. durch falsche Sensorenwerte, verändern eine Variable Y durch Addition des Fehlers zu der Ausgangsgröße Y, multiplikative Fehler wirken sich sowohl auf die Eingangsgrößen U und den zu regelnden Prozeß P als auch auf den Fehler aus (vgl. [Ise04, Kapitel 2.1]).

Die Fehlererkennung wird hier nur exemplarisch anhand zweier Verfahren dargestellt, eine genauere Darstellung findet sich in [Ise94, Kapitel 1.3ff]. Die Fehlererkennung versucht dabei anhand der in $U(t)$ zusammengefaßten Eingangsgrößen sowie Ausgangsgrößen $Y(t)$ auf Prozeßfehler zurückzuschließen.

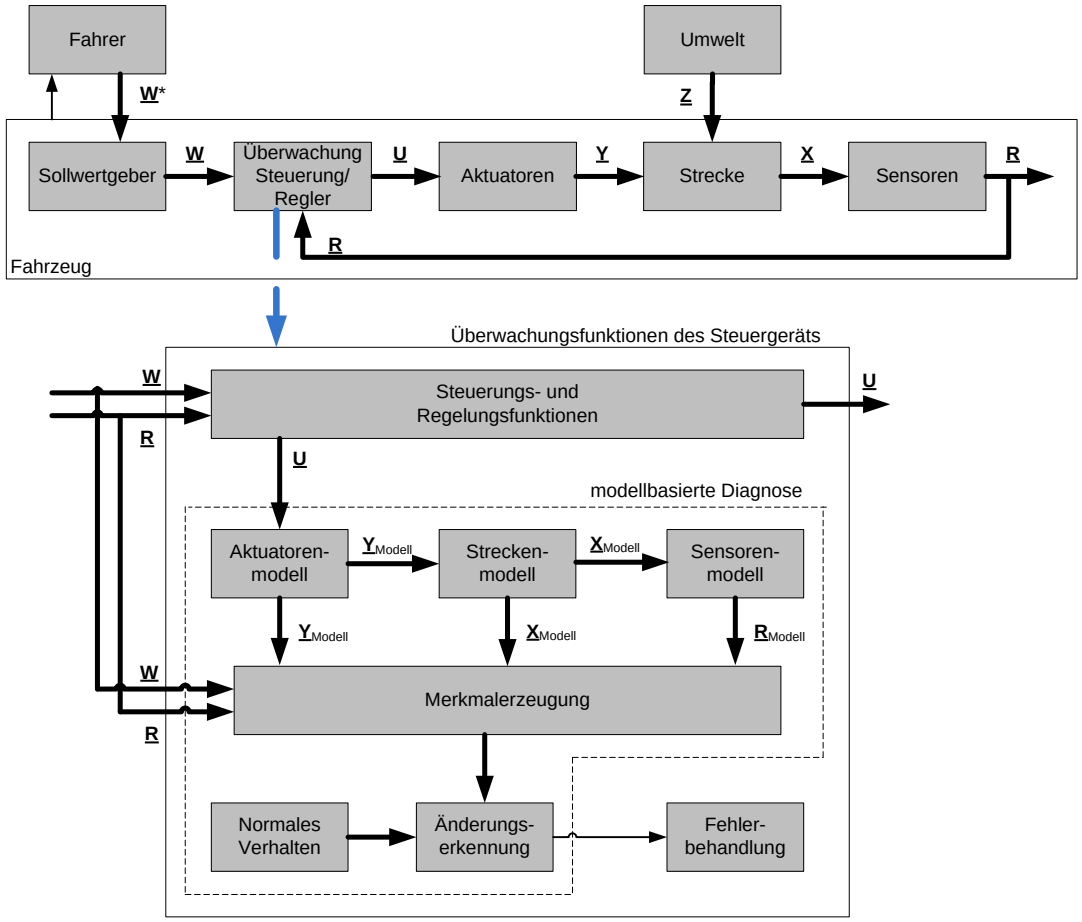


Abbildung 2.22: Übersicht modellbasierte Diagnose für Regelungssysteme. Quelle: [SZ10, S.112], entnommen aus [Ise94].

Fehlererkennung durch Kontrolle direkt meßbarer Signale. Die Kontrolle direkt meßbarer Signale ist eine der bekanntesten und geläufigsten Fehlererkennungsmethoden. Hierbei wird unterschieden zwischen einer Absolut- und Trendkontrolle. Die Absolutkontrolle überprüft dabei, ob ein vorliegendes Signal S innerhalb eines spezifizierten Intervalls liegt [Ise94, Kap. 1.3.1.a]:

$$S_{min} < S < S_{max}$$

Die Grenzen des Intervalls sollten dabei so eingestellt werden, daß genügend Zeit zur Reaktion auf dieses Ereignis bleibt, aber auch so hart sein, daß Störungsmeldungen nicht gehäuft auftreten.

Aufgrund der Schwierigkeit, die Intervallgrenzen zu bestimmen, wird oft auch eine Trendentwicklung über einen definierbaren Zeitraum t überwacht. Je nach Wahl der Zeiteinheit kann so ein sich anbahnender Fehler früher als bei der Absolutkontrolle entdeckt werden. Hierbei wird geprüft, ob die Änderungsgeschwindigkeit $\dot{S} = \frac{dS(t)}{dt}$ des Signals S über den Zeitraum t innerhalb des folgenden Intervalls liegt [Ise94, Kap. 1.3.1.b]:

$$\dot{S}_{min} < \dot{S} < \dot{S}_{max}$$

Fehlererkennung in Prozeßmodellen mittels Parameterschätzung. Im vorigen Kapitel wurde gezeigt, wie Fehler von sich ändernden Signalen entdeckt werden können. Problematisch bei der Fokussierung auf Fehler von Signalen ist, daß oft unklar ist, durch welche Größe des Prozesses das Signal geändert wurde. Somit ermöglicht dieser Ansatz keine detaillierte Fehlerdiagnose.

In diesem Absatz wird ein Verfahren vorgestellt, das es ermöglicht, die Abhängigkeiten des zu überwachenden Prozesses P einzubeziehen und somit eine tiefergehende Diagnose und genauere Fehlerlokalisierung erlaubt. Das Verfahren besteht dabei aus folgenden Schritten [Ise94, Kapitel 1.3.3.1]:

1. Erstellen eines Modells für das statische Verhalten eines Prozesses in Form von Polynomen:
   $Y(U) = \beta_0 + \beta_1 U + \ldots + \beta_q U^q$ mit Ausgangsgröße Y und Stellgröße U.
   Läßt sich das dynamische Verhalten des Prozesses um einen Arbeitspunkt $(Y_{00}|U_{00})$ linearisieren, kann das dynamische Verhalten wie folgt beschrieben werden:
   $y(t) + a_1 y^{(1)}(t) + \ldots + a_n y^{(n)}(t) = b_0 u(t) + b_1 u^{(1)}(t) + \ldots + b_m u^{(m)}(t)$
   mit $y(t) = Y(t) - Y_{00}$ sowie $u(t) = U(t) - U_{00}$.
   Eine vereinheitlichte Darstellung des Prozeßmodells ist:
   $Y(t) = \underline{\psi}_s^T \underline{\theta}_s$ mit Parametervektor $\underline{\theta}_s^T = [\beta_0 \ldots \beta_q]$ und
   $y(t) = \underline{\psi}_d^T \underline{\theta}_d$ mit $\underline{\theta}_d^T = [a_1, \ldots a_n b_0 \ldots b_n]$.
2. Aufstellen der Beziehung $\underline{\theta} = f(\underline{p})$ zwischen den Modellparametern $\underline{\theta}$ und den Prozeßkoeffizienten $\underline{p}$.
3. Schätzung der Parameter $\underline{\theta}$ aus gemessenen Eingangssignalen $U(t)$ und $Y(t)$.
4. Berechnung der Prozeßkoeffizienten $\underline{p} = f^{-1}(\underline{\theta})$
5. Berechnung der Änderungen der Prozeßkoeffizienten $\Delta\underline{p} = \underline{p} - \underline{p}_0$.
   Abweichungen gegenüber den Normalwerten werden als Symptome für Fehler gewertet.
6. Falls keine theoretische Modellbildung und die Berechnung der Prozeßkoeffizienten durchgeführt werden kann, können auch die Modellparameteränderungen $\Delta\underline{\theta} = \underline{\theta} - \underline{\theta}_0$ als Symptome weiter verwendet werden.

Inferenz Die Inferenz erfolgt analog zu der im vorigen Abschnitt dargestellten modellbasierten Diagnose, indem die modellierten Soll- mit den Istwerten verglichen werden. Dies geschieht bei der FDI beispielsweise durch boolesche Aussagenlogik (vgl. [Ise04, Tab. 6]).

Anwendungsbeispiele finden sich in [Ise04, Kap. 4] sowie [NS04].

2.4.5 Diagnose als Boolesches Erfüllbarkeitsproblem

In diesem Abschnitt wird ein Ansatz für die Diagnose vorgestellt, der auf der modellbasierten Diagnose aus dem vorigen Abschnitt basiert, es jedoch ermöglicht, auch andere Diagnoseansätze einzubauen. Der Ansatz wurde zuerst in der Dissertation von [Bau06] definiert und unabhängig davon weiterentwickelt in [Tri09] und für die

Domäne Automobil erweitert und angepaßt in [KB10]. In diesem Abschnitt wird kurz das Verfahren aufgezeigt, eine genauere Darstellung mitsamt den Erweiterungen durch diese Dissertation wird in Abschnitt 4.4.2 vorgestellt.

Aufbau. Das Grundprinzip des Ansatzes ist die Kodierung des Systemmodells mitsamt den erstellten Beobachtungen als formales Modell. Dem statisch definierten Systemmodell werden zur Laufzeit dynamisch erfaßte Beobachtungen hinzugefügt. Anhand der Beobachtungen wird dann geprüft, ob sich das System gemäß der Spezifikation verhält.

Während [Bau06] mit der *Structured Assertion Language for temporal logic* (SALT) eine Erweiterung der linearen temporalen Logik [Pnu77] zur Spezifikation des Systemverhaltens einsetzt, verwendet Tripakis aussagenlogische Formeln zur Systembeschreibung, sogenannte Property Symbols P_i, sowie eine Funktion $\mathcal{P}_i : \mathcal{X} \mapsto \{true, false, ?\}$, die prüft, ob das beobachtete Verhalten x mit $x \in \mathcal{X}$ die gestellte Bedingung bzw. Systemeigenschaft erfüllt und dementsprechend $\mathcal{P}(x)$ setzt. Der Wert $?$ ergibt sich, falls die Bedingung nicht überprüft werden konnte.

Beiden Ansätzen gleich ist dann die Auswertung der Beobachtungen. Die Ergebnisse der Beobachtungen und somit der aktuelle Zustand des Systems werden auf boolesche Variablen transformiert. Die Beobachtungen stehen also in Korrelation mit booleschen Variablen, die entsprechend den Ergebnissen der Beobachtungen gesetzt werden.

Inferenz als Erfüllbarkeitsproblem. Zusätzlich zu den Eigenschaften des Systems wird die Menge aller Fehlerkonfigurationen f definiert. f enthält alle möglichen Fehler $f_1, ...f_n$ des betrachteten Systems. f_i erhält dabei den Wert *true*, falls der Fehler f_i im System vorhanden ist. Die Diagnose $\Phi(p)$ wird dann definiert als Menge aller Fehlerkonfigurationen f, die konsistent mit der gegebenen Systemkonfiguration p sind, also (vgl. [Tri09, Kap. 2.3]):

$$\Phi(p) = \{f | \Phi[f, p] \text{ ist erfüllbar}\}.$$

Das Ziel der Diagnose ist nun die Verarbeitung und Darstellung der Diagnose des Systems zum Zeitpunkt t. Durch die Auswertung aller Beobachtungen ergibt sich also ein boolesches Erfüllbarkeitsproblem, das sich algorithmisch effizient durch einen SAT-Solver lösen läßt. Beispielhaft sind hierfür der Davis-Putnam-Algorithmus [DP60], dessen Erweiterung zum Davis-Putnam-Logemann-Loveland-Algorithmus [DLL62] sowie Implementierungen erweiterter Versionen dieses Algorithmus in [MMZ+01, ES04b].

2.4.6 Zusammenfassung der Diagnoseansätze

In diesem Abschnitt wurden die gängigsten Diagnoseansätze in der Automobilbranche vorgestellt. Tabelle 2.3 zeigt zusammenfassend wie die einzelnen Aufgaben der Diagnose (vgl. Abschnitte 2.2.2 sowie 2.3.1) in den vorgestellten Ansätzen umgesetzt werden. Eine Übersicht über andere Diagnoseansätze von verteilten Systemen, die aber nicht in dieser Dissertation verwendet werden, findet sich in Abschnitt 2.5.

Diagnose-ansatz	Beobachtungen wo plazieren?	Beobachtungen erfassen?	Symptom-generierung	Inferenz
Experten-wissen (Abs. 2.4.1)	Expertenwissen definiert Beob-achtungen	jeweiliges Verhalten	Vergleich mit hin-terlegtem Wissen	anhand Experten-wissens
DES (Abs. 2.4.2)	Systemmodell definiert beob-achtbare Ereig-nisse	jeweiligen Systemzu-stand	Auswerten beob-achtbarer Ereig-nisse und Sichern in Form eines Tra-ces	Durchsuchen des Traces nach Ereig-nissen, die in Be-ziehung zu Feh-lern stehen
MBD (Abs. 2.4.3)	Systemmodell definiert beob-achtbares Ver-halten	jeweiliges Verhalten	Auswerten des beobachteten Ver-halten	Vergleich Soll-/Ist-Verhalten
FDI (Abs. 2.4.4)	Systemmodell definiert beob-achtbares Ver-halten	jeweiliges Verhalten	Auswerten des beobachteten Ver-halten	Vergleich Soll-/Ist-Verhalten
Diagnose als Erfüll-barkeits-problem (Abs. 2.4.5)	Systemmodell definiert beob-achtbares Ver-halten	jeweiliges Verhalten	Auswerten des beobachteten Ver-halten	Umwandlung Diagnose in ein Erfüllbarkeitspro-blem und lösen durch SAT-Solver

Tabelle 2.3: Übersicht der Funktionsweise der einzelnen Diagnoseansätze

Gegenwärtig wird für die off-board Diagnose im Tester meist das fallbasierte Schließen eingesetzt, für Software-lastige Steuergeräte meist das regelbasierte Schließen. In Abschnitt 3.7 wird eine ausführlichere Diskussion der Vor- und Nachteile der vorgestellten Ansätze dargelegt sowie die Ansätze auf einen Einsatz in der Automobilbranche untersucht.

Die Nachteile umfassen dabei beispielsweise eine erschwerte Wiederverwendung des erstellten Diagnosewissens durch mangelnde hierarchische Struktur des Diagnosemodells sowie vor allem die mangelnde Fähigkeit der Überwachung von verteilten Software-Systemen. Diese Systeme verfügen über einen enormen, exponentiell anwachsenden Zustandsraum. Da dieser Zustandsraum durch die Diagnose überwacht bzw. durchsucht werden muß, ist ein effizientes Überwachen bzw. Durchsuchen dieses Zustandsraums notwendig.

Aufgrund der dargelegten Nachteile der bisherigen Ansätze ist die Konzeption eines neuen Ansatzes notwendig, der in Kapitel 4 eingeführt wird.

2.5 Verwandte Arbeiten

In diesem Kapitel wurden die Grundlagen der Diagnose automobiler Systeme vorgestellt. Eine Aufgabe der automobilen Diagnose ist das Erkennen von Fehlern dieser *reaktiven Systeme*. Für den Fehlerbegriff existieren mehrere Definitionen, die sich von

der in dieser Arbeit verwendeten *Fehlerterminologie* der Domäne *Zuverlässigkeit* unterscheiden. Weiterhin wurden die wichtigsten *Diagnoseansätze* der Domäne Automobil vorgestellt.

Für diese Themen existieren unterschiedliche Arbeiten, die sich von den in dieser Arbeit verwendeten Definitionen und Ansätzen zum Teil deutlich unterscheiden. In diesem Abschnitt wird zur Abrundung des Kapitels kurz auf verwandte Arbeiten hingewiesen.

Reaktive Systeme

Automobile Systeme gehören zu der Klasse der reaktiven Systeme, für die es verschiedene Definitionen gibt, wie schon in Abschnitt 2.1.1 gezeigt wurde. Geprägt wurde der Begriff der reaktiven Systeme in Arbeiten von Harel und Pnueli [HP85, Pnu86]. Die Definition wurde aufgegriffen und erweitert, vor allem durch Halbwachs und Berry [BCG87, Ber89, Hal93] im Rahmen der Entwicklung von ESTEREL, einer Programmiersprache für reaktive, synchron agierende Systeme. Eine aktualisierte Fassung der Beschreibung von reaktiven Systemen mitsamt ihren Anforderungen und einer Literaturübersicht bietet [Hal98]. Eine Übersicht über ESTEREL liefert [BCG87] oder [BS91].

Zuverlässigkeit

Für die Zuverlässigkeit von Systemen existieren differenzierende Definitionen.

In der Arbeit wird die Definition von Avižienis und Laprie [ALRL04] aus dem Jahr 2004 verwendet. Diese Arbeit ist eine überarbeitete Version gemeinsamer Arbeiten von Avižienis und Laprie, beginnend mit [Avi67], [Lap85] und [Lap92]. Der Anhang von [Lap92] enthält Übersetzungen der Terminologie in verschiedene Sprachen und trug so einen wichtigen Beitrag zur deutschen Übersetzung von *Reliability* in *Zuverlässigkeit* zu; dieser unterscheidet sich von der Übersetzung *Verläßlichkeit*, die die deutsche Gesellschaft für Informatik (GI) verwendet. Weitere wichtige Arbeiten in diesem Bereich, die sich jedoch in der Terminologie nur unwesentlich unterscheiden, stammen in zeitlicher Reihenfolge von [MIO87, LA90, Lev95, Pha03].

Die im deutschen Sprachraum weitgehend verwendeten Definitionen der Zuverlässigkeit stammen aus den Industrienormen DIN40042 [DIN70], DIN40041 [DIN90] sowie DIN 9000 [ISO05a].

Die DIN40042 von 1970, die im Jahre 1986 zurückgezogen wurde, versteht unter Verfügbarkeit „die Wahrscheinlichkeit, ein System zu einem gegebenen Zeitpunkt in einem funktionsfähigen Zustand anzutreffen“ [DIN70].

Die DIN 40041 in der Version von 1990 versteht unter Zuverlässigkeit die „Beschaffenheit einer Einheit bezüglich ihrer Eignung, während oder nach vorgegebenen Zeitspannen bei vorgegebenen Anwendungsbedingungen die Zuverlässigkeitsanforderungen zu erfüllen“. Die Zuverlässigkeitsanforderung ist die „Gesamtheit der betrachteten Einzelforderungen an die Beschaffenheit einer Einheit, die das Verhalten der Einheit während oder nach vorgegebener Zeitspannen bei vorgegebenen Anwendungsbedingungen betreffen, und zwar in der betrachtenden Konkretisierungsstufe

der Einzelforderungen“ [DIN90]. Zusammengefaßt versteht die DIN 40041 unter Zuverlässigkeit, daß das System während definierten Zeitspannen unter gegebenen Rahmenbedingungen das erwartete Verhalten bezüglich der an das System gestellten Anforderungen zeigt.

Die DIN 9000 wiederum versteht die Zuverlässigkeit als „zusammenfassender Ausdruck zur Beschreibung der Verfügbarkeit und ihrer Einflußfaktoren Funktionsfähigkeit, Instandhaltbarkeit und Instandhaltungsbereitschaft“ [ISO05a].

Fehlerterminologie

ASIL-Fehlerterminologie

Die ASIL-Terminologie für Fehler, definiert in [ISO09a, Teil I], unterscheidet sich leicht von der in der Arbeit verwendeten Definition von Avižienis et al. In diesem Abschnitt werden die ASIL-Definitionen kurz dargelegt.

Ein *Fault* ist eine abnormale Bedingung, die das Versagen eines Elements oder Gegenstand verursachen kann [ISO09a, Teil I Kap. 1.42]. Die ISO26262 versteht als *Error* eine „discrepancy between a computed, observed or measured value or condition and the true, specified, or theoretically correct value or condition.“ Ein Fehler könne entstehen als Ergebnis einer unvorhergesehenen Betriebsbedingung oder aufgrund einer Fehlerursache im betrachteten System, Subsystem oder Komponente. Dies geschehe dadurch, daß eine Fehlerursache sich als Fehlerzustand im betrachteten Element offenbare und nach Ablauf einer Latenzzeit ein Versagen verursachen könne [ISO09a, Teil I, Kap. 1.36]. Der Unterschied zu Avižienis et al. liegt in der Erweiterung des Fehlerzustandsbegriffs durch die explizit definierte Latenzzeit. Ein *Failure* ist das Fehlschlagen der Fähigkeit eines Elements oder Gegenstands, eine Funktion so durchzuführen wie verlangt. Die ISO 26262 unterscheidet dabei zusätzlich zu Avižienis et. al zwischen dem Versagen, eine Funktion durchzuführen wie verlangt sowie dem Versagen, eine Funktion so durchzuführen wie spezifiziert. Letzteres kann beispielsweise durch eine inkorrekte Spezifikation geschehen [ISO09a, Teil I, Kap. 1.39].

Die kausale Kette zwischen den einzelnen Fehlerbegriffen der ISO 26262 findet sich in [ISO09a, Teil X, Kap. 5].

Musas Fehlerterminologie für Software

Musa [MIO87] definiert eine Fehlerterminologie für Software. In [MIO87, Kap. 1.3.1] wird ein *Software Failure* als Abweichung einer externen Ausgabe einer Programmoperation von seiner Anforderung verstanden. Ein *Fault* ist ein Defekt in einem Programm, das -falls unter bestimmten Bedingungen durchgeführt- ein Versagen verursacht. Eine Fehlerursache sei zudem mehr Eigenschaft eines Programms als Eigenschaft seines Verhaltens. Ein Fehler entsteht dadurch, daß der Entwickler des Programms einen *Error* begehe. Hier unterscheidet sich die von Musa definierte Terminologie von der in dieser Arbeit verwendeten.

Weitere Klassifizierung der Fehlerbegriffe

In [Bre01, Kap. 2.1] stellt Breitling eine zusätzliche Klassifikation der Begriffe *Fehlerursache* und *Versagen* vor, die mehrere zusätzliche Quellen zusammenfaßt. *Fehlerursachen* werden dabei nach den Klassifikatoren **Lokalisierung** (intern oder extern), **Verursacher** (Mensch oder Umwelt), **Zeitpunkt** (zur Laufzeit oder während der Entwicklung), **Schwere** (unkritisch, kritisch, katastrophal) und **Dauer** (permanent oder temporär) unterschieden. Versagen werden je nach **Art** (wertbezogen im Sinne eines falschen Ergebnis oder zeitbezogen, bspw. zu spät), **Auswirkung** (unkritisch, kritisch oder katastrophal) oder **Beobachtung** (konsistente oder inkonsistente Wahrnehmung durch Benutzer) differenziert.

Diagnoseverfahren

Kapitel 2.4 bot eine Übersicht über die geläufigsten Diagnoseverfahren in der Automobilindustrie.

Die Arbeiten von [Rei87] und [Kle86] sowie [KW87] (aktualisiert 2008), stellen grundlegende Werke für die modellbasierte Diagnose dar. Diese Werke sind auch in der ersten grundlegenden Übersicht der modellbasierten Diagnose von Hamscher [HCK92] zu finden. Eine kurze zusammenfassende Übersicht der Forschung des modellbasierten Ansatzes findet sich zudem in [KK03]. Eine Übersicht des Standes der modellbasierten Diagnose von System mit kontinuierlichen Systemen in den 1990er Jahren bietet [Ise94], eine aktualisierte Fassung stellt [Ise04] dar.

Weitere Diagnoseansätze sind beispielsweise die Anpassung bekannter stochastischer Ansätze für die Automobildiagnose, wie beispielsweise die Verwendung neuronaler Netze in [MKL+09]. Weitere Anpassungen stochastischer Verfahren werden in Abschnitt 4.8 auf ihre Tauglichkeit für die automobile Domäne diskutiert.

2.6 Zusammenfassung

Automobilsteuergeräte sind verteilte, reaktive Systeme mit speziellen Anforderungen wie beispielsweise hohe Verfügbarkeit und Sicherheit, die zudem aufgrund ihrer großen Stückzahl einem hohen Kostendruck unterliegen.

Da ein Fehlverhalten von Steuergeräten kritische Folgen haben kann, wird die Diagnose benötigt, um Fehler zu erkennen und den Fehler einer Fehlerursache zuzuweisen. Zusätzlich leistet die Diagnose durch das Erkennen und Vermeiden von Fehlern sowie durch Speichern von Hinweisen für eine schnelle Reparatur einen entscheidenden Beitrag zur Erhöhung der Verfügbarkeit des Systems.

Weiterhin wurde eine Fehlerklassifizierung vorgestellt, die es ermöglicht, alle potentiellen Fehler der automobilen Domäne mitsamt ihrer Auftretenssystematik zu erfassen. Hintergrund der Klassifizierung nach Auftretenssystematik ist, daß alle systematischen Fehler durch Qualitätsmethodiken wie beispielsweise eine FMEA erfaßt werden können.

Die automobile Diagnose unterliegt speziellen Anforderungen und Eigenschaften.

Aufgrund dieser Anforderungen und der zunehmenden Komplexität des Autos kommt der Diagnose eine Schlüsselkompetenz in den nächsten Jahren zu. Die Aufgaben der Diagnose werden also erweitert durch die Aufgabe, die Komplexität des Fahrzeugs beherrschbar zu machen.

Um diese Eigenschaften zu erfüllen, ist ein leistungsfähiger Diagnoseansatz notwendig. Die bekanntesten Ansätze wurden vorgestellt. Im späteren Verlauf wird basierend auf der Diskussion der Ansätze in Abschnitt 3.7 ein modellbasierter Diagnoseansatz zur Modellierung des Systemverhaltens mit einem SAT-Solver für die Inferenz gewählt, da dieser die Anforderungen am effizientesten erfüllt.

Kapitel 3

Potentiale der automobilen Diagnose

Eine umfassende Analyse der gegenwärtigen automobilen Diagnose ergab mehrere Optimierungspotentiale hinsichtlich Qualität und Kosten. Der Grundgedanke der Dissertation ist, daß schon eine Reduzierung der Kosten der Diagnose - bei gleichbleibendem monetärem Einsatz- Handlungsspielraum für Investitionen in die Qualität der Diagnose bietet.

Ein gleichzeitiges Ansetzen an beiden Punkten jedoch ermöglicht durch Erhöhung der Qualität bei gleichzeitiger Senkung der Kosten eine deutliche Steigerung der Effizienz der Diagnose. Da zusätzliche Potentiale im Lebenszyklus der Diagnose gehoben werden, werden langfristige Effizienzpotentiale erschlossen.

Abschnitt 3.1 bietet eine kurze Übersicht der Potentiale sowie ihre Auswirkungen auf Qualität und Kosten. Die Potentiale werden in den folgenden Abschnitten im Detail vorgestellt. Anschließend wird aufgezeigt, wie diese Potentiale gehoben werden können. Zusätzlich wird kurz der Nutzen mit den Auswirkungen auf Kosten und/oder Qualität skizziert.

Ziel dieses Kapitels ist, den Leser mit den größten Potentialen der automobilen Diagnose vertraut zu machen, die die Ansatzpunkte für die in der Dissertation vorgestellte Methodik darstellen.

Übersicht

3.1 Übersicht Diagnosepotentiale

Tabelle 3.1 bietet eine Übersicht der Top-Potentiale der gegenwärtigen Diagnose. Zusätzlich werden die Auswirkungen auf die Faktoren Kosten und Qualität gezeigt sowie eine Verknüpfung zum entsprechenden Abschnitt in diesem Kapitel. Dabei bedeutet der ↓-Pfeil in der Spalte Qualität, daß der dargestellte Punkt der Qualität der Diagnose abträglich ist. Der ↑-Pfeil bei Kosten bedeutet eine Erhöhung der Kosten der Diagnose durch das aufgelistete Potential.

Wie in der Einleitung schon erwähnt, wird unter dem Begriff der Qualität in der Dissertation die Definition der IS0 9000:2005 verstanden, die Qualität als „Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt“ [ISO05a] definiert. Unter Qualität der Diagnose wird in dieser Arbeit verstanden, wie die Diagnose die an sie gestellten Anforderungen erfüllt (vgl. Tabelle 2.2).

Potential	Auswirkungen	Kosten	Qualität
Verschiedene Diagnosedatenquellen (Abs. 3.2)	redundante Daten, erschwerte Wartbarkeit		↓
Einbindung Zulieferer (Abs. 3.3)	Informationsfluß stark verbesserbar		↓
Diagnosespezifikation (Abs. 3.4)	Diagnoseergebnisse oft schwerverständlich	↑	↓
	Diagnose schwer(er) validierbar		↓
Effizienz der Diagnoseinferenz (Abs. 3.5)	Erhöhung der Inferenzgeschwindigkeit	↑	
	Zustandsraumexplosion bei Software-Systemen erschwert Inferenz		↓
Diagnose verteilter Funktionen (Abs. 3.6)	Komplexität verteilter Funktionen schwer beherrschbar	↑	↓
Kein gesamthaft geeigneter Ansatz für die Diagnose aller Komponenten (Abs. 3.7)	bisherige Ansätze sind optimierbar und decken nicht alle möglichen Fehler ab	↑	↓
Manuelle Erstellung und eingeschränkte Wiederverwendung Diagnose (Abs. 3.8)	wiederholte manuelle Erstellung ist zeitaufwendig, fehlerträchtig und teuer	↑	↓

Tabelle 3.1: Übersicht Top-Potentiale der Diagnose und ihre Auswirkungen auf Kosten und Qualität. Quelle: eigene Darstellung.

3.2 Verschiedene Datenquellen für die Diagnose

3.2.1 Verschiedene Diagnosedatenquellen in den Entwicklungsphasen

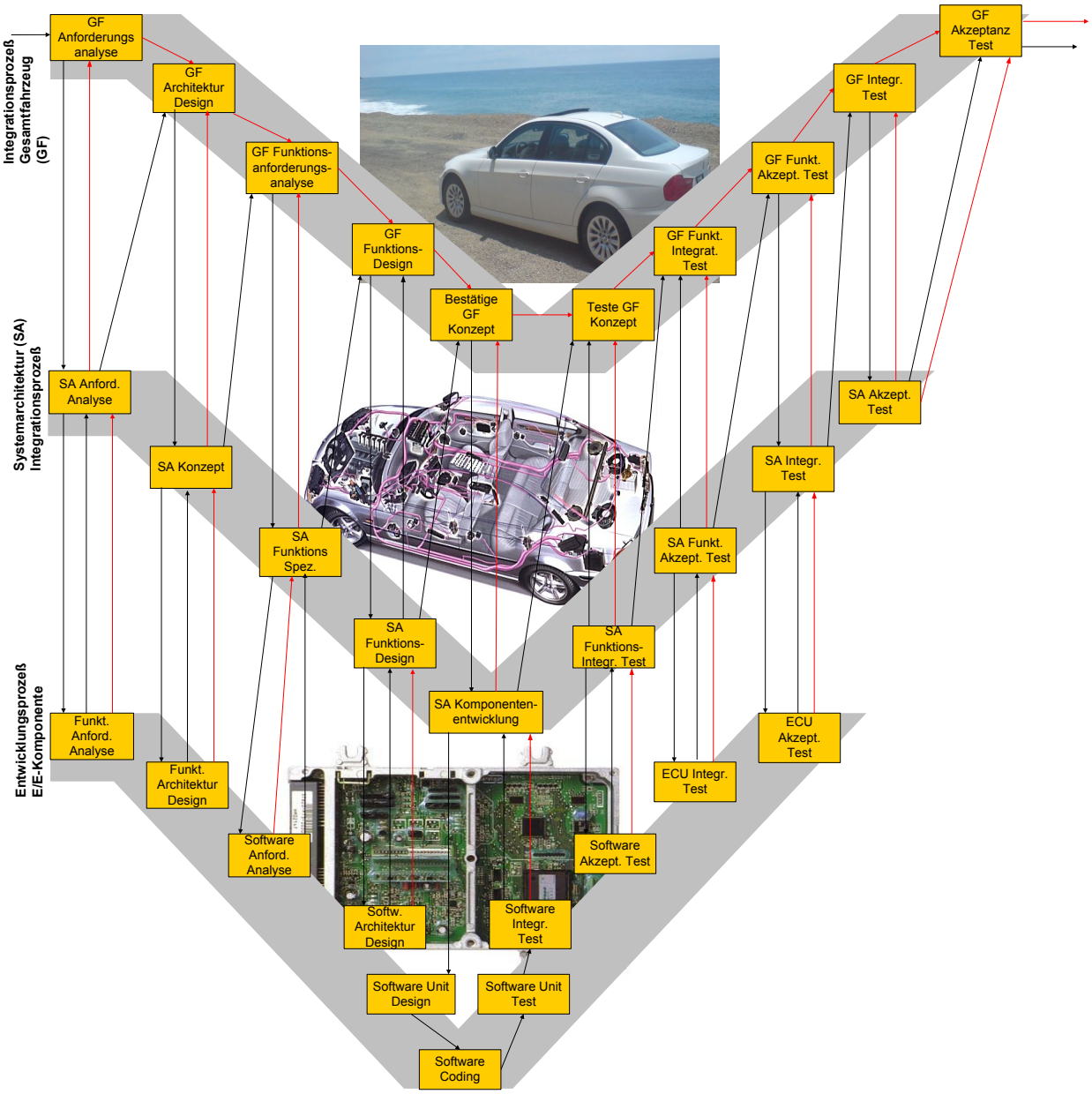


Abbildung 3.1: Dokumenten- und Daten-Workflow im Gesamtentwicklungsprozeß. Quelle: basierend auf Abbildung A.1, [BMW06] und [BMW07]

Abbildung 3.1 zeigt den Datenaustausch zwischen den einzelnen Phasen der vorhandenen Prozeßebenen des Entwicklungsprozeß aus Abbildung A.1. Dabei zeigen rot markierte Pfeile den Austausch von diagnoserelevanten Daten an.

Bei dem in diesem Abschnitt beschriebenen Prozeß handelt es sich um den Standard-BMW-Entwicklungsprozeß (vgl. [BMW06], [BMW07] sowie [SZ10, Kap. 1.4.3]), der

aber stellvertretend für die gesamte Automobilbranche gesehen werden kann. Die Ebenen werden dabei bezeichnet als *Komponentenentwicklungsprozeß*, *Systemarchitekturintegrationsprozeß* sowie *Gesamtfahrzeugintegrationsprozeß*. Diese Prozesse basieren dabei auf dem V-Modell [Koo92] und koordinieren sich gegenseitig über Datenaustausch in verschiedenen Formaten.

Der *Gesamtfahrzeugintegrationsprozeß* findet dabei meist auf höherer Management-Ebene des OEM statt und sorgt dafür, daß zu einem definierten Zeitraum ein stimmiges Fahrzeugprodukt unter Einhaltung eines gegebenen Budgets mit bestmöglicher Qualität geliefert wird. Der Gesamtfahrzeugintegrationsprozeß stimmt sich diesbezüglich mit den unteren beiden Ebenen ab, besitzt jedoch Vorrang gegenüber diesen Ebenen. Dies zeigt sich darin, daß die beiden anderen Ebenen sich ihre Konzepte von der Gesamtfahrzeugebene genehmigen lassen müssen.

Aufgrund der Einführung und Zunahme komponentenübergreifender, verteilter Funktionen ist dem *Systemarchitekturintegrationsprozeß* in den letzten Jahren enorme Bedeutung zugekommen. Hier wird dafür Sorge getragen, daß durch eine geeignete Planung der Hardware-Infrastruktur der Systemarchitektur (Bussysteme, Hardware-Verbindungen, ...) die Teilfunktionen der verteilten Funktionen miteinander echtzeitnah kommunizieren können. Zudem wird schon bei der Planung der Systemarchitektur der Einsatz von weiterentwickelten Komponenten im Rahmen der später erfolgenden Modellüberarbeitung berücksichtigt.

Der *Gesamtfahrzeugintegrationsprozeß* sorgt dafür, daß zum Serienanlauf ein stimmiges Fahrzeugprodukt mit bestmöglicher Qualität geliefert wird. Der Systemarchitekturintegrationsprozeß sorgt zusätzlich für die Integration der einzelnen Komponenten, die im *Entwicklungsprozeß E/E-Komponente* entwickelt werden.

Dieser Entwicklungsprozeß wird oft auch von Zulieferern übernommen, was die spätere Integration der Komponente in das Bordnetz erschweren kann. Dies liegt darin begründet, daß Zulieferer zum einen nur eine lokale Sicht auf ihre Komponente oder Funktionsumfänge haben sowie zum anderen am berechtigten Interesse des Zulieferers, sein wettbewerbsrelevantes geistiges Eigentum zu schützen, weshalb er oft sein Wissen nicht umfassend mit dem OEM teilen möchte (vgl. Abschnitt 3.3).

Dies ist besonders für die Integrationsphasen im V-Modell relevant, da erst dann viele Fehler im Zusammenspiel der verteilten Funktionen aufgedeckt werden können. Die Abstellung dieser Fehler bedarf der intensiven Kommunikation zwischen Zulieferer und OEM. Hinzu kommt, daß die komponentenübergreifende Spezifikation der verteilten Funktionen enorm verbesserbar ist (vgl. Abschnitt 3.6).

Tabelle 3.2 zeigt eine Zusammenfassung der zwischen den einzelnen Ebenen aus Abbildung A.1 ausgetauschten Diagnosedaten.

Durch die Verwendung verschiedener Datenformate für ähnliche Informationen können folgende Probleme entstehen:

- Datenkonsistenz und Nachverfolgbarkeit der Daten
- eingeschränkte Wiederverwendbarkeit der Daten
- erhöhter Wartungsaufwand

Dokument	Typ	Erstellt von
generische Diagnoseanforderungen für alle Komponenten	Anforderung	OEM
komponentenspezifische Diagnoseanforderungen	Anforderung	OEM/Zulieferer
generische Diagnosespezifikation aller Komponenten	Spezifikation	OEM
komponentenspezifische Diagnosespezifikation	Spezifikation	OEM/Zulieferer
Dokumentation der Fehlerspeichereinträge (DTC)	Tabelle	OEM/Zulieferer
Dokumentation der off-board Diagnosefunktionen	Tabelle	OEM/Zulieferer
Programm-Code Diagnose für das Steuergerät	C/C++/...-Datei, CASE-Tool-Modelle	OEM/Zulieferer
Bus-Telegramme für diagnoserelevanten Datenaustausch verteilter Funktionen	Nachrichtenkatalog	OEM, abgestimmt mit den Zulieferern
Programm-Code Diagnose für Werkstatt-Tester	XML-Datei	OEM

Tabelle 3.2: Übersicht Datenformate der Diagnose im Entwicklungsprozeß in zeitlicher Reihenfolge. Quelle: eigene Darstellung.

Datenkonsistenz und Nachverfolgbarkeit. In den Integrationsphasen des Entwicklungsprozesses können viele unbekannte funktionale Abhängigkeiten sowie transitive Fehler entdeckt werden. Für diese gefundenen Fehler werden Maßnahmen entwickelt, um diese on-board im Steuergerät zu beheben oder zu vermeiden. Dies führt zu zeitnahen Änderungen der Steuergeräte-Software und Nachtests, da die Abstellung der Fehler vom Lieferanten und vom OEM bestätigt werden muß. Aufgrund der zeitlichen Verzögerung zwischen beiden Bestätigungen und der fortlaufenden Weiterentwicklung wird jedoch die **Nachverfolgbarkeit** dieser für die Diagnose relevanten Daten erschwert.

Diese Problematik wirkt sich zudem auf die Diagnosefunktionen aus, die in den Integrationsphasen validiert werden (vgl. Abschnitt 2.3.5). Die Komponentendiagnose wird meist von Zulieferern erstellt und in vereinbarten Umfängen von sowohl Zulieferern als auch OEM geprüft, die off-board Diagnose jedoch vorwiegend vom OEM. Zeitliche Latenz und der fortschreitende Entwicklungsprozeß wirken sich somit auch erschwerend auf die **Konsistenz** der on- und off-board Diagnosedaten aus.

Eingeschränkte Wiederverwendbarkeit der Daten. Die Gründe und Auswirkungen der eingeschränkten Wiederverwendbarkeit von Diagnosedaten werden detaillierter in Abschnitt 3.8 dargestellt.

Erhöhter Wartungsaufwand. Das Problem der getrennten Datenhaltung für on- und off-board Diagnose verschärft sich in der Praxis durch die Tatsache, daß die

Diagnose des Fahrzeugs inklusive seinen Teilen durch den Automobilhersteller bis zu 15 Jahre nach Ende der Serienproduktion des Fahrzeugs aufrechterhalten werden muß (vgl. Abschnitt 2.3).

Wie in [Koh06, Edl01, KKP+11] gezeigt, treten zu viele Fehler erst nach der Produktion im Lebenszyklus auf. Die Diagnose muß dann um Maßnahmen für diese Fehler erweitert werden, was ein Zusammenspiel und somit Abstimmung der on- und off-board Diagnose erfordert, wie die Fehlerklassifikation aus Abschnitt 2.3.3 zeigt.

3.2.2 Diagnosefunktionen nicht aus einheitlicher Quelle

In diesem Abschnitt soll nochmals detaillierter auf die Generierung der Diagnosefunktionen für Tester und Steuergerät eingegangen werden.

Tabelle 3.2 zeigt, daß sich die Diagnosefunktionen für die on- und off-board Diagnose sowohl im Ursprungs- als auch Zielformat unterscheiden. Der Grund hierfür liegt darin, daß die Diagnose für die Werkstätten meist vollständig vom OEM erstellt wird, die Diagnose für das Steuergerät jedoch vorwiegend vom Zulieferer.

Weiterhin haben die off- und on-board Diagnose verschiedene technische Detailtiefen. Der Detail-Level der Werkstattdiagnose ist aufgrund der beschränkten Zugriffsmöglichkeiten der Werkstätten deutlich geringer als der der Diagnose im Steuergerät. Dies wirkt sich auf die Wahl des Diagnoseverfahrens aus. In den Werkstätten wird meist der *Programmablaufplan* (Abschnitt 2.4.1) verwendet, während im Steuergerät neben dem häufig eingesetzten *regelbasierten Verfahren* (Abschnitt 2.4.1) auch *modellbasierte Verfahren* (Abschnitte 2.4.3, 2.4.4) zum Einsatz kommen. Die aufgeführten Verfahren unterscheiden sich nicht nur in der Inferenz, sondern auch in der Speicherung des zugrundeliegenden Wissens (siehe auch Abschnitt 3.7) deutlich.

Sowohl Detaillierungsgrad als auch die verschiedene Struktur des erfaßten Diagnosewissens erschweren eine gemeinsame Datenhaltung und wirkt sich besonders auf die Aktualisierung und Wartung der Diagnose in den Integrationsphasen und im Lebenszyklus aus. Der Grund hierfür ist, daß in diesen Phasen das Zusammenspiel der Komponenten getestet wird und somit viele bisher unbekannte transitive Abhängigkeiten und unbekannte Fehler erstmalig in kurzen zeitlichen Abständen auftreten. Aufgrund der erwähnten Differenzen zwischen der on- und off-board Diagnose sowie den kurzen Zeitabständen muß darauf geachtet werden, daß die zu entwickelnde Diagnoseabdeckung dieser neugewonnenen Erkenntnisse zeitlich nicht divergiert.

Lösungsansatz

In diesem Abschnitt wird kurz skizziert, wie die beiden in diesem Abschnitt beschriebenen Potentiale gehoben werden.

Es wurde beschrieben, daß im Entwicklungsprozeß verschiedene Dokumente bearbeitet und ausgetauscht werden. Der gesamte Datenaustausch zwischen den einzelnen Ebenen wurde in Abbildung 3.1 gezeigt. Die Analyse des Workflow ergab, welche Daten für die Diagnose relevant sind. Diese finden sich in Tabelle 3.2. Zudem wurde erwähnt, daß die Diagnose für das Steuergerät und für den Tester gegenwärtig aus verschiedenen Datenquellen erstellt wird.

Diese beiden Probleme wirken sich sowohl auf die Wartbarkeit und Konsistenz der Diagnosedaten als auch auf ihre Wiederverwendbarkeit aus.

Der Beitrag der Arbeit hierzu besteht aus drei Punkten. Es wird ein auf einer erweiterten *Failure mode and effects analysis* (FMEA, [DIN06]) basierendes, *hierarchisches Datenmodell* für die on- und off-board Diagnose eingeführt, das vorgibt, welche Daten für die Diagnose notwendig sind. Aus diesem Datenmodell werden dann die Diagnosefunktionen für das Steuergerät und den Tester generiert. Eine Einbindung des Modells in einen *Prozeßvorschlag für die Diagnose* gibt vor, zu welcher Prozeßphase welche Daten bzw. Dokumente vorliegen müssen. Weiterhin wird durch eine *statistische Analyse der Lebenszyklusdaten* für eine Wartung des Diagnosedatenmodells gesorgt.

Nutzen

Das hierarchische Datenmodell ermöglicht eine Generierung der on- und off-board Diagnosefunktionen, die sich auf verschiedenen Abstraktionsebenen befinden. Weiterhin erlaubt das hierarchische Datenmodell mit seinen Abstraktionsebenen die Wiederverwendung von Diagnosefunktionen und -daten. Dies führt zum einen zu einer Reduktion des Erstellungsaufwand der Diagnose, zum anderen wird durch die Wiederverwendung die Qualität der Diagnose an sich erhöht und es wird vermieden, daß bekannte und erfaßte Probleme erneut ins Feld gelangen. Die für die Wiederverwendung notwendige Aktualität der Diagnosedaten wird durch eine kontinuierliche statistische Analyse der Diagnosedaten und einer daraus abgeleiteten Wartung gewährleistet.

Die Reduktion des Umfangs der Diagnosedaten auf „wichtige“ Daten reduziert den Wartungsaufwand und -kosten der Diagnose und verringert die Gefahr, daß die off-board Diagnose aufgrund ihrer langsameren Entwicklung in großen zeitlichen Verzug zur on-board Diagnose gerät.

3.3 Einbindung Zulieferer

> „What we’ve got here is...failure to communicate. Some men you just can’t reach.“
>
> *(The Captain, Cool Hand Luke)*

Potential

Abbildung A.1 beschreibt den automobilen Entwicklungsprozeß. In diesem Prozeß werden die Fahrzeugkomponenten entweder vom OEM, vom OEM in Zusammenarbeit mit Zulieferern oder mehrheitlich vom OEM nur spezifiziert und von Zulieferern entwickelt. Die von den Zulieferern entwickelten Teilfunktionalitäten werden beginnend mit den Integrationsphasen des Entwicklungsprozesses in das vom OEM verantwortete Bordnetz und Gesamtfahrzeug eingebaut.

Um eine reibungslose Integration sicherstellen zu können, definiert der OEM Anforderungen an die Komponente, wie beispielsweise einzuhaltende Schnittstellen. Dies

gilt natürlich auch für die Diagnose, wie Tabelle 3.2 zeigt. Abbildung 3.1 stellt den definierten Entwicklungsworkflow zwischen Entwickler und Zulieferer dar.

Dennoch sind Zusammenarbeit und insbesondere Wissensaustausch zwischen OEM und Zulieferer verbesserbar. In diesem Abschnitt werden die Gründe hierfür dargelegt sowie ein Lösungsansatz mit Nutzen skizziert.

Die OEM vergeben immer mehr Umfänge des Fahrzeugs zur Entwicklung an Zulieferer. So ergaben Studien, daß der Wertschöpfungsanteil der OEM an ihren Fahrzeugen von 35% im Jahre 2003 auf ungefähr 25% [MI03, S. 44ff] oder 23% [DK04] im Jahr 2015 sinken werde.

Abschnitt 3.4 zeigt, daß die ungenaue Spezifikation des Gesamtsystems und seiner untergeordneten Teilfunktionalitäten durch den OEM oft dazu führt, daß sowohl Zulieferer als auch OEM wichtige Informationen fehlen, insbesondere für die Diagnose.

In Abschnitt 2.1.2 wurden zudem die Auswirkungen des Kostendrucks auf die Automobilbranche beschrieben. Aufgrund dieses Drucks versuchen die OEM einerseits Kosten an ihre Zulieferer weiterzugeben (indem der Zulieferer seinerseits Kostenpotentiale heben soll), andererseits aber durch Outsourcing von Technikkompetenz an die Zulieferer Geld zu sparen [Bec05, Kap. 5.2.1]. Dadurch entsteht das Dilemma der Zulieferer, intern Kosten sparen zu müssen, um den Kostendruck der Hersteller auffangen zu können und zugleich Geld in die Forschung und Entwicklung für OEM-attraktive Innovationen zu investieren. Auf Seiten der OEM ist es wichtig, daß für einen Auftrag mehrere Zulieferer in Frage kommen, damit der Kostendruck des Herstellers aufrechterhalten werden kann.

Dies führt dazu, daß die Zulieferer untereinander in Wettstreit um Aufträge stehen, die sie entweder über Innovationsführerschaft bei gewissen Komponenten (vgl. [MI03, Abb. 4], zitiert in Abschnitt 2.1.2), ausgiebiges Know-how, einen billigen Stückpreis [Bec05, Kap. 5.3.2], oder durch das beste Gesamtpaket bekommen. Für eine ausführlichere Darstellung möglicher Strategien für Zulieferer sei beispielsweise auf [Bec05, Kap. 5.3] verwiesen.

Für Zulieferer aus Hochlohnländern wie Deutschland führt somit der Weg zu Aufträgen meist über stetige technische Innovationen und Innovationsführerschaft. Deshalb ist es für die Zulieferer auch eminent wichtig, ihr wettbewerbsrelevantes Know-how zu schützen.

Dieses Bestreben hat für die Diagnose negative Auswirkungen, da das für den OEM notwendige Wissen zur Erstellung der Diagnose mit dem Ziel des Zulieferers, Know-how zu schützen, in Konflikt steht. Sind die vom Zulieferer erstellten Umfänge Teil eines Gesamtsystems, so wird abhängig von der eingesetzten on-board Diagnosemethodik (vgl. Abschnitt 3.7) ein mehr oder weniger umfangreiches (internes) Modell benötigt. Dies ist ein Grund, wieso von Seitens des Zulieferers Diagnoseverfahren bevorzugt werden, die ihre Komponenten mittels einer sogenannten „Black Box“-Sicht überwachen und diagnostizieren können. Bei der off-board Diagnose wird zumeist in Absprache zwischen OEM und Zulieferer eine möglichst kleine austauschbare Einheit definiert, die die interne Diagnose des Zulieferers identifizieren muß. Dennoch gibt es auch hier Konflikte zwischen dem Bestrebens des OEM, durch ein ausführliches Diagnosemodell diese austauschbare Einheit möglichst klein und folglich kostengünstig werden zu lassen und dem erwähnten Interesse des Zulieferers möglichst wenig an

Wissen preiszugeben.

Lösungsansatz

Der Lösungsansatz baut auf der Tatsache auf, daß sich in den letzten Jahren aufgrund des Kostenhebels der OEM sowie der ASIL-Vorgabe, die Fehlerdaten sicherheitsrelevanter Systeme zu analysieren [ISO09a, Teil VII, Kapitel 6.4.2.1], eine Entwicklung in Richtung Bereitstellung ausführlicherer Diagnosedaten durch den Zulieferer schrittweise vollzogen hat. So legen die Lastenhefte der OEM fest, daß der Zulieferer die Qualität seiner Komponenten durch Anwendung von Qualitätsmethoden sicherstellen und die Daten aus Tabelle 3.2 zur Verfügung stellen muß.

Dennoch kann aus den gelieferten Daten, wie schon Abschnitt 3.2 zeigte, erhöhter Nutzen gewonnen werden. Kernpunkt ist hier das schon erwähnte geschichtete Datenmodell.

So wird in Abschnitt 3.4 dargelegt, wie durch Zusammenspiel mit dem definierten Datenmodell sowohl die Spezifikation der Diagnose auf Gesamtsystemebene für den OEM als auch die Spezifikation der Umfänge eines Lieferanten verbessert werden.

Zusätzlich wird in der Arbeit ein Diagnoseprozeß definiert, der festlegt, in welchen Entwicklungsphasen vom Zulieferer die Diagnosespezifikation mit welchen Daten befüllt werden und welche Daten zusätzlich geliefert werden müssen.

Durch Prozeß und Datenmodell wird somit eine Zuliefersicht mit Schnittstelle zum OEM erstellt (siehe auch [BGK+09]), die es dem Zulieferer ermöglicht, beginnend mit den frühen Entwicklungsphasen, die Diagnose besser zu entwickeln und gegen die Schnittstelle des Datenmodells zu prüfen. Die Sicht berücksichtigt zudem, welche Daten der Zulieferer vom OEM benötigt.

Da der Zulieferer dennoch darauf bedacht sein mag, sein Wissen möglichst umfangreich zu schützen, wird zudem im Rahmen des Ansatzes ein Parser eingeführt, der nur die für den OEM relevanten Diagnosedaten extrahiert. Da der Parser vom Zulieferer ausgeführt werden kann, bleibt so eine Black Box-Sicht des OEM auf die nicht diagnoserelevanten Daten bestehen.

Nutzen

Der Nutzen der Zuliefersicht besteht nicht nur in der Erhöhung des Informationsflusses im Entwicklungsprozeß Diagnose, sondern auch darin, daß der Informationsfluß früher starten kann, da aufgrund der definierten Schnittstelle sowohl OEM und Zulieferer wissen, welche Daten wann benötigt werden. Dies reduziert redundante Daten und vermeidet zudem die Entstehung weiterer Fehler durch nicht rechtzeitigen Austausch von Daten.

Ein wichtiger Beitrag der Zuliefersicht ist zudem, daß der Zulieferer frühzeitig die Diagnose spezifizieren kann und diese dank der OEM-Schnittstelle auch für verteilte Funktionen validieren kann. Somit steigt die Anlieferqualität der Diagnose insgesamt.

3.4 Spezifikation der Diagnose

Potential

Der Gesamtentwicklungsprozeß aus Abschnitt A.1 zeigt, daß das Fahrzeug auf den Ebenen Gesamtfahrzeug und Systemarchitektur vollständig vom OEM spezifiziert wird. Diese Spezifikationen dienen den Zulieferern als Basis für die Entwicklung der beauftragten funktionalen Umfänge und sind zumeist in *textueller Form*. Die Diagnose ist Teil dieser Anforderungen, wird jedoch nur generell definiert (vgl. Tabelle 3.2).

Abschnitt 2.2.2 zeigte, daß aufgrund der Sicherheits- und Kundenrelevanz von Fahrzeugsystemen oder -funktionen die Diagnose in die Funktion oder das System eingreift und somit deren Verhalten ändert. Aufgrund dieser Eigenschaft muß die Spezifikation der Diagnose auf zwei verschiedenen Ebenen gesehen werden.

Zum einen muß in der *Funktionsspezifikation* das Sollverhalten der Funktion genau festgelegt werden und somit welches Verhalten als fehlerhaft zu werten ist. Diese Spezifikation dient als Basis für die Fehlererkennung der Diagnose.

Zum anderen muß das *Sollverhalten im Fehlerfall* definiert werden. Dieses Sollverhalten ergibt sich aus den Maßnahmen, die die Diagnose für erkannte Fehler durchzuführen hat. Das fehlerhafte Verhalten der Funktion ist durch die Funktionsspezifikation bekannt. Erkennt die Diagnose fehlerhaftes Verhalten, so hat sie das Auftreten mitsamt der bestimmten Fehlerursache für eine Fehlerbehebung zu dokumentieren. Zusätzlich kann die Diagnose in der Lage sein, das aufgetretene Versagen abzuschwächen oder das Eintreten des Versagens beginnend mit der Erkennung des Fehlers zu verhindern. Da durch einen Eingriff der Diagnose in das System das erkannte, fehlerhafte Verhalten geändert werden kann, ist die Spezifikation dieses Sollverhaltens im Fehlerfall notwendig, auch um Kundenirritationen zu vermeiden.

In Abschnitt 2.4 wurden die verschiedenen im Automobil eingesetzten Diagnoseansätze vorgestellt sowie deren Speicherung des relevanten Diagnosewissens. Datengrundlage des Diagnosewissens sind die in Tabelle 3.2 gezeigten Datenformate. Meist sind weder die Eingangsdaten für die Diagnose noch das gespeicherte Diagnosewissen formal. In diesem Abschnitt werden die Probleme dargestellt, die die gegenwärtige textuelle Spezifikation der Diagnose nach sich ziehen kann.

Diagnose oft nicht Teil der Funktionsspezifikation. Die Diagnose ist oft nicht in ausreichendem Maße Teil der Funktionsspezifikation. Wie schon erwähnt, werden auf den oberen Abstraktionsebenen nur allgemeine Diagnoseanforderungen erstellt. Dies liegt daran, daß viele Fehler von Funktionen bei neuen Komponenten zum Zeitpunkt der Entwicklung unbekannt sind, von unbekannten Einflußgrößen verursacht oder aufgrund eines optimierbaren Datenmanagements (vgl. Abschnitt 3.2) nicht richtig aus Vorgängerkomponenten erfaßt wurden. Zudem kann gerade bei beginnender Entwicklung das Sollverhalten des Systems, auch im Fehlerfall, sowohl durch den OEM als auch Zulieferer nur grob und nicht im Detail definiert werden.

Dies ist besonders für Diagnoseansätze, die ein genaues Verhaltensmodell für die Diagnose benötigen, wie bspw. die Discrete Event System (Abschnitt 2.4.2) oder die regelbasierte Diagnose (Abschnitt 2.4.1) nachteilig. Diagnoseansätze, die in der Lage

sind, Komponenten mit Black Box-Sicht anhand von abweichenden Verhalten zu diagnostizieren, wie beispielsweise die modellbasierte Diagnose (Abschnitt 2.4.3) oder der FDI-Ansatz (Abschnitt 2.4.4), sind hier vorteilhaft.

Diagnose oft nicht exakt spezifiziert. Viel häufiger jedoch wird die Diagnose nicht exakt spezifiziert. Die Automobilhersteller spezifizieren die Funktionalitäten des Fahrzeugs und deren Diagnose auf ganz unterschiedlichen Abstraktionsebenen in einem verbesserbaren Workflow (vgl. Abschnitt 3.2). Dies zeigt sich darin, daß für die Komponentenumfänge nur recht grobe Spezifikationen existieren, die von den Zulieferern erweitert werden. Besteht eine Funktionalität aus verschiedenen Teilfunktionen, die von verschiedenen Zulieferern entwickelt werden, ergibt sich so ein erhöhter Koordinationsaufwand.

Weiterhin führt die Spezifikation der Diagnose auf höheren Abstraktionsebenen oft dazu, daß die Spezifikation nicht mit ausreichend exakten Informationen versehen ist. Dies betrifft vor allem verteilte Funktionen, die mit anderen Funktionen des Fahrzeugs interagieren. Bei Fehlern dieser Funktionen ist es sowohl für den Zulieferer als auch den OEM schwierig, die Auswirkungen der Funktionen oder Fehler auf das Gesamtfahrzeug im Entwicklungszeitraum genau zu erfassen. Es existieren zwar zeitlich aufwendige Qualitätsmethoden wie bspw. die *System-FMEA* [VDA96], die die Auswirkungen eines Fehlers auf das Gesamtsystem untersuchen, dennoch können auch hiermit aufgrund der fehlenden Schnittstelle bzw. Zuliefersicht nicht alle Fehler genau erfaßt werden.

Zudem existieren auch viele transitive Abhängigkeiten in den Fahrzeugfunktionen, die erst in den Integrationsphasen entdeckt werden können. Auch diese sind aufgrund der verbesserbaren Spezifikation der Funktionen und Diagnose schwerer abzuprüfen und festzustellen.

Schließlich sind die bereitgestellten und ausgetauschten Informationen oft auch für Experten schwerverständlich und schwer nachvollziehbar. Das betrifft bei Reparaturen vor allem die Werkstätten, nicht nur aufgrund ihres geringeren technischen Wissens, sondern vor allem wegen des ohnehin beschränkten Zugriffs auf die Steuergeräte. Eine exakte Spezifikation des Fahrzeugs, am besten auf mehreren Detailebenen, kann so das technische Verständnis der Werkstätten erhöhen.

Da das zu erkennende Fehlverhalten meist nur informell beschrieben wird, stellt sich auch die Frage nach der genauen Umsetzung der spezifizierten Diagnose.

Hier sind besonders die *Setzbedingungen eines Fehlerspeichereintrags* zu nennen (vgl. Abschnitt 4.5.3), die die Bedingungen für das Speichern eines Fehlerspeichereintrags festlegen. Da Fehler nur informell beschrieben werden, kann es abhängig vom Programmierstil zu härteren oder weicheren Setzbedingungen führen. Dies führt beispielsweise dazu, daß bei Verletzung einer zu weichen Setzbedingung ein Fehlerspeichereintrag gesichert wird, obwohl kein Fehler vorliegt. Dies wird als *unberechtigter Fehlerspeichereintrag* bezeichnet. Unberechtigte Fehlerspeichereinträge kommen besonders oft bei verteilten sicherheitsrelevanten Funktionen vor, falls die Fehlersetzbedingungen der Teilfunktionen nicht abgestimmt werden. Das Sicherheitskonzept von sicherheitsrelevanten Funktionen sieht oft vor, daß sich Teilfunktionen auf verschiedenen Steuergeräten gegenseitig für eine erhöhte Ablaufsicherheit überwachen und als fehlerhaft gewertetes Verhalten interagierender Komponenten mit Fehlerspei-

chereintrag festhalten und das Setzen ihren Kommunikationspartnern per Nachricht mitteilen.

Keine komponentenübergreifende Diagnosespezifikation. Die Diagnosespezifikation besitzt, vor allem bei von Lieferanten erstellten Komponenten, eine komponentenzentrierte Sichtweise, was die Diagnose und Spezifikation verteilter Systeme erschwert. Auf diesen Aspekt wird in Abschnitt 3.6 genauer eingegangen. Dies betrifft auch stark das Sollverhalten im Fehlerfall, da ein Eingreifen der Diagnose in eine Komponente Auswirkungen auf andere Komponenten haben kann.

Einbindung Zulieferer. Es wurde in diesem Abschnitt schon erwähnt, daß die Einbindung des Zulieferers bei der Spezifikation der Diagnose verbesserbar ist. Dies wurde in Abschnitt 3.3 im Detail dargelegt.

Lösungsansatz

In diesem Abschnitt wird kurz skizziert, wie die Spezifikation der Diagnose verbessert wird.

In der Arbeit wird ein Vorschlag für einen Entwicklungsprozeß für verteilte Systeme vorgestellt (siehe auch Prozeß in Abschnitt 5.1 sowie die einzelnen Schritte in Abschnitt A.3). Dieser Prozeß macht die Diagnose, ihre Anforderungen sowie Spezifikation zu verpflichtenden Bestandteilen des Entwicklungsprozesses und stuft die Diagnose somit gleichwertig zu den funktionalen Umfängen ein.

Es wurde erwähnt, daß aufgrund des zum Zeitpunkt der Entwicklungsphasen unvollständigen Wissens um mögliche Fehler die Spezifikation der Diagnose sich meist auf generelle Anforderungen beschränkt. Dennoch bietet sich durch die Einführung von Funktionsarchitekturen an, die Spezifikation der in der Architektur enthaltenen Funktionsumfänge mit der der Diagnose zu koppeln.

In der Arbeit wird ein geschichtetes Datenmodell für die Diagnose vorgestellt und die diagnoserelevanten Daten formal mit aussagenlogischen Formeln sowie bei verteilten Funktionen in Form von Message Sequence Charts (MSC) [MSC04] spezifiziert. Dabei wird auch das sich durch Eingriffe der Diagnose ändernde Verhalten berücksichtigt. Die Inhalte der MSC lassen sich hierbei wiederum auf aussagenlogische Formeln zurückführen. Die Schichten sind dabei mit Kunde, Werkstatt und Entwicklung den Handlungsrollen der Diagnose nachempfunden. Dieses Modell wird an eine Funktionsarchitektur gekoppelt.

Da die oberen Ebenen der Funktionsarchitekturen auch in den frühen Phasen der Entwicklung schon spezifiziert werden, ist eine frühe Spezifikation der Diagnose möglich. Zudem ergibt sich durch die Definition der verschiedenen Abstraktionsebenen gleichzeitig eine Reduktion der Komplexität sowie Erhöhung der Verständlichkeit der Diagnose, insbesondere für die Werkstätten.

Dennoch ist die Spezifikation der Diagnose ein so wichtiger Aspekt, daß sie zu einem wichtigen Bewertungskriterium der Diagnoseansätze in Abschnitt 3.7 wird.

Nutzen

Die enorme Wichtigkeit der formalen Spezifikation der Diagnose zeigt sich beispielsweise darin, daß sie sowohl eine Verifikation und Validierung der Diagnose als auch die automatisierte Generierung von Diagnosefunktionen erleichtert.

Die Validierung der Diagnoseanforderungen ermöglicht zum einen eine Erhöhung der Qualität der Diagnose, zum anderen aber auch, daß keine ungeprüften Diagnosefunktionen im Feld eingesetzt werden. Weiterhin wird dadurch der Aufwand der bisher notwendigen manuellen Prüfung der Diagnose reduziert.

Das Schichtenmodell für die Diagnose erlaubt die Anpassung des Diagnoseumfangs an das technische Know-how der handelnden Rollen, bspw. der Werkstatt, und erlaubt zudem die Einbindung des Kunden, die besonders bei unbekannten Fehlern wichtig ist.

Abschließend sei ein weiterer Nutzen einer umfangreichen Spezifikation der Diagnose genannt. Durch eine formale Spezifikation der Diagnose ist es prinzipiell möglich, alle systematischen, reproduzierbaren Fehler eines Systems zu erfassen. Bei einer Erfassung aller reproduzierbaren Fehler bleiben somit nur die nicht- oder schwer reproduzierbaren, sporadisch auftretenden, Fehler übrig, die aber somit leichter durch Ausschlußverfahren bestimmt werden können.

Das Erreichen einer möglichst hohen Abdeckung von systematischen Fehlern wird in dem in der Dissertation vorgestellten Ansatz angestrebt, indem das vorhandene, formal erfaßte Diagnosewissen erweitert wird durch zusätzliches, im Lauf des Lebenszyklus gewonnenes, Diagnosewissen. Zudem erlaubt die Strukturierung des Diagnosewissens die Wiederverwendung vorhandenen Wissens bei ähnlichen Komponenten oder Weiterentwicklungen.

3.5 Effiziente Diagnoseinferenz

Potential

Die Diagnose versucht anhand von Beobachtungen festzustellen, ob sich das überwachte System fehlerfrei verhält. Stellt die Diagnose Unterschiede zwischen dem beobachteten und spezifizierten Verhalten fest, muß sie die Ursache für die Abweichung feststellen. Dieser Schluß basiert auf den generierten Symptomen und wird als *Inferenz* bezeichnet. Die Inferenz ist essentieller Bestandteil der Diagnose und findet bei jedem Diagnosevorgang statt.

Eine schnelle und effiziente Inferenz reduziert also die Dauer jedes Diagnosevorgangs. Eine Reduktion der Diagnosedauer in den Werkstätten bedeutet somit eine Kostenreduktion von Reparaturen sowie eine Steigerung der Kundenzufriedenheit, da sich der Werkstattaufenthalt verkürzt. Da aber viele Diagnosen oft ausschließlich im Zusammenspiel mit dem Werkstattpersonal erfolgen, ist die Geschwindigkeit der Inferenz vom Menschen abhängig. Eine erhöhte Inferenz im Steuergerät hingegen steigert die Fähigkeit des Diagnoseansatzes, anbahnende Fehler schneller zu erkennen.

Trotz der Wichtigkeit der Inferenz ist es um so verwunderlicher, daß die Inferenz

der meisten Diagnoseansätze über eine eher schlechte Laufzeitperformanz verfügt. So setzt beispielsweise der im Tester häufig eingesetzte Prüfplan eine Breitensuche (vgl. [Knu97]) ein, die im schlechtesten Fall ein Laufzeitverhalten von $\mathcal{O}(|V| + |E|)$ verfügt mit $|V|$ Anzahl der zu durchsuchenden Knoten und $|E|$ Anzahl der Kanten. Im Prüfplan entspricht V dabei der Anzahl der Fälle, Testmodule oder Testschritte und E der Anzahl der Wahlmöglichkeiten.

Das Problem der Laufzeitperformanz der Inferenz vergrößert sich bei Systemen mit großen Zustandsräumen, wie den verteilten Software-Funktionen des Automobils, deren Umfang zudem stetig wächst. Bei diesen Software-Funktionen tritt das als *state explosion* bekannte Problem des exponentiellen Wachstums der möglichen Zustände auf (vgl. [VW94]), die durch die Diagnose überwacht werden müssen.

Bei N Komponenten und Z_{K_i} der Anzahl der Zustände einer Komponente K_i ergibt sich für die Größe des zu überwachenden Zustandsraum $\prod_{i=1}^{N} Z_{K_i}$. Sind beispielsweise in einer verteilten Funktionalität 9 Komponenten involviert, so ergeben sich selbst für eine geringe Anzahl von 10 möglichen Zuständen pro Komponente ein gesamthafter Zustandsraum von 10^9 Zuständen.

Lösungsansatz

Aufgrund der Wichtigkeit der Inferenz für die Effizienzsteigerung sowohl der on- als auch off-board Diagnose wird sie als Kriterium bei der Bewertung der Diagnoseansätze in Abschnitt 3.7 genommen.

Die in der Dissertation verwendete Diagnosemethodik setzt einen SAT-Solver (vgl. [GPFW96]) für die Inferenz ein. Für die Lösung eines Erfüllbarkeitsproblem durch den SAT-Solver gilt $\mathcal{NP}$-Komplexität (vgl. [Coo71]), der SAT-Solver benötigt im Worst Case 2^n-Suchschritte [GPFW96] mit n Anzahl der Atome der Formel.

Durch die Speicherung der Variablen in der konjunktiven Normalform sowie des Einsatzes von Heuristiken beim Lösen (vgl. [GPFW96, Zha03] sowie Abschnitt 4.7) muß der Solver jedoch nicht alle möglichen Pfade des Zustandsraum für eine Lösung durchsuchen. Deshalb ist der SAT-Solver deutlich besser in der Lage, große Zustandsräume, wie sie bei Software-intensiven Systemen auftreten, zu durchsuchen als andere Verfahren [GPFW96].

Weiterhin wird der eingeschränkte Suchraum durch den in der Arbeit vorgestellten rollenbasierten Ansatz zusätzlich eingegrenzt.

Nutzen

Eine effiziente Inferenz ermöglicht eine Verkürzung des Diagnosevorgangs. Da die Kosten von Reparaturen des Fahrzeugs im Gewährleistungszeitraum vom Hersteller übernommen werden müssen, können enorme Kosten entstehen (vgl. Abbildung 1.2). Diese Kosten setzen sich aus den Teilekosten sowie dem Arbeitslohn bzw. -kosten für die benötigte Reparaturzeit zusammen. Hier kann die Inferenz durch eine schnellere Bestimmung der Fehlersuchzeit die Reparaturdauer verkürzen und so die Gesamtkosten einer Reparatur senken.

Die Fähigkeit, enorme Zustandsräume selektiv und somit effizient durchsuchen zu können, ermöglicht im Mittel ein deutlich besseres Laufzeitverhalten als bei Diagnoseverfahren, die den Suchraum sequentiell, bspw. in Form einer Breitensuche, durchsuchen. Zudem ermöglicht ein solches Verfahren eine umfangreichere Diagnose verteilter Software-Umfänge als bisher einzusetzen, da aufgrund der effizienten Verarbeitung der limitierende Faktor Zustandsraum aufgehoben wird. Dieser Aspekt ist vor allem unter der Tatsache zu sehen, daß der Software-Wachstum im Fahrzeug anhalten wird.

Schließlich erlaubt ein besseres Laufzeitverhalten der Diagnoseinferenz umfangreichere Auswertungen in der vorgesehenen Zeitspanne für die Inferenz und ermöglicht so die Berechnung eines detaillierteren, besseren Ergebnisses. Eine detaillierte Inferenz erleichtert zudem in den Werkstätten die Reparaturen.

3.6 Diagnose von verteilten Funktionen

Potential

Im Abschnitt 3.4 wurde dargelegt, daß die Spezifikation verteilter, komponentenübergreifender Funktionen verbesserbar ist. In diesem Abschnitt wird genauer auf die Diagnose verteilter Funktionen eingegangen. Wie in den Abschnitten 2.2.2 und 2.3.4 dargelegt, wird das funktionale Verhalten der verteilten Funktion von der Diagnose auf von der Spezifikation abweichendes Verhalten überwacht und im Falle einer erkannten Abweichung die Ursache einer austauschbaren Einheit (Komponente) zugewiesen.

Obwohl im Entwicklungsprozeß durch den OEM auf Gesamtfahrzeug- sowie Systemarchitekturebene komponentenübergreifende Funktionen spezifiziert werden, ist diese Spezifikation nicht ausreichend. Dies liegt daran, daß die Interaktionen der Funktionen nicht ausreichend erfaßt bzw. spezifiziert werden und zu wenig Wissen über die Zusammenhänge und Interaktionen der einzelnen Funktionen vorliegt. Dies betrifft sowohl das Normalverhalten, als auch das fehlerhafte Verhalten und wird durch transitive Abhängigkeiten zwischen den Funktionalitäten vergrößert.

Oft liegt der Grund für das fehlende Gesamtverständnis einer verteilten Funktion darin, daß die einzelnen Teilfunktionalitäten der verteilten Funktion von verschiedenen Zulieferern und vom OEM erstellt werden.

Eine zusätzliche, enorme Herausforderung für die Diagnose besteht in der Überwachung des exponentiell wachsenden Zustandsraumes, der durch das Zusammenspiel mehrerer, verteilter Funktionen entsteht. Auf diese Herausforderung wird in Abschnitt 3.5 näher eingegangen.

Lösungsansatz

In der Arbeit wird ein Ansatz für die Diagnose verteilter Systeme vorgestellt, der sowohl die on- und off-board Diagnose als auch ihre Spezifikation erfaßt.

Die Diagnose für die verteilten Funktionen wird auf drei Abstraktionsebenen spezifi-

ziert, die die Rollen eines Diagnosevorfalls abdecken. Dabei wird auf den Funktionen und ihren Interaktionen, die beispielsweise mittels *Message Sequence Charts* [MSC04] formal spezifiziert werden können (vgl. [Krü00] sowie Abschnitt 4.6), eine erweiterte FMEA zur Erfassung des möglichen fehlerhaften Verhaltens durchgeführt.

Aus den MSC mitsamt ihren durch die Diagnose notwendigen Änderungen und Erweiterungen kann dabei automatisiert Code generiert werden (vgl. [Krü00, Kap. 7]). Es ist aber auch möglich, aus den Daten der erweiterten FMEA direkt den Diagnose-Code für sowohl Steuergerät als auch den Tester der Werkstätten zu generieren.

Eine genauere Darstellung des Ansatzes findet sich in Kapitel 4. Der Ansatz wird zudem in einen Vorschlag für einen Entwicklungsprozeß für die Diagnose eingebettet.

Nutzen

In der Dissertation wird ein Ansatz für die Diagnose verteilter Funktionen vorgestellt, der als erster Ansatz sowohl die Spezifikation als auch die Implementierung auf Steuergerät *und* Tester für verteilte Funktionen umfaßt. Ziel des Ansatzes ist eine Steigerung der Qualität der Diagnose verteilter Funktionen.

3.7 Kein gesamthafter Diagnoseansatz für alle Komponenten

Potential

Sowohl Zulieferer als auch OEM bedienen sich verschiedener Diagnoseansätze für die Steuergeräte. Für die Software-intensiven Multimedia-Systeme des Fahrzeugs werden meist auf Expertenwissen (vgl. Abschnitt 2.4.1) basierende Verfahren eingesetzt, für die Regelsysteme des Fahrzeugs häufig der FDI-Ansatz (vgl. Abschnitt 2.4.4).

In Abschnitt 2.4 wurden die gängigsten Diagnoseansätze der Automobilbranche vorgestellt und gezeigt, daß diese sich sowohl in der Speicherung des diagnoserelevanten Wissens als auch in der Fehlerursachenbestimmung deutlich unterscheiden. Durch die verschiedenen Arten der Erfassung des Diagnosewissens ergibt sich somit ein erhöhter Wartungsaufwand.

Zusätzlich führt der Einsatz verschiedener Diagnoseansätze zu einem größeren Koordinationsaufwand bei der Integration von Teilfunktionen zu einer Gesamtfunktion sowie bei der Integration von Komponenten in das Fahrzeuggesamtsystem, da die lokale Symptomgenerierung und Inferenz auf der Ebene der Gesamtfunktion bzw. des Gesamtsystems abgestimmt werden muß.

Weiterhin besteht die Notwendigkeit für den OEM, das diagnoserelevante Wissen für die Werkstatt verfügbar zu machen. Da die off-board Diagnose meist den im Steuergerät nicht einsetzbaren fallbasierten Ansatz verwendet, muß das Wissen transformiert werden, was wiederum den Wartungsaufwand zusätzlich erhöht. Im Abschnitt 3.2 wurden die Vorteile der automatischen Generierung der Diagnose aus einer Datenquelle besprochen. Die automatische Generierung hilft den Erstellungs- und Wartungsaufwand aufgrund der verschiedenen Wissenserfassungen zu reduzieren.

Ein noch größeres Potential birgt jedoch die Entwicklung eines Diagnoseansatzes, der in der Lage ist, sowohl die Anforderungen an die on- als auch off-board Diagnose zu erfüllen oder gar einen gesamthaften Ansatz zu bieten. Dies ist um so mehr relevant, als daß alle vorgestellten Diagnoseansätze der Automobildomäne Optimierungspotentiale bieten. Im Folgenden werden diese Ansätze diskutiert und anhand von definierten Kriterien bewertet.

3.7.1 Bewertungskriterien Diagnose

In diesem Abschnitt werden die Kriterien für die vergleichende Bewertung der Diagnoseansätze aufgelistet. Die Kriterien basieren dabei teilweise auf einzelnen grundlegenden Bewertungsgrößen, die Ausgangspunkt einer von BMW in Auftrag gegebenen Vergleichsstudie verschiedener Diagnoseverfahren waren [CTS02]. Diese Kriterien werden aber an die in diesem Kapitel bisher herausgearbeiteten Punkte angepaßt und um die Kriterien Strukturierbarkeit, formale Spezifikation, Wartungsaufwand, Robustheit sowie effiziente Inferenz erweitert. Aufgrund der geänderten und neuen Kriterien ergeben sich somit auch unterschiedliche Ergebnisse als in [CTS02].

Datenbeschaffung. Bewertet den nötigen Aufwand zur Erfassung des Diagnosewissens. Je leichter die Daten für das Diagnosewissen einholbar sind, desto besser wird bewertet. Hierbei ist besonders die Einbindung der Zulieferer zu beachten (vgl. Abschnitt 3.3).

Automatisierte Erstellbarkeit. Bewertet, ob sich die Diagnose und ihr Datenmodell automatisiert erstellen lassen. Eine automatisierte Erstellung reduziert den Wartungs- und Erstellungsaufwand deutlich (vgl. Abschnitt 3.2.2). Die Bewertung erfolgt abhängig vom möglichen Grad der Automatisierung.

Erstellungsaufwand Diagnosemodell. Bewertet den notwendigen Aufwand für die Erstellung des Diagnosemodells. Diagnoseverfahren, die ein internes Modell der zu diagnostizierenden Komponente benötigen, bedürfen mehr Aufwand für die Erstellung des Diagnosemodells als Diagnoseverfahren mit sogenannter Black Box-Sicht. Je geringer der Aufwand zur Erstellung eines Diagnosemodells ist, desto besser wird bewertet.

Strukturierbarkeit. Bewertet, ob ein strukturiertes Diagnosemodell des Systems erstellt werden kann. Ein strukturiertes Modell erleichtert die Wiederverwendung von Diagnoseelementen sowie die Generierung der Diagnose für sowohl Steuergerät als auch Werkstatt (vgl. Abschnitt 3.8). Zudem kann aus strukturierten Diagnosemodellen für Funktionen oder Komponenten leichter ein Modell des Gesamtsystems erstellt werden.

Formaler Ansatz. Bewertet, ob die Diagnosemodelle formal spezifiziert werden können (vgl. Abschnitt 3.4). Formale Methoden reduzieren Design-Fehler deutlich (*Bauer-Zemanek-Hypothese* [ER03, Kap. 3.3.8], entnommen aus [BW82] und [Zem68]). Ein formales Diagnosemodell erleichtert zudem deutlich die Validierung der Diagnosefunktionen in den Test- und Integrationsphasen.

Komplexität des Ansatzes. Bewertet die Komplexität des Diagnoseansatzes und insbesondere die des Diagnosemodells. Ein wichtiger Punkt hierbei ist die Speicherung des Diagnosemodells. Ein hochgradig in einem speziellen Format kodiertes Diagnose-

modell ist selbst für Experten oft nur schwerverständlich. Somit erhöht sich neben dem Wartungs- auch der Koordinationsaufwand beim Austausch des Modells zwischen Zulieferer und OEM.

Wartungsaufwand. Bewertet den Aufwand zur Wartung des Diagnosemodells. Aufgrund des langen Lebenszyklus der Diagnose von bis zu 15 Jahren nach Ende der Serienproduktion eines Fahrzeugs (vgl. Abschnitt 2.1.2) ist dieser Faktor enorm wichtig. Strukturierte, formale bzw. modellbasierte Diagnoseansätze bedürfen eines geringeren Wartungsaufwands verglichen mit auf Expertenwissen basierenden Verfahren.

Erweiterbarkeit. Bewertet, wie gut sich das Diagnosemodell erweitern läßt. Aufgrund der verschiedenen Systemausstattungen (vgl. Abschnitt 2.1.2) kann es notwendig sein, das Diagnosemodell zu erweitern, um die erweiterten Funktionalitäten mit Diagnose abzudecken.

Einsetzbar im Steuergerät. Bewertet, ob das Verfahren im Steuergerät unter Echtzeitbedingungen einsetzbar ist. Darunter wird verstanden, ob ein komplexes, rechenaufwendiges Verfahren zur Generierung der Symptome und Inferenz eingesetzt wird.

Integrierbarkeit. Dieser Faktor bewertet, inwieweit ein Diagnoseansatz bzw. seine Ergebnisse sich in die Diagnose eines Gesamtsystems integrieren lassen. Dieser Faktor ist besonders für die Diagnose verteilter Funktionen im Spannungsfeld zwischen OEM und Zulieferer wichtig (vgl. Abschnitt 3.3), da hier die Diagnose der Gesamtfunktion sich aus den lokalen Diagnose sowie Symptomen mehrerer Teilsysteme zusammensetzt.

Robustheit. Hiermit wird bewertet, wie robust der Diagnoseansatz gegenüber möglicherweise fehlerbehafteten oder gar unvollständigen Eingangsgrößen ist. [PBKS07] weist darauf hin, daß aufgrund der Komplexität und Größe von automobilen Systemen ein vollständiges Wissen über die gesamten Subsysteme und das Gesamtsystem eine Illusion sei. Je stabiler der Diagnoseansatz gegenüber möglicherweise fehlerhaften oder unvollständigen Werten ist, desto höher wird der Ansatz bewertet.

Umgang mit unbekannten Fehlern. Bewertet, wie gut das System in der Lage ist, Fehler zu entdecken, die bei der Erstellung des Modells oder Diagnosewissen nicht bekannt waren. Es wird bewertet, ob das System in der Lage ist, solche Fehler dennoch zu entdecken. Weiterhin fließt in die Bewertung ein, ob der Diagnoseansatz die Entwicklungsabteilungen bei der Analyse des Fehlers unterstützen kann, beispielsweise in Form eines System-Traces wie bei der Discrete Event System-Diagnose.

Effiziente Inferenz. Dieser Punkt wurde genauer in Abschnitt 3.5 diskutiert. Die Inferenzgeschwindigkeit ist sowohl für die präventive Diagnose als auch für die Werkstattdiagnose essentiell. Je schneller die Inferenz im Steuergerät erfolgen kann, um so eher können Fehlervermeidungsstrategien gestartet werden und dadurch Versagen des Systems vermieden werden. Weiterhin kann bei schneller Inferenz der Zeitpunkt des Eingreifens genauer angepaßt werden, um so Kundenirritationen zu vermeiden. Beispielhaft hierfür sind Kundenbeanstandungen von Fahrassistenzsystemen, die durch ihr für den Kunden als zu früh empfundenes Eingreifen gewolltes „sportliches Fahren“ verhinderten. Eine schnellere Inferenz in der Werkstatt hingegen kann wie erwähnt die Reparaturdauer verkürzen und somit die Kosten senken.
Weiterhin ist der Faktor Inferenz besonders bei der Durchsuchung großer Zustands-

räume, wie sie bei Software-Systemen vorliegen, enorm wichtig. Verfahren, die in der Lage sind, große Zustandsräume effizient zu durchsuchen bzw. den Suchraum einzuschränken, sind deutlich vorteilhafter zu sehen als Verfahren, die den gesamten Zustandsraum sequentiell durchsuchen müssen.

Eignung für präventive Diagnose/ Prädiktion. Hiermit wird bewertet, ob das Diagnoseverfahren in der Lage ist, einen Fehler rechtzeitig zu erkennen bzw. einen sich anbahnenden Fehler zu erkennen. Da solche Fehler in sehr kurzen Zeitabständen auftreten können, sind rechenaufwendige Verfahren bzw. Verfahren mit komplexen Diagnosemodellen prinzipiell weniger für die präventive Diagnose geeignet. Die Abgrenzung zum Inferenzkriterium liegt darin, daß bei dem Prädiktionskriterium bewertet wird, ob der Diagnoseansatz zyklisch das System überwacht und einen fehlerhaften Zustand beispielsweise wie die FDI-Diagnose durch eine Trend-Analyse vorhersagen kann.

Verständlichkeit der Diagnose. Dieses Kriterium bewertet, inwieweit die Ergebnisse der Diagnose verständlich für Werkstatt und Entwicklung sind. Hierbei ist besonders die Güte der Fehlerlokalisierung wichtig; also ob der Ansatz nicht nur sagen kann, daß ein Fehler vorliegt, sondern auch sagen kann, *wo* in welcher (Sub-)Komponente der Fehler vorliegt. Unverständliche Diagnoseergebnisse führen in der Werkstatt zur Anwendung zusätzlicher oder wiederholter Reparaturmaßnahmen und somit zu erhöhten Kosten und Kundenunzufriedenheit.

3.7.2 Vergleich der Diagnoseansätze

Die in Abschnitt 2.4 vorgestellten Diagnoseansätze werden anhand der Kriterien des vorigen Abschnitts bewertet. Um diesen Abschnitt kurz zu halten, werden nur die wichtigsten Punkte bewertet und referenziert. Tabelle 3.3 zeigt die Gesamtbewertung aller für die Arbeit relevanten Ansätze. Eine kurze übersichtliche Bewertung der Ansätze wurde zudem in [KB10, Kap. 2] gezeigt.

Expertenwissen-basierende Diagnoseansätze

Vor- und Nachteile des regelbasierten Schließens. Regelbasierte Systeme (vgl. [RN10, Kap. 7.5 und 9.3]) sind aufgrund ihrer *geringen Komplexität* relativ leicht zu verstehen und *einfach zu erstellen*. Aufgrund der einfachen Erstellung sowie der meist auch für Nicht-Experten *gut verständlichen Diagnose* werden sie oft für Software-intensive Systeme *im Steuergerät eingesetzt*. Da eine Black Box-Sicht auf andere Komponenten im Ansatz kodierbar ist, kann der Ansatz auch *durchschnittlich integriert* werden.

Aufgrund der schlechten Strukturiertheit des Diagnosewissens ergibt sich eine *erschwerte, schlechte Wartbarkeit* sowie *Erweiterbarkeit* des Systems, die sich mit Wachstum des Systems zudem verschlechtert. Weiterhin ist das für das Diagnosewissen notwendige Expertenwissen *schwer beschaffbar* und komplex zu lernen. Dies ist um so kritischer zu sehen als daß die Modelle auch auf expliziten Daten von Zulieferern basieren.

Weiterhin ist die *Inferenz* bei einer Wissensbasis mit N Schlußfolgerungen und K Prämissen im schlechtesten Fall mit N^K als exponentiell zu sehen. Um den Ansatz

robuster machen zu können, wurden Wahrscheinlichkeitsfaktoren eingebaut. Russel und Norvig merken jedoch an, daß mit Vergrößerung der Regelbasis unerwünschte Interaktionen zwischen den Regeln anstiegen und somit die *certainty factors* von Hand optimiert werden mußten [RN10, S.549].

Vor- und Nachteile des Bayes'schen Netzes. Bayes'sche Netze (vgl. [Pea86], [RN10, Kap. 14]) ermöglichen durch die sogenannte *d-Separierung* [RN10, Kap. 14.2.2] eine *Strukturierung* des Diagnosewissens und Reduktion des gesamten Wissens auf das für das vorliegende Diagnoseproblem notwendige Wissen. Zudem ist der Ansatz in der Lage, neue Abhängigkeiten zu lernen [Pea86, Kap. 6].

Da für jedes Element des Netzwerks sein Wahrscheinlichkeitswert in Abhängigkeit seiner verbundenen Elemente berechnet werden muß, ist das System *schlecht wartbar*. Die Bestimmung der unabhängigen Größen für die Separation wird zudem erschwert durch die vielen funktionalen Abhängigkeiten, die teilweise auch Experten unbekannt sind. Eine nicht sorgfältige Separation führt aber dazu, daß der Rechenaufwand bei großen Netzen und somit die *Inferenz* mit langen Abhängigkeitsketten exponentiell werden kann [RN10, Kap. 14.4.3]. Trotz Separation bleibt der Rechenaufwand relativ groß und der Ansatz somit für die Diagnose von *Echtzeitsystemen* kaum einsetzbar.

Vor- und Nachteile des fallbasierten Schließens/ Prüfplans. Der Vorteil des fallbasierten Schließens [AW91, AP94] ist darin zu sehen, daß der Ansatz auch für Systeme mit wenig vorhandenen Informationen oder vielen unbekannten Zusammenhängen angewendet werden kann und das System *leicht erstellbar* ist. Der Ansatz kann mit einer relativ geringen Anzahl von Fällen gestartet werden, die sukzessive durch erfolgreich gelöste Fälle erweitert werden.

Das Verfahren ermöglicht es Experten, ihr Wissen relativ detailliert und schnell und dabei dennoch *leichtverständlich* einzugeben, beispielsweise in Form eines Sequenzdiagramms. Im Prüfplan können zudem nicht nur die Aufgaben der Diagnose, sondern auch weitere relevante Handlungsabläufe eingegeben werden. Dies macht den Ansatz für die Werkstätten sehr interessant, da ein komplettes Service-Handbuch erstellt werden kann und somit die Ergebnisse ebenfalls *leichtverständlich* sind.

Problematisch ist jedoch, daß die Wissensbasis nur das schon bekannte Fehlerwissen abdecken kann, was die Diagnose neuer Systeme deutlich erschwert. Hinzu kommt, daß die Wissensbasis nur relativ *aufwendig wartbar* ist. Nachteilig ist zudem, daß sich aufgrund der Beschränkung auf das vorhandene Wissen *unbekannte Fehler nicht entdecken lassen*, da nach diesen nicht gesucht wird/werden kann. Deshalb lassen sich komplexe Systeme mit vielen, teils nicht bekannten Einflüssen nur äußerst schwer mit diesem Ansatz modellieren. Bei der *Inferenz* dieses Ansatzes handelt es sich um die schon erwähnte Breitensuche [Knu97].

Einem Einsatz im Steuergerät steht zusätzlich im Wege, daß der Ansatz sich im Grunde genommen für eine Fehlerentdeckung und Fehlerbestimmung nicht eignet, sondern mehr für eine Reparaturanleitung.

Modellbasierte Diagnose

Die modellbasierte Diagnose ist ein *formaler Ansatz*, der auf logischen Formeln basiert. Bei der modellbasierten Diagnose wird das Systemmodell von den Inferenzregeln getrennt, was den *Wartungsaufwand reduziert* und die Wiederverwendbarkeit steigert. Die modellbasierte Diagnose versucht mittels Beobachtungen Abweichungen des unter Beobachtung stehenden Systems von seiner Spezifikation zu entdecken.

Es existieren mehrere Arbeiten, die zeigen, wie sich aus einer FMEA automatisch ein Systemmodell für die modellbasierte Diagnose erstellen läßt (vgl. bspw. [PPWS95, PT02] oder [Fra09]). Da eine FMEA für die meisten sicherheitsrelevanten Systeme erstellt werden muß, ist hier die *Datenbeschaffungsmöglichkeit als positiv* zu sehen.

Ein großer Vorteil der modellbasierten Diagnose ist die Möglichkeit des quantitativen sowie qualitativen Schließens (vgl. [Kui94]). Während beim quantitativen Schließen der ganze Wertebereich einer Variablen erfaßt wird, werden beim qualitativen Schließen nur die wesentlichen Aspekte einer Größe im Modell gespeichert. Eine Eingangsgröße wie der gemessene Druck ließe sich qualitativ beispielsweise mit „Druck zu hoch“ oder „Druck zu niedrig“ modellieren, ohne den ganzen Wertebereich darstellen zu müssen. Die qualitative Modellierung *steigert die Robustheit* des Systems, kann die *Verständlichkeit der Diagnose erhöhen* sowie die *Komplexität des Systemmodells verringern*. Weiterhin ermöglicht die qualitative Modellierung eines Systems eine Abstraktion von den genauen Details der Implementierung des Systems. Dadurch ist die modellbasierte Diagnose in der Lage, Systeme zu diagnostizieren, ohne deren innere Vorgänge im Detail kennen zu müssen. Somit kann das Verfahren mit anderen Systemen *gut gekoppelt* werden. Die modellbasierte Diagnose mit quantitativem Schließen wird im folgenden Abschnitt (Fault Detection and Isolation) genauer dargestellt.

Dennoch ist das Systemmodell abhängig vom gewünschten Detaillierungsgrad sowohl *kosten-* als auch *zeitaufwendig erstellbar*. Zwar läßt sich das Model leichter *erweitern* als bei den heuristischen Systemen, dennoch ist die Erweiterbarkeit zeitintensiv aufgrund des Fehlens einer *expliziten Struktur* des Modells.

Mikaelian et al. weisen darauf hin, daß modellbasierte Diagnoseverfahren zumeist zur Analyse von mechanischen oder elektrischen Systemen eingesetzt werden, *nicht* jedoch *für Software-lastige Steuergeräte*, da diese einen viel höheren Zustandsraum haben. Die modellbasierte Diagnose sei eher gedacht für komponentenbasierte, statische Analyse von Systemen wie beispielsweise Schaltgitter [MWS05]. Dies zeigt sich darin, daß das zeitliche Verhalten des Systems nicht explizit erfaßt wird. Es ist also kein System-Trace vorhanden, der die Analyse *unbekannter Fehler unterstützen* könnte. Dennoch ist die modellbasierte Diagnose aufgrund ihrer Robustheit und der Fähigkeit der Black Box-Diagnose in der Lage, diese Fehler zu erkennen, falls sie zu abweichenden Verhalten des beobachteten Systems führen.

Der andere Grund ist, daß die *Inferenz* bzw. Bestimmung der Diagnosen vor allem bei mehreren fehlerhaften Komponenten *exponentiell* ansteigen kann. De Kleer zeigt in [KK03, Kapitel 6.3], daß es ab einer Anzahl von ungefähr 60 Komponenten zu einem exponentiellen Anstieg der zur Bestimmung der Diagnose notwendigen minimalen Conflict Sets kommt. Das andere gezeigte Inferenzverfahren, der Conflict Set Algorithmus [Rei87], berechnet zur Bestimmung der Diagnosen das minimale Hitting Set für eine Menge von Conflict Sets. Die Bestimmung der Hitting Sets wird

auch als *transversal problem* bezeichnet. Bauer [Bau06, S.111] verweist auf [EG95], die dieses Problem als NP-vollständig identifizierten. Bei der Durchsuchung des Zustandsraum der Conflict Sets handelt es sich um eine Breitensuche (vgl. [Knu97]), deren Aufwand $\mathcal{O} = (|\#Knoten| + |\#Kanten|)$ bei einem konstanten Suchraum entspricht. Der Suchraum ist konstant, da das Verfahren den Suchraum im worst case vollständig durchsuchen muß und den Suchraum zudem nicht einschränkt, bspw. durch Heuristiken.

Aufgrund des relativ hohen Rechenaufwands ist auch die Eignung der modellbasierten Diagnose hinsichtlich *präventiver Diagnose* kritisch zu sehen.

Fault-Detection and Isolation

Der FDI-Ansatz ist ein modellbasiertes Verfahren, das prinzipiell für die Diagnose von dynamischen Systemen wie beispielsweise Regelungssystemen entwickelt wurde. Das zu überwachende Regelsystem mitsamt seinen Ein- und Ausgangsgrößen wird *formal* in Form eines mathematischen Modells erfaßt. Die Bandbreite der zu überwachenden Systeme und damit die *Komplexität des Diagnosemodells* reicht dabei von *einfachen* signalbasierten Systemen bis hin zu *komplexen* mathematischen Modellen, die mittels Differentialgleichungen das System beschreiben. Aufgrund der möglichen Komplexität des Modells lassen sich die Modelle meist *nicht automatisiert erstellen*.

Der FDI-Ansatz läßt sich *gut in Steuergeräten einsetzen* und ist auch für die *präventive Diagnose geeignet*. Dennoch ist besonders bei komplexen Systemen ein hoher Berechnungsaufwand notwendig. Da die Diagnosemodelle Verhalten des Systems über Zeit erfassen können und zudem Trend-Änderungen von zu überwachenden Größen analysiert werden können, ist die *Robustheit* als *gut* zu betrachten.

Nachteilig ist, daß dieser Ansatz nicht für die Werkstattdiagnose geeignet ist, da bei einer Reparatur anfallende Größen wie Kundenaussagen oder Werkstattmessungen nicht in das System integriert werden können. Dies betrifft auch die *Integration* des Ansatzes in andere Diagnoseverfahren.

Discrete Event System-Diagnose

Die Discrete Event System-Diagnose (DES-D) ist ein modellbasierter, *formaler*, *strukturierbarer* Diagnoseansatz, der zudem ermöglicht, einzelne Modelle in ein Gesamtmodell per *Komposition zu integrieren*. Aufgrund der Strukturiertheit ist auch die *Erweiterbarkeit prinzipiell gut*, wird aber durch die später erwähnte Modellkomplexität eingeschränkt. Die Inferenz besteht aus der Analyse eines mitlaufenden Fehler-Traces und kann linear erfolgen. Aufgrund der *linearen Inferenz* ist die DES-D auch im *Steuergerät einsetzbar* und für *präventive Diagnose geeignet*. Der durch den Ansatz automatisch erstellte System-Trace ist zudem ein *große Hilfe bei der Analyse unbekannter Fehler*. Ein weiterer Vorteil der DES-D ist, daß sich die minimale Anzahl an Beobachtungspunkten formal berechnen läßt (das Diagnosability-Problem, vgl. [SSL+94]).

Dennoch ist die DES-D in der Praxis schwierig anzuwenden, da für jede Komponente ein eigenes Verhaltensmodell erstellt werden muß, das auf Zuständen basiert. Wie in Abschnitt 3.3 erwähnt, sind aufgrund des beschränkten Wissens um Zulieferkompo-

nenten oft nicht alle Zustände von Systemen bekannt. Die *Datenbeschaffung* ist hier als *kritisch* zu sehen. In solchen Fällen muß die DES mit Black Box-Sicht und Ein- und Ausgangsgrößen anstelle der geforderten White Box-Sicht arbeiten. Zudem verfügen viele Systeme über enorm viele Zustände mit vielen Abhängigkeiten. Dies macht die Systemmodellierung zu einer enorm komplexen Aufgabe (vgl. [PBKS07, S. 14]) und steigert somit den ohnehin recht *hohen Erstellungs-* und *Wartungsaufwand*. Auch hier wirkt sich wieder der eingeschränkte Einblick in Zulieferkomponenten negativ aus.

Schließlich können bei der DES-D Daten wie Kundenaussagen oder Werkstattmessungen nicht eingebaut werden, da diese ja Aussagen oder Messungen, aber keine Zustände sind. Dennoch sind diese Daten für Reparaturen enorm wichtig, um alle möglichen Fehler abzudecken.

Diagnose als boolesches Erfüllungsproblem

Die von Bauer [Bau06] und Tripakis [Tri09] vorgeschlagene Transformation der Diagnose in ein boolesches Erfüllungsproblem ist eigentlich ein auf [Rei87, KW87] basierender modellbasierter Diagnoseansatz, bei dem aber die vorher erwähnten (langsamen) Inferenzalgorithmen durch einen SAT-Solver ersetzt werden. Für den Ansatz gelten also die schon vorher erwähnten Vorteile der modellbasierten Diagnose, die hier nur dann erläutert werden, falls sie sich von der modellbasierten Diagnose unterscheiden.

Ein Vorteil des Ansatzes ist die Abbildung der Symptomauswertung auf boolesche Formeln. Durch die Transformation der Ergebnisse der generierten Symptome auf die Wertemenge $\{0, 1\}$ bzw. Wahrheitswerte falsch und wahr können so verschiedene Diagnoseansätze eingebaut werden. Diese Fähigkeit *empfiehlt* den Ansatz für eine zentrale Analyse von Teilfunktionen im *Steuergerät*, da zudem auch Symptome über einen Zeitverlauf analysiert werden können.

Aufgrund der speziellen Struktur der Variablenspeicherung, dem Einsatz von Heuristiken beim Lösen (vgl. [GPFW96, Zha03] sowie Abschnitt 4.7) ist der SAT-Solver deutlich besser in der Lage, große Zustandsräume zu durchsuchen [GPFW96]. Dies ist besonders für verteilte Software-Systeme relevant, da hier das als *state explosion* bekannte Problem der exponentiellen Vergrößerung des Zustandsraums auftreten kann (vgl. [VW94]). Sind beispielsweise für eine verteilte Funktionalität 6 Komponenten involviert, so ergeben sich selbst für eine geringe Anzahl von 10 Zuständen pro Komponente ein gesamthafter Zustandsraum von 10^6 (vgl. Abschnitt 3.5).

Da der SAT-Solver in der Lage ist, größere Zustandsräume effizienter zu durchsuchen als die Breitensuche der modellbasierten Diagnose, kann der Einsatz *deutlich besser* für die *präventive Diagnose* eingesetzt werden. Dies geschieht beispielsweise durch Eingrenzung des Suchraums durch Einsetzen von Heuristiken.

Dennoch steht einem Einsatz in *Steuergeräten der SAT-Solver entgegen*. Um die Diagnosen zu berechnen, verwendet der SAT-Solver Rekursionen und fordert dynamisch Speicher an. Diese dynamische Allokation wird von MISRA-Regel 20.4 [Mot04] nicht erlaubt. Jedoch wird in Abschnitt 4.7.3 eine Lösung hierfür gezeigt.

Weiterhin wächst auch bei diesem Ansatz der Zustandsraum enorm. Dies liegt am Aufsetzen des Ansatzes auf der modellbasierten Diagnose und an einer *fehlenden*

Strukturierung des Diagnosemodells, die den Suchraum des Solvers einschränken kann.

Rollenbasierte Diagnose

Die rollenbasierte Diagnose, die in Kapitel 4 dieser Arbeit eingeführt wird, setzt auf der Diagnose als boolesches Erfüllungsproblem auf und behebt deren erwähnten Schwachstellen.

Beispielhaft sei hier die Einführung einer Struktur für das Diagnosemodell genannt, die zum einen die *Komplexität des Modells reduziert* und somit die Wiederverwendbarkeit erhöht und den *Wartungsaufwand deutlich reduziert*, zum anderen aber durch die Strukturierung den Suchraum des SAT-Solvers zusätzlich reduziert und somit die *Inferenz weiter steigert*. Zudem *erleichtert* die Strukturierung des Datenmodells die *Erweiterbarkeit des Diagnosemodells*.

Zusätzlich wird durch eine Abbildungsvorschrift zwischen dem Datenmodell der Arbeit und der rollenbasierten Diagnose eine *automatische Erstellung* der Diagnose ermöglicht und somit der *Erstellungsaufwand weiter reduziert*.

3.7.3 Zusammenfassung

Kriterium	Bayes	MYCIN	Prüfplan	Regelbasiert	DES	FDI	Modellbasiert	SAT	**rollenbasiert** (→ Kap. 4)
Datenbeschaffung	-	-	+	-	-	0	+	+	+
automatisierte Erstellbarkeit	-	0	-	0	-	-	+	+	+
Erstellungsaufwand	-	+	+	+	-	0	-	-	+
Strukturierbarkeit	+	-	-	-	+	-	-	-	+
formaler Ansatz	+	-	-	-	+	+	+	+	+
Komplexität Ansatz	-	+	+	+	-	0	+	+	+
Wartungsaufwand	-	-	-	-	-	0	+	+	+
Erweiterbarkeit	-	-	+	-	+	0	0	+	+
Einsetzbar im Steuergerät	-	-	-	+	+	+	-	+	+
Integrierbarkeit	-	-	-	0	+	-	+	+	+
Robustheit	-	-	-	-	+	+	+	+	+
Umgang mit unbekannten Fehlern	+	-	-	-	+	+	+	+	+
Inferenz	-	-	-	-	+	+	-	+	+
Eignung präventive Diagnose	-	0	-	0	+	+	-	+	+
Verständlichkeit Diagnose	0	+	+	+	0	+	+	+	+

Tabelle 3.3: Übersicht Bewertung Diagnoseansätze. Quelle: eigene Darstellung

Tabelle 3.3 zeigt eine Bewertung der in der Dissertation vorgestellten Diagnoseansätze. Dabei stellt + eine positive, − eine schlechte und 0 eine durchschnittliche Bewer-

tung dar. Bei der Bewertung handelt es sich um eine Zusammenfassung der vorigen Abschnitte.

Der *Fault Detection and Isolation*-Ansatz ist in der Werkstatt nicht einsetzbar, da sich informelle Texte wie bspw. Kundenaussagen nicht integrieren lassen. Bei Software-intensiven Steuergeräten wächst der Suchraum, den die *modellbasierte Diagnose* durchsuchen muß, exponentiell an, wodurch die Inferenz bei diesen Systemen nicht praxistauglich durchgeführt werden kann. Auf *Expertenwissen* basierende Diagnoseverfahren sind hingegen schlecht anpaßbar, was den Erstellungs- und Wartungsaufwand drastisch erhöht. Die erwähnten Punkte treffen auf die *Discrete Event System*-Diagnose nicht zu, dennoch ist hier kritisch zu sehen, daß der Ansatz Probleme mit der Diagnose von Komponenten mit eingeschränkter Einsicht hat, wie dies bei Zulieferkomponenten häufig vorkommt. Nachteilig bei der Anwendung des *SAT-Solvers* ist die mangelnde Wartbarkeit sowie Echtzeitfähigkeit aufgrund der MISRA-Regel.

Als Fazit läßt sich ziehen, daß es bisher keinen Ansatz gibt, der sowohl im Steuergerät als auch im Tester effizient einsetzbar ist. Von allen vorgestellten Ansätzen schneidet gesamt gesehen der *SAT-Solver* Ansatz am besten ab.

Dieses Verfahren wird deshalb als Ausgangspunkt für den Ansatz der Dissertation gewählt und zudem in den erwähnten problematischen Punkten verbessert. So wird beispielsweise in der Methodik ein Verfahren gezeigt, mit dem die Problematik der dynamischen Speicherallokation vermieden werden kann.

Ziel des vorgestellten Diagnoseansatzes ist die Reduktion signifikanter Kostenfaktoren wie beispielsweise die Erstellungs- und Wartungskosten sowie die Erhöhung der Qualität der Diagnose durch den strukturierten, modellbasierten Aufbau.

3.8 Eingeschränkte Wiederverwendbarkeit der Diagnose

Potential

In Tabelle 3.2 wurde gezeigt, daß ein Großteil der automobilen Diagnose sowohl für das Steuergerät als auch für den Tester manuell programmiert wird, was die Wiederverwendung einschränkt. Dieses Problem wird durch die Tatsache der steigenden Funktionsumfänge sowie der Funktionsvielfalt aufgrund der hohen Variantenzahl des Fahrzeugs verschärft (vgl. Abschnitt 2.1.2).

In den letzten Jahren wurden in der Automobilbranche Methoden entwickelt, um diese steigenden Funktionsumfänge beherrschen zu können. Hierbei sind besonders die Definition von Funktionsarchitekturen (vgl. [WFH+06, Rit08]) oder Software-Produktlinien (vgl. [TH02]) zu nennen. Beide Ansätze sind mit Verfahren für mechanische Umfänge wie Baukasten oder Produktlinie verwandt, die schon seit längerer Zeit in der Automobilbranche eingesetzt werden (vgl. [Ren07], [EKL07, Kap. 7.12.6]).

Weiterhin wurden für die Erstellung der funktionalen Software-Anteile *Computer Aided Software Engineering* (CASE)-Tools eingeführt (vgl. bspw. [ECM91] oder [Bal98, Teil IV Kapitel 2.1]). CASE-Tools ermöglichen durch einen vollständig Tool-unterstützten modellbasierten Entwicklungsprozeß eine Erhöhung der Wiederverwendbarkeit der erstellten Umfänge sowie durch die automatische Generierung des Ziel-Codes aus

den Modellen eine Erhöhung der Qualität des Codes. Bekannte Beispiele hierfür sind AUTOFOCUS [HSSS96, BHS99], Matlab Simulink (vgl. [ABRW09]) sowie ASCET [ETA10].

Im Gegensatz hierzu existieren für die Diagnose keine automatisierten, umfassenden Erstellungsverfahren. Selbst bei Modellierung und Erstellung der Steuergerätefunktionalitäten mit CASE-Tools können nur Teilumfänge der Diagnose mit modelliert werden. Ein Großteil der Software-intensiven Steuergeräte wie beispielsweise Audio- und Navigationssysteme wird jedoch von Hand in Programmiersprachen wie C/C++ oder Java erstellt und somit auch deren Diagnoseanteile. Jedoch ist in diesem Fall der Aufwand des manuellen Programmierens für die Diagnose doppelt zu werten, da der off-board Anteil zusätzlich programmiert werden muß.

Die vorwiegend manuelle Kodierung der Diagnose hat mehrere Auswirkungen. Zum einen besteht ein *erhöhter Wartungsaufwand*, um den Diagnose-Code auf aktuellen Stand zu halten. Andererseits ist die Diagnose durch die manuelle Kodierung *schwer wiederverwendbar*. Diese beiden Faktoren erhöhen den ohnehin bestehenden *hohen Zeit-* und *Kostenaufwand* der manuellen Erstellung.

Zusätzlich steigern sich die erwähnten Probleme durch die *lange Betreuungsdauer der Diagnose* von bis zu 15 Jahren nach Ende der Serienproduktion (vgl. Abschnitt 2.1.2) durch das Auftreten neuer Fehlerbilder im Lebenszyklus und die kontinuierlich wachsenden Funktionsumfänge.

Lösungsansatz

Der Lösungsansatz baut auf dem in Abschnitt 3.2.2 erwähnten definierten Datenmodell sowie der in Abschnitt 3.4 vorgestellten formalen Spezifikation der Diagnose auf und umfaßt auch verteilte Funktionen (vgl. Abschnitt 3.6). Hier soll der Lösungsansatz nur kurz skizziert werden, für eine genauere Beschreibung sei auf die Abschnitte 4.4 sowie 4.6 verwiesen.

Auf einer Funktionsarchitekturstruktur mitsamt ihren Interaktionen wird eine erweiterte, geschichtete FMEA durchgeführt, um die möglichen, systematischen Fehler sowie Entdeckungs- und Vermeidungsmaßnahmen zu erfassen. Diese erfaßten Fehlerdaten werden auf aussagenlogische Formeln abgebildet. Diese Formeln können anschließend automatisiert in auf dem Steuergerät sowie Tester speicherbare Daten umgewandelt werden.

Das Datenmodell wird durch ein Schichtenmodell in drei Abstraktionsebenen eingeteilt, die den Handlungsrollen eines Diagnosevorfalls entsprechen. Die Abstraktionsebenen ermöglichen die Wiederverwendung von Diagnosedaten wie Abschnitt 4.2 zeigt.

Nutzen

Durch die Wiederverwendung von Daten aus dem Diagnosedatenmodell ähnlicher Komponenten oder überarbeiteter Versionen ergibt sich der Nutzen, Code nicht mehrfach erstellen zu müssen ([LG84, Jon94] sowie [Bal98, Teil IV Kapitel 3]). Dies ist besonders wichtig, da für die Diagnoseanteile im Gegensatz zu den funktionalen

Software-Anteilen die Möglichkeit der Wiederverwendung bisher stark eingeschränkt ist.

Dieser Faktor wird zudem dadurch erhöht, daß die Elemente der beiden oberen Schichten des vorgestellten Diagnosedatenmodells auch bei vielfältigen Änderungen der technischen Anteile meist wiederverwendet werden können. Dies liegt beispielsweise in der höheren Abstraktion der Werkstattebene durch den limitierten Zugriff derselbigen auf Steuergeräte begründet.

Die Wiederverwendung der Diagnosefunktionen ermöglicht zudem eine Erhöhung der Qualität [Bal98, Teil IV Kapitel 3.6]. So reduziert sich durch die Fokussierung auf die Wiederverwendung von funktionierender Diagnose die Gefahr, bekannte fehlerhafte Diagnosefunktionen oder bekannte Fehler ohne Abdeckung durch Diagnose ins Feld zu lassen.

Dennoch ist aufgrund des erwähnten hohen Diagnoseumfangs (vgl. Abschnitt 1.1 sowie [KKP+11]) von mehr als 3 Millionen Code-Zeilen die Möglichkeit der Kosteneinsparung durch die Wiederverwendung von vorhandener Diagnosefunktionen als wichtigster Nutzen zu betrachten.

3.9 Weitere Diagnosepotentiale

In diesem Abschnitt werden zusätzliche Potentiale aus dem gesamten Umfeld automobile Diagnose beschrieben.

Einbindung des Kunden. Die Fehlerklassifikation in Abschnitt 2.3.3 zeigt, daß zur Abdeckung aller möglichen Fehler der Automobildomäne der Kunde eingebunden werden muß. Die Kundenaussage ist einerseits wichtig, um vom Kunden fälschlicherweise als Fehler erkannte Verhalten des Fahrzeugs zu klären (Fehlerklasse 3).

Zum anderen wird die Kundenaussage benötigt, um die Analyse von unbekannten Fehlern zu erleichtern, indem vom Kunden eine möglichst detaillierte Aussage über die Fehlereffekte sowie den Fahrzeugstand während des Fehlers eingeholt wird. Dies trifft besonders auf sporadisch auftretende Fehler zu. Hier gewinnt die Kundenaussage zusätzlich an Wert, da das Fehlverhalten sich ja nicht oder nur schwer systematisch in der Werkstatt oder in den Entwicklungsabteilungen reproduzieren ließe.

Der in der Arbeit vorgestellte rollenbasierte Diagnoseansatz bildet den Kunden in der obersten Schicht des Abstraktionsmodells ab. Für jeden Fehler wird untersucht, ob und wie er für den Kunden sichtbar ist.

Einbindung Diagnose-Feedback. [KKP+11, Kap. 1] legt dar, daß täglich ca. 5 Gigabyte Diagnosedaten aus den Werkstätten an die Hersteller übermittelt werden, die das Verhalten der Fahrzeuge im Feld widerspiegeln. Diese Daten bieten wertvolle Hinweise über die Häufung und das Auftreten von Fehlern im Lebenszyklus sowie die Wirksamkeit der Diagnose im Steuergerät und Werkstätten.

Aufgrund der enormen täglichen Datengröße können diese Daten jedoch nicht manuell verarbeitet werden. In [Edl01, Kapitel 3.5] sowie [Koh06] wird dargelegt, daß

eine Nutzung gegenwärtig durch methodische Defizite in der Datenerfassung und -integration erschwert wird. Endres und Rombach bezeichnen diese Defizite als das *Librarian law*, das besagt, daß „the more knowledge that is available, the more effort has to be spent on the processes to use it“ ([ER03, Kap. 10.3.5], zitiert aus [Deg00]).

In Abschnitt 4.8 wird ein Ansatz vorgestellt, der zeigt, wie die Lebenszyklusdaten statistisch aufbereitet werden, um so für eine stetige Wartung der Diagnosedaten sowie zusätzlich für den von der ISO 9001 geforderten, kontinuierlichen Verbesserungsprozeß (KVP, [ISO08b]) der Diagnose eingesetzt werden können. Der Ansatz basiert dabei auf Vorarbeiten aus [Koh06, KKP+11] und ermöglicht eine automatisierte Erkennung von nicht (ausreichend) funktionierenden Diagnosefunktionen und unterstützt die Wartung dieser Funktionen. Weiterhin werden die statistischen Daten für eine Sortierung der Reparaturmaßnahmen nach Erfolgswahrscheinlichkeit genützt. Eine solche Sortierung ermöglicht dann den Werkstätten, die erfolgversprechendsten Maßnahmen für eine Reparatur zu wählen.

Generische Software-FMEA. In Abschnitt 2.3.3 wurde eine Fehlerklassifikation gezeigt, die alle Fehler der automobilen Domäne abdeckt. Dabei erfaßt Klasse 1 alle durch die on-board Diagnose des Steuergeräts entdeckbaren Fehler.

Die Fehler dieser Klasse sind dabei sehr vielseitig und reichen von Fehlern von elektrisch/elektronischen Bauteilen bis hin zu durch Software verursachte Fehler wie Speichermanagement, Deadlocks, Ausnahmen (Exceptions) oder Timer.

Eine Erfassung und Klassifizierung der häufigsten Software-Fehler in Form einer generischen Software-FMEA (vgl. [Rei79, BW01]) böte die Möglichkeit, *generische Detektionsmaßnahmen* in Form von Code-Templates für den Programmcode der zu entwickelnden Funktionen ableiten zu lassen. Das Lernen der häufigsten Fehler könnte dabei durch eine Kopplung an die im vorigen Abschnitt beschriebene automatisierte Analyse der Feedback-Daten geschehen.

Der Nutzen dieser „Bibliothek“ erhöht sich bei der Bereitstellung der selbigen an die Lieferanten, da so einheitliche Fehlersetzbedingungen für die klassifizierten Fehler vorgegeben werden können. Zudem lassen sich bei einer automatischen Generierung dieser Templates sowohl Entwicklungszeit als auch -kosten für die Diagnose reduzieren.

Dieses Thema wird im Ausblick der Arbeit nochmals kurz aufgegriffen.

Generisches Fehlerspeicherkonzept für alle Komponenten. Die Erstellung eines generischen, modularen Fehlerspeicherkonzepts mit dem Anspruch der Erzeugung eines Mehrwerts von Informationen verspricht durch eine präzisere Beschreibung des Fehlers eine schnellere Fehlereingrenzung und somit eine Steigerung der Effizienz der Fehlerbehandlung. Dies betrifft beispielsweise Änderungen sowohl in der Art der Ablage eines Fehlerspeichereintrags als auch im Umfang der gesicherten Informationen, der sich an dem in Abschnitt 4.5 vorgestellten Datenmodell orientiert.

3.10 Verwandte Arbeiten

Viele Probleme der Diagnose, die auch in diesem Kapitel besprochen wurden, sind schon seit langem beschrieben, beispielsweise von [Him78] und [Pau81]. In diesem Abschnitt wird auf weitere Quellen oder ähnliche Ansätze verwiesen.

Verschiedene Datenquellen für die Diagnose und eingeschränkte Wiederverwendung. Ein in den letzten Jahren immer häufiger in der Automobilbranche eingesetzter Ansatz einer einheitlichen Datenbasis für die Informationen des gesamten Fahrzeugs ist der *Produktdatenmanagement* (PDM)-Ansatz [ES04a]. Dabei wird das Produkt über das in der ISO-Norm 10303 definierte *STandard for the Exchange of Product model data* (STEP)-Datenmodell beschrieben [ISO94a]. Das definierte Datenformat ermöglicht einen Austausch von Daten zwischen Zulieferer und OEM. Der Nutzen eines solchen Systems erhöht sich mit der Entwicklung von in Baukästen gesammelten Gleichteilen.

Für mechanische Umfänge werden auch heute schon für die Diagnose relevante Informationen als Teil des PDM erfaßt. So lassen sich beispielsweise Fehlerbeschreibungen mechanischer Teile mit standardisierten Informationen bzw. Aussagen erfassen, die somit leicht an verschiedensprachige Zulieferer oder Werkstätten verteilt werden können. Diese Erfassung hat somit nicht nur Vorteile bei der Erstellung der Diagnose, sondern auch bei der Analyse der Rückmeldungen aus dem Feld, da eindeutige Beschreibungen zurückgemeldet werden können.

Die Entwicklung von Software-Baukästen und der Einbindung der Diagnose dieser Umfänge ist jedoch noch nicht in einem fortgeschrittenen Status (vgl. [TH02]), da bisher keine standardisierten Funktionsarchitekturen für automobile Software definiert sind. Hier könnte die durch AUTOSAR [HSF+04] angestrebte strukturierte Standard-Architektur für Software Abhilfe schaffen (vgl. Abschnitt 8.2).

Generierung on- und off-board Diagnose aus selber Quelle. Im *Arbeitskreis zur Standardisierung von Automatisierungs- und Meßsystemen* (ASAM) wurden zwei Verfahren entwickelt, die eine Generierung von Teilumfängen der Diagnose ermöglichen. Es handelt sich hierbei um *ASAM-ODX* sowie *ASAM-OTX*.

ASAM Open Diagnostic Exchange (ODX) definiert ein offenes, auf XML basierendes Datenformat für den Austausch von Diagnosedaten für das Steuergerät zwischen OEM, Lieferant und den Werkstätten [ASA04, ISO08a]. Durch den strukturierten, formalen Aufbau auf verschiedenen Ebenen hilft ASAM-ODX bei der Beschreibung der für die on-board Diagnose notwendigen Informationen. Dabei kann ODX benutzt werden, um das für die Kommunikation zwischen Fahrzeug und Tester standardmäßig verwendete UDS-Protokoll [ISO06] zu beschreiben.

Aufgrund der formal definierten Struktur lassen sich mittels ODX nicht nur wie erwähnt Daten zwischen den Teilnehmern der Prozeßkette Diagnose austauschen, sondern auch Diagnosedaten generieren. Mit ODX können aber nur Teilumfänge der Diagnose erstellt werden; vielfältige Symptomauswertungen und Inferenzalgorithmen sind aber aufgrund ihrer Komplexität damit bisher nicht möglich. Zudem ist der Haupteinsatzbereich des ODX-Protokolls auch die Umwandlung der ausgetauschten

Protokolldaten in lesbarer Form.

ASAM Open Test sequence exchange (OTX) [ISO09b] basiert ebenfalls auf XML und hat das Ziel, Sequenzen bzw. Handlungsschritte der Diagnose graphisch zu beschreiben und die Sequenzen dann ausführbar zu machen. Durch definierte offene Schnittstellen soll es möglich sein, auch OTX entlang der gesamten Prozeßkette Diagnose und somit sowohl im Tester als auch Steuergerät einzusetzen. Der OTX-Standard befindet sich gerade in Abstimmung.

Eine ausführlichere Beschreibung beider Protokolle findet sich auch in [ZS08].

Weitere Arbeiten, die sich mit der automatischen Generierung der Diagnose aus einer Datenquelle (FMEA) befassen, sind [PBC+02] und [Fra09].

Diagnose verteilter Funktionen. In den Arbeiten von Ermagan und Krüger [EKM+07, EHK+07] wird beschrieben, wie sich der dienstbasierte Architekturansatz für das Fehlermanagement von verteilten automobilen Funktionen nutzen läßt. Teile dieser Dissertation (vgl. Abschnitt 4.5.1) entstanden während eines Forschungsaufenthalts in der veröffentlichenden Forschungsgruppe. Im Ausblick der Arbeit wird der Einsatz der dienstbasierten Architektur für die automobile Diagnose genauer diskutiert.

3.11 Zusammenfassung

In diesem Kapitel wurden Optimierungspotentiale der automobilen Diagnose aufgezeigt. Die Potentiale lassen sich in die Kategorien Kosten und Qualität einordnen. Die dargestellten Potentiale wurden im einzelnen beschrieben und das Vorgehen zur Hebung des Potentials kurz skizziert. Zusätzlich wurde kurz auf den Nutzen für die Kosten und/oder die Qualität der Diagnose eingegangen, der sich aus der Erschließung des Potentials ergibt.

Aufgrund der Vielfältigkeit der Potentiale sowie den beschriebenen Problemen der gegenwärtigen Diagnoseansätze ist ein neuer Ansatz notwendig, um die Potentiale gesamthaft zu erschließen.

Dieser Ansatz wird in Kapitel 4 im Detail dargestellt. Da sich die Potentiale zudem über den gesamten Lebenszyklus des Fahrzeugs erstrecken und sowohl Zulieferer als auch OEM betreffen, wird der vorgestellte Ansatz in einen Prozeß eingebettet, der in Kapitel 5 vorgestellt wird.

Kapitel 4

Effiziente Diagnose

In diesem Kapitel wird die durch die Dissertation eingeführte Methode einer effizienten Diagnose beschrieben. Es handelt sich um einen hierarchischen, holistischen sowie modellbasierten Ansatz mit besonderem Fokus auf verteilten Funktionen, der sich über den gesamten Fahrzeuglebenszyklus erstreckt. Die Einbindung von im Lebenszyklus anfallenden Daten ermöglicht eine kontinuierliche Wartung der Diagnose.

Abschnitt 4.1 bietet eine Übersicht über den gesamten Ansatz. In den darauf folgenden Abschnitten werden die Schritte des Ansatzes einzeln im Detail vorgestellt. Die möglichen Fehler der Fahrzeugfunktionalitäten und ihrer Interaktionen werden mittels einer erweiterten, geschichteten FMEA analysiert und die Maßnahmen mittels einer formalen Abbildung in aussagenlogische Formeln umgewandelt. Somit ergibt sich ein formalisiertes, hierarchisches Datenmodell, dessen Elemente sowohl on-board im Steuergerät als auch off-board im Tester gespeichert werden können. Die Inferenz erfolgt durch Transformation der Monitore in ein boolesches Erfüllungsproblem, das durch einen SAT-Solver gelöst wird. Das Einbinden von Feedback-Daten sorgt für eine ständige Wartung der Diagnosedaten.

Übersicht

4.1 Übersicht

„Das was aus Bestandteilen so zusammengesetzt ist, daß es ein einheitliches Ganzes bildet, ..., das ist offenbar mehr als bloß die Summe seiner Bestandteile“

(Aristoteles - Metaphysik)

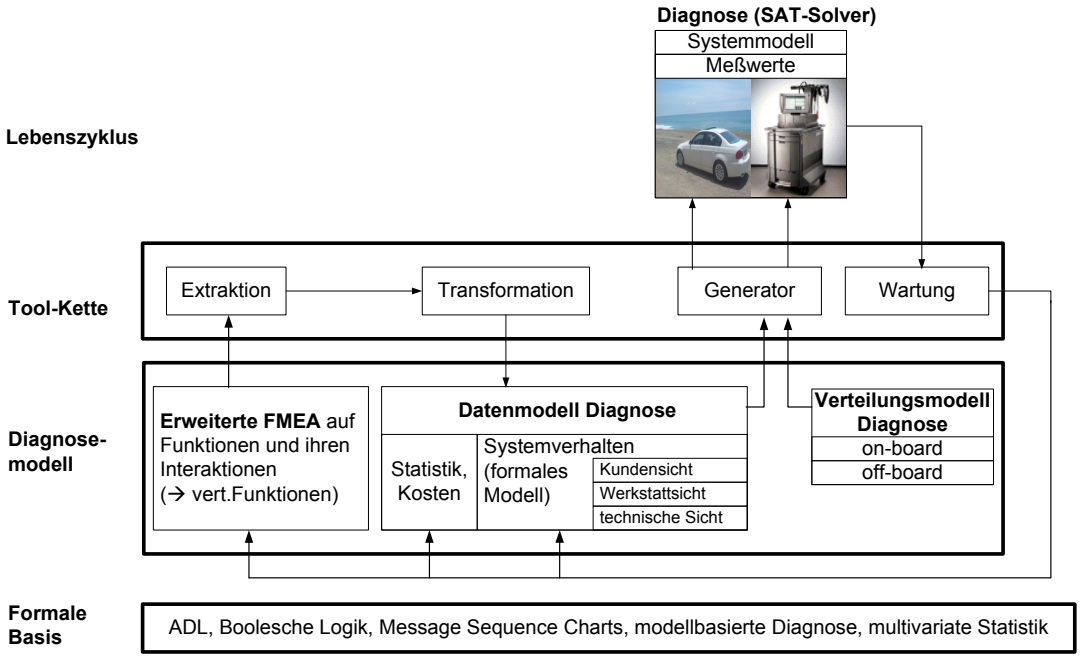


Abbildung 4.1: Übersicht Tool-Kette effizienter Diagnoseprozeß

Die vorgestellte Methodik setzt auf einer Funktionsstruktur und ihren Interaktionen auf. Idealerweise handelt es sich dabei um eine hierarchische Funktionsstruktur wie in Abschnitt 4.2 vorgestellt, um den Wiederverwendungsgedanken zu fördern. Auf diese Struktur wird eine erweiterte FMEA angewandt.

Die relevanten Daten der FMEA-Analyse werden extrahiert. Eine Transformationsvorschrift ermöglicht die Zuordnung der FMEA-Daten in das Diagnosedatenmodell. Die (geschichtete) Funktionsstruktur wird in ein hierarchisches Datenmodell eingeordnet, dessen Ebenen den Rollen jedes Diagnosefalls, *Kunde*, *Werkstatt* sowie *Entwicklung* entsprechen.

Die logischen Formeln der Detektionen und Maßnahmen in der FMEA können *on-* und *off-board* auf Steuergeräte oder das Testsystem der Werkstätten gespeichert werden. Die zu speichernden Funktionen und Maßnahmen werden durch einen Generator in eine für den im weiteren Verlauf der Prozeßkette eingesetzten Satisfiability (SAT)-Solver (vgl. Abschnitt 4.7.1) verständliche Form umgewandelt.

Das on-board im Steuergerät gespeicherte Diagnosemodell prüft zur Laufzeit anhand von im Steuergerät erstellten Messungen sowie bei verteilten Funktionen anhand von Nachrichten, die das Steuergerät von seinen Kommunikationspartnern empfängt, ob ein Fehler vorhanden ist. Falls ja, kann das Steuergerät präventive Maßnahmen einlei-

ten, um ein Versagen zu vermeiden. Im Tester der Werkstätten wird das gespeicherte Systemmodell der off-board Diagnose zur Laufzeit durch vom Tester ausgelesene (Fehler-)Daten aus dem Steuergerät sowie durch Messungen in der Werkstatt erweitert. Diese off-board durchgeführten Messungen beinhalten neben Kundenaussagen über das aufgetretene Verhalten auch zusätzliche vom Mechaniker durchgeführte Tests wie beispielsweise Stromprüfungen. Anhand dieser Daten bestimmt der SAT-Solver mögliche Lösungen. Diese Lösungen stellen die Diagnosen dar und werden zusätzlich über ein (einfaches) Kosten- und/oder Wahrscheinlichkeitsmodell gewichtet.

Um die Diagnose möglichst automatisiert zu warten, werden die Feedback-Daten der Diagnose durch ein multivariates Statistikverfahren auf optimierbare Diagnoseformeln sowie neue, bisher unbekannte Fehlerzusammenhänge untersucht. Optimierbare Formeln sind in diesem Zusammenhang beispielsweise Diagnoseformeln, die nicht funktionierende Reparaturen vorschlagen.

4.2 Hierarchisches Funktionsmodell mit Interaktionen

4.2.1 Hierarchische Funktionsstruktur

Es existieren mehrere in Frage kommende Architekturmodellansätze für die Arbeit. Untersucht wurden Wild et al. (zuerst definiert in [WFH+06], weiterentwickelt in [BFG+08] und [BGK+09]) sowie East-ADL [EAD10].

Allen Ansätzen gemein ist der Aufbau in Schichten. [Bro06] gibt an, daß die Komplexität des Fahrzeugs nach einer Strukturierung in Schichten und Abstraktionsebenen verlangt. Zudem müssen die Architekturmodelle in der Lage sein, mit nur teilweise vorhandenen Informationen umzugehen, da aufgrund der Komplexität und des Umfangs automobiler Systeme ein vollständiges Wissen über Systeme eine Illusion sei [PBKS07]. Zusätzlich erleichtern verschiedene Abstraktionsebenen die Wiederverwendung. Schließlich ermöglichen die Abstraktionsebenen die Darstellung eines Fehlers auf verschiedenen Detailebenen und somit auf Ebenen mit geringerer Komplexität. Dies ist besonders für die Werkstätten aufgrund ihres erwähnten beschränkten Eingriffs in das Steuergerät sowie für Zulieferer, die ihr technisches Wissen durch einen limitierten Einblick in das Steuergerät schützen wollen, nützlich. So kann beispielsweise eine permanente Fehlfunktion der Kundenfunktion „schließe Fenster“ verfeinert werden bis hinunter auf eine fehlerhafte Ausgabe der Software Unit „Analyse Hall Signals“, die wiederum durch ein defektes Bauteil hervorgerufen wurde, wie anhand des Fallbeispiels in Abschnitt 6.2 gezeigt wird.

Allen erwähnten Arbeiten fehlt jedoch die explizite Erfassung fehlerhaften Verhaltens sowie eine Einbindung der für die Diagnose relevanten Handlungsrollen, um alle möglichen Fehler abzudecken (vgl. Abschnitt 2.3.3). Zudem ermöglichen die Rollen eine hierarchische Strukturierung der Diagnose und erleichtern somit die Wiederverwendung. Deshalb wird ein auf diesen Arbeiten aufbauender Ansatz für das Funktionsmodell verwendet und um notwendige Umfänge erweitert (vgl. [KB10]). Aufgrund des Umfangs wird der Aufbau schrittweise dargelegt.

Abbildung 4.2 zeigt die Einordnung der Modellebenen der hierarchischen Funktionsstruktur in die Handlungsrollen der Diagnose. Der Vorteil einer hierarchischen

Struktur liegt in der Reduktion der Komplexität [Sim62].

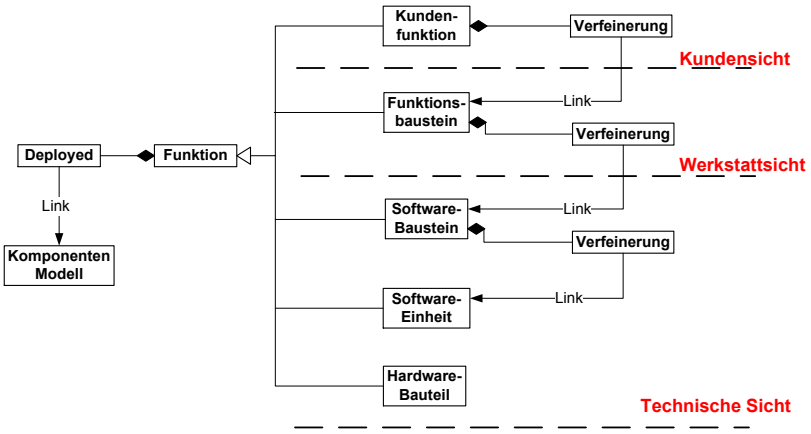


Abbildung 4.2: Einordnung Funktionsstruktur in Rollen der Diagnose.
Quelle: basierend auf [KB10]

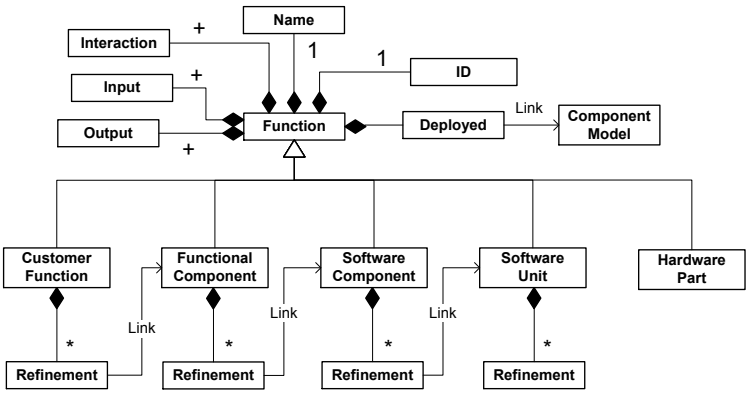


Abbildung 4.3: Hierarchisches Funktionsmodell.
Quelle: basierend auf [AKMP05] und [KB10]

Im Folgenden werden die Attribute des hierarchischen Modells aus Abbildung 4.3 vorgestellt.

Name gibt den Funktionsnamen an. Für eine Abgrenzung der Namen wird vorgeschlagen, Präfixe zu verwenden, die sich aus der Abkürzung des Namens der jeweiligen hierarchischen Ebene ergeben. Für Kundenfunktionen wird das Kürzel KF_ vorgeschlagen, für Funktionsbausteine FB_ sowie für Software-Bausteine SB_ und Software Units SU_.

ID ermöglicht eine eindeutige Identifizierung der Funktion. Eingeführt wird folgender zehnstelliger Aufbau für die ID der Funktion F_i:

$$FC_iAK_iFB_iSB_iSU_i$$

mit den jeweils zweistelligen Ziffern für die genaue Bezeichnung des Funktions-Clusters FC_i, der atomaren Kundenfunktion AK_i, des Funktionsbausteins FB_i, des Software-Bausteins SB_i und der Software Unit SU_i. Die Länge der ID einer Funktion ist somit abhängig von ihrer Zerlegung in feinere Funktionseinheiten.

In-/ und Output geben die Ein- und Ausgaben der Funktion an. Die Eingaben können -je nach verwendetem Detailgrad der Funktion- Kundeneingaben, Werte einer anderen Funktion, Eingaben eines Sollwertgebers bzw. Sensors, empfangene Nachrichten oder Bussignale sein.

Interagierende Komponenten in der Automobildomäne haben sehr häufig nur eine beschränkte Sicht auf die Aktionen der mit ihnen kommunizierenden Komponenten. Dies liegt im wesentlichen an zwei Gründen: in der beschränkten Kommunikation zwischen Steuergeräten aufgrund limitierter Busnachrichtenlänge sowie hohen Zeitrastern der Nachrichtenübertragungen. Beide Punkte verhindern den Austausch ausführlicher Daten. Zum anderen haben vor allem Zulieferer das Interesse, ihr gewonnenes Know-how zu schützen und gewähren deshalb nur beschränkte Informationen über die internen Abläufe ihrer Steuergeräte.

Deshalb muß für die Fehlererkennung und Diagnose ein Ansatz gewählt werden, der in der Lage ist, fehlerhaftes Verhalten der auf den Komponenten gespeicherten Funktionen anhand ihrer meßbaren Ein- und Ausgaben zu erkennen, ohne zwingend genauere Informationen über deren Innenleben zu benötigen. Daraus folgt, daß der Einsatz von Diagnoseansätzen, für die ein explizites internes Systemmodell notwendig ist, erschwert wird, wie bspw. die von Sampath et al. definierte Discrete Event System-Diagnose [SSL+94, SSL+96]. Auf diesen Punkt wurde schon bei der Diskussion der Diagnoseansätze hingewiesen (vgl. Abschnitt 3.7.3).

Interaction bildet die Programmlogik ab. Zur Vermeidung fehlerhaften Verhaltens kann die Diagnose Teil einer Funktion sein und in deren Ablauf eingreifen. Die Interaktionen lassen sich aufteilen in ein- oder zweistellige Operatoren, die sich über Kompositionen verknüpfen lassen, wie Abbildung 4.6 später zeigt.

Deployed verweist auf das **Komponentenmodell**, das angibt, wie und auf welchen Bauteilen die Funktionsbausteine gespeichert werden (vgl. Abschnitt 4.6). Sowohl Kundenfunktionen als auch Funktionsbausteine können auf mehreren Steuergeräten verteilt sein. Die Frage der Verteilung des Diagnosemodells ist für die on-board Diagnose durch Sicherheitsanforderungen vorgeschrieben, für die off-board Diagnose auf den Tester beschränkt. Deployment-Modelle für Komponenten finden sich in [SZ10, Kapitel 2.3] oder [EAD10, Abbildung 8]. Das Mapping des funktionalen Modells auf das Komponentenmodell wird in Abschnitt 4.6.2 dargelegt.

Kundenfunktionen (Customer functions) sind für den Kunden direkt erlebbare Funktionalitäten des Fahrzeugs. Für eine bessere Übersichtlichkeit werden verwandte Kundenfunktionen in sogenannten **Funktions-Clustern** gebündelt. Die Kundenfunktionen werden zerlegt bis zur sogenannten **atomaren Kundenfunktion**, die die kleinste für den Kunden erlebbare Funktion darstellt. Somit sind sowohl Ein- als auch Ausgaben einer (atomaren) Kundenfunktion für den Kunden sicht- und erlebbar. Das Gesamtfahrzeug wird auf der Ebene Kundenfunktion definiert. Verfeinert wird die Kundenfunktion in Funktionsbausteine.

Funktionsbaustein (Functional component) stellt die erste Verfeinerung der Kundenfunktion dar. Auf der Ebene der Funktionsbausteine wird das Bordnetz mit seinem *funktionalen Netzwerk* bzw. *Feature Function Network* definiert. Zudem werden die Funktionsbausteine für das Deployment-Modell in Abschnitt 4.6.2 benötigt, da die Funktionsbausteine auf die Steuergeräte partitioniert werden. Diese Ebene ist für den Kunden nicht mehr einsehbar, jedoch für die Werkstatt und die Entwicklung. Der

Funktionsbaustein läßt sich in die Software-Bausteine zerlegen.

Software-Baustein (Software component) stellt die Verfeinerung des Funktionsbausteins dar. Es handelt sich hierbei um die erste Ebene der technischen Implementierung der Umfänge des Funktionsbausteins und ist somit nur für die Entwicklung einsehbar.

Software-Einheit (Software unit) stellt die tiefste Ebene des hier vorgestellten Modells für Software dar. Diese Ebene wird eingeführt, um den Wiederverwendungsgedanken zu fördern.

Mit der Ebene **Hardware-Bauteil (Hardware part)** wird das funktionale Sollverhalten von Hardware-Teilen erfaßt, die die ihnen zugewiesene(n) Aufgabe(n) ohne Software durchführen. Beispielhaft seien hier ein mechanischer Fensterheberschalter sowie ein Temperatursensor genannt (vgl. Abschnitt 2.1.3), der als Eingangswert eine Temperatur mißt und den gemessenen Wert als elektrische Ausgangsgröße zur Verfügung stellt. Die Erfassung dieses Verhaltens ist notwendig für die Fehlererkennung der Diagnose.

Abbildung 4.4 zeigt anhand des Fensterhebers einen beispielhaften Auszug für die Funktionshierarchie. Zudem zeigt die Abbildung neben dem Bestreben, eine hierarchische Struktur zu schaffen, den Fokus auf den Wiederverwendungsgedanken, da beispielsweise die Funktion *Poll Interrupts* mehrfach verwendet wird. Dabei stellen fettgedruckte Funktionsnamen, wie bspw. *Window Position Detection*, verteilte Funktionen dar.

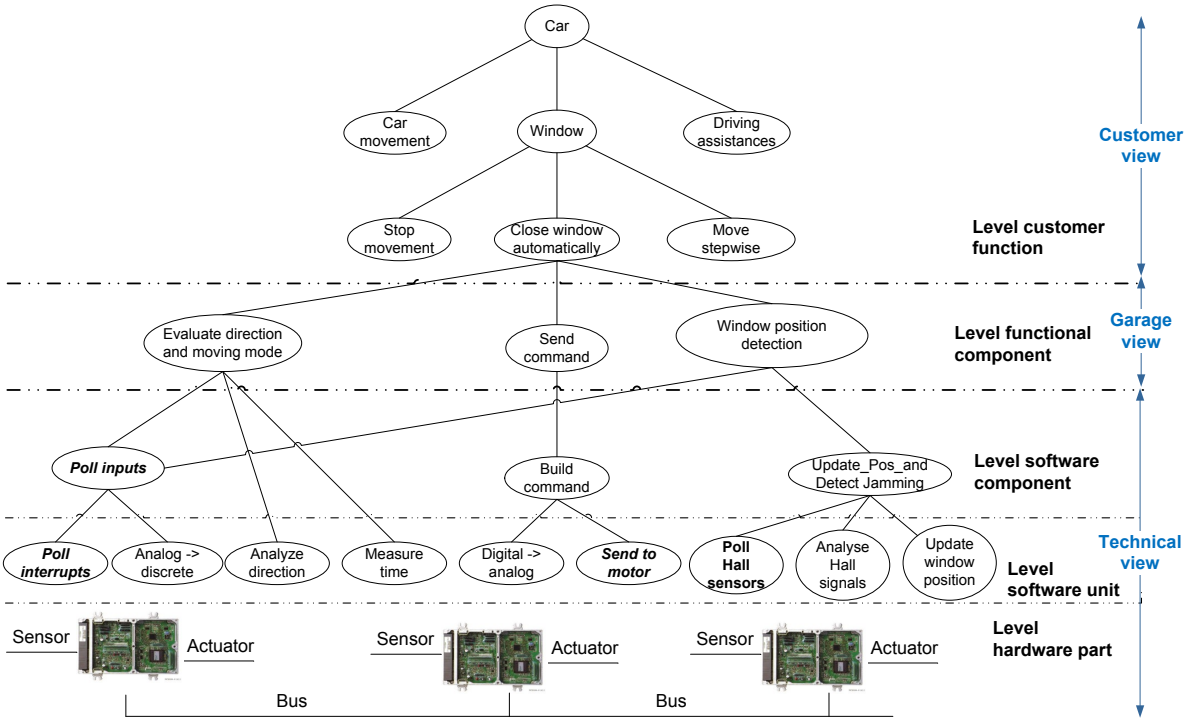


Abbildung 4.4: Beispiel Funktionshierarchie anhand atomarer Kundenfunktion *automatisches Schließen des Fensters*. Quelle: eigene Darstellung.

4.2.2 Interaktionen

In der Einleitung wurde gezeigt, daß autonom handelnde Komponenten seit Anfang der 90er Jahre durch örtlich verteilte, aber zusammen agierende Komponenten ersetzt werden. Deshalb ist die Erfassung der Interaktionen, beginnend mit der Spezifikation des Systems, sehr wichtig. Dies gilt besonders für die Diagnose, da die Transitivität von Funktionen Fehler bewirken kann, die ohne eine strukturierte Erfassung der zusammenhängenden Interaktionen nicht erfaßt werden können.

Es existieren mehrere Verfahren zur Spezifikation von funktionalen Interaktionen. Aufgrund der später durchgeführten erweiterten FMEA auf das Funktionsmodell und seinen Interaktionen sowie der Transformation der FMEA-Daten in aussagenlogische Formeln bestehen für die Wahl der Spezifikationsmethode der Interaktionen Freiheitsgrade.

In dieser Arbeit wird aufgrund seines Bekanntheitsgrades der *Message Sequence Charts* (MSC)-Ansatz beginnend mit der Spezifikation der Interaktionen der Kundenfunktionen verwendet. Krüger stellte in [Krü00] eine Erweiterung der MSC-Notation für die Spezifikation verteilter Systeme dar. Eine genaue Beschreibung der MSC findet sich in [MSC04]. Eine Übersicht über vergleichbare Ansätze wird in Abschnitt 4.9 geboten.

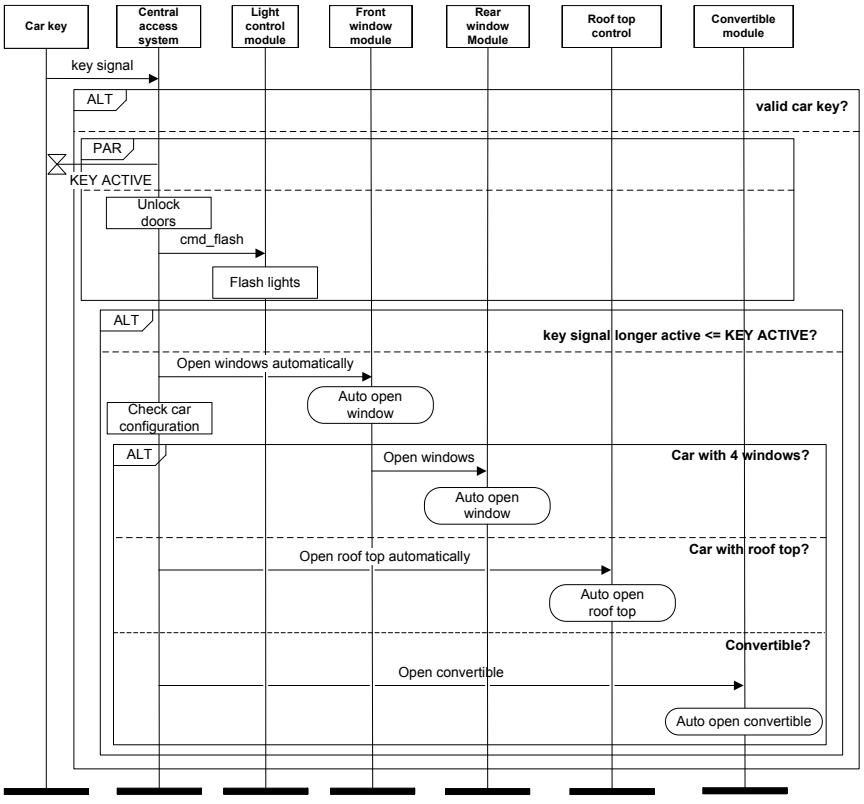


Abbildung 4.5: MSC-Beispiel für Interaktionen beim Fahrzeugöffnen.
Quelle: eigene Darstellung.

Abbildung 4.5 zeigt mit der Beschreibung der Interaktionen des Anwendungsfalls *Fenster und Türen öffnen durch langes Drücken des Fahrzeugschlüssels* ein Beispiel für die Spezifikation mit MSC. Beim Knopfdruck des Fahrzeugschlüssels wird ein Signal

gesendet. Anhand des Signals prüft das *Central Access System* (CAS) die Gültigkeit des Schlüssels. Ist der Schlüssel gültig, werden die vorhandenen Türen des Fahrzeugs geöffnet sowie ein Befehl an das *Lichtmodul* gesendet, daß die Lichter kurz aufgeleuchtet werden sollen, damit der Fahrer sein Auto finden kann. Parallel wird über einen Timer die Zeit gemessen, wie lange das Schlüsselsignal aktiv war. War das Schlüsselsignal länger aktiv als ein festlegbarer Wert *KEY_ACTIVE*, werden die Fenster des Fahrzeugs automatisch geöffnet. Abhängig von der Systemausstattung des Fahrzeugs werden zusätzlich die hinteren Fenster sowie das Cabrio-Dach bzw. das Dachfenster geöffnet.

4.3 Erweiterte FMEA

In den bisher erwähnten Arbeiten für die Funktionsarchitektur sowie Interaktionen wurde nur das Normal- bzw. *Nominalverhalten* einer Funktion untersucht. Für den Kontext der Arbeit muß dieses Verhalten um Fehlverhalten erweitert werden, wie dieses entdeckt werden kann sowie wie die Komponente bzw. aufgrund der vorhandenen Interaktionen das Gesamtsystem darauf reagiert.

In Abschnitt 2.2.1 wurde eine Fehlerterminologie vorgestellt, die nun in die Methodik eingebaut wird. Die Fehlerbegriffe basieren dabei auf [ALRL04] und der vorgestellten Fehlerklassifikation. Grundgedanke ist dabei, alle potentiellen, systematischen Fehler mitsamt ihrer Entdeckungs-, Vermeidungs- sowie Reparaturmaßnahmen zu erfassen. Hierfür wird der Einsatz der bekannten Qualitätsmethodik *Failure Mode and Effects Analysis* (FMEA) vorgeschlagen.

Dieser Abschnitt beginnt mit einer kurzen Darstellung der Geschichte der FMEA und zeigt den Aufbau einer Standard-FMEA. Anschließend erfolgt eine Diskussion über die Einsetzbarkeit der FMEA für die Domäne der Dissertation. Die bisherige FMEA ist für das vorhandene Anwendungsgebiet nicht ausreichend. In diesem Abschnitt wird dargelegt, um welche Anteile die FMEA erweitert werden sollte, um den Nutzen zu erhöhen. Die so erweiterte FMEA wird in die Funktionsstruktur der Arbeit eingeordnet und ist zudem wichtiger Bestandteil des in der Dissertation vorgeschlagenen Prozesses, um mögliche Fehler und ihre Auswirkungen systematisch zu erfassen sowie mit Maßnahmen zu belegen. Abbildung 4.7 zeigt die Erweiterung, wie Fehlverhalten entdeckt wird sowie die Reaktion des System hierauf.

4.3.1 Geschichte der FMEA

Die *Failure Mode and Effects Analysis* (FMEA)-Methode wurde vom amerikanischen Militär entwickelt und zum ersten Mal 1949 als Standard MIL-P-1629 veröffentlicht mit dem Ziel, Versagen militärischer Ausrüstung zu erfassen. Die FMEA wurde vom US-Militär mehrfach aktualisiert, der aktuelle Stand ist [FME80].

Eines der ersten Anwendungsgebiete außerhalb des militärischen Bereichs war der Einsatz im Apollo-Programm (vgl. [NAS66, Pat01]). In der Automobilbranche begann der weitreichende Einzug der FMEA durch Ford Ende der 1970er Jahre, nachdem es zu tödlichen Unfällen aufgrund des Benzintanks des Pinto-Modells kam. Auffahrunfälle konnten zu Schäden am Tank des Pintos führen, wodurch Benzin auslief und es für die Insassen zu tödlichen Bränden führte. Hinzu kam, daß sich Ford der Probleme

bewußt war, das Risiko aber als gering und vernachlässigbar einschätzte (vgl. [BF94]).

Die FMEA wurde in den 80er Jahren auf Bestreben von Chrysler, Ford und General Motors standardisiert als QS-9000 [ISO05a]. Eine erste Standardisierung für Deutschland erfolgte 1980 als [DIN80]. Eine verbesserte Systematik der FMEA, die *System-FMEA* wurde vom VDA 1996 vorgestellt [VDA96] und 2006 aktualisiert als [DIN06].

Neben diesem Standard sind heutzutage auch die von Toyota entwickelte *Design Review Based on Failure Mode* (DRBFM) (vgl. bspw. [SK05]) sowie der von der Automotive Industry Action Group (AIAG) entwickelte Standard [Aut08] gültig. Der AIAG-Standard behandelt dabei die bekannte Tabellen-Darstellung der FMEA (vgl. Tabelle 4.2), während der DRBFM-Ansatz sich auf mögliche Fehlerquellen konzentriert, die durch häufige Änderungen einer Komponente verursacht wurden.

Die meisten bisher genannten Standards wurden vorzugsweise für die Erfassung von Fehlern mechanischer Teile entwickelt. Eine FMEA für Software-Komponenten zu erstellen ist schwieriger und aufwendiger, da zum einen die Fehlerklassifikation durch fehlende Einsicht in das Steuergerät erschwert wird (vgl. Abschnitt 3.9) und zum anderen ein viel größerer zu überprüfender Zustandsraum vorliegt. Als wichtige Arbeiten hinsichtlich Software-FMEA sind hier [Rei79, BW01] zu nennen und mit speziellem Fokus auf eingebettete Systeme [God00]. Eine Literaturübersicht über die FMEA und Software-FMEA findet sich in [HH02].

Im weiteren Verlauf der Arbeit wird sich aber an dem aktuellen FMEA-Standard, definiert in [DIN06], orientiert.

4.3.2 Ablauf der FMEA

Phase	Fragestellung	Ausgabe
Identifiziere	Was kann schiefgehen?	mögliche Versagen mit ihren Ursachen und ihren Effekten
Analysiere	Wie wahrscheinlich ist ein Versagen und was sind die Folgen?	Evaluation RPZ-Zahl: Schwere · Wahrscheinlichkeit
Handle	Was kann getan werden um die Fehlerursache zu eliminieren oder die Schwere zu lindern?	Design- oder Konstruktionsänderungen, Fehlerprüfung, präventive Diagnose, ...

Tabelle 4.1: Drei Phasen der FMEA. Quelle: aus [KFI99]

Tabelle 4.1 zeigt die drei Phasen einer FMEA-Erstellung.

In der ersten Phase wird *identifiziert*, welche Fehlversagen des betrachteten Objekts auftreten können und welche Effekte diese haben können. Das Ergebnis dieser Phase ist eine tabellarische Auflistung der möglichen Fehlversagen des Objekts mitsamt den Ursachen und Effekten dieser Versagen.

In der nächsten Phase wird *analysiert*, wie wahrscheinlich das Auftreten von identifizierten Fehlerquellen ist. Zusätzlich wird die Fehlerfolge im Falle eines Auftretens des Versagens analysiert. Das Ergebnis dieser Phase ist eine Bestimmung der *Risikopriori-*

tätszahl (RPN) für jedes mögliche Versagen und Effekt. Der Aufbau der RPN wird im folgenden Abschnitt genauer dargelegt.

In der dritten Phase werden *Handlungsmaßnahmen* abgeleitet. Ziel der Handlungsmaßnahmen ist, die untersuchten Fehlerursachen zu vermeiden oder die Folgen der Fehlerursachen im Falle eines Eintretens zu lindern. Die bestimmten Handlungsmaßnahmen umfassen dabei Maßnahmen im gesamten Lebenszyklus des Objekts beginnend mit Design- oder Konstruktionsänderungen während der Entwicklung, Überwachungsfunktionen zur Laufzeit oder vorgeschriebene Arten der Entsorgung am Ende des Lebenszyklus.

Die erste Phase zeigt, wie wichtig eine systematische Vorgehensweise bei der FMEA-Erstellung ist, denn für nicht identifizierte Fehler können ja keine Maßnahmen bestimmt werden. Deshalb ist das Ziel einer FMEA-Durchführung die Erfassung aller *systematischen* Fehler einer Komponente.

Ein weiterer wichtiger Aspekt, der jedoch in dieser Arbeit nicht behandelt wird, ist wie eine erstellte FMEA unter die involvierten Personen im Entwicklungsprozeß und Lebenszyklus der untersuchten Komponente bzw. System verbreitet werden sollte.

4.3.3 Aufbau der FMEA

In diesem Abschnitt werden die wichtigsten FMEA-Begriffe vorgestellt. Für eine ausführlichere Darstellung sei auf [Aut08] sowie [VDA96] verwiesen.

System:									
Funktion:									
Zeilennr.	Fehlerursache	Fehlerart	Fehlerfolge	Schwere	Vermeidungs-maßnahme	Auftretenswahr-scheinlichkeit	Entdeckungs-maßnahme	Entdeckungswahr-scheinlichkeit	RPZ

Tabelle 4.2: Tabellarischer Aufbau der FMEA.Quelle: basierend auf [DIN80]

System: Name des betrachteten Systems.

Funktion: Name der betrachteten Funktion.

Fehlerursache/ Fault cause: die Ursache der Fehlfunktion bzw. des Versagens.

Fehlerart/Failure mode: mögliche Fehlfunktion bzw. Versagen des Systems.

Fehlerfolge/ Failure effect: mögliche Folgen des Fehlers auf System und Umwelt.

Bedeutung des Fehlers (B)/ Severity (S): bewertet die Bedeutung der Fehlerfolge auf das Gesamtsystem und den Kunden aus Sicht des FMEA-Erstellers mit einer Zahl von 1 bis 10. Dabei hat 10 hochgefährliche Auswirkungen für den Kunden und 1 geringe Auswirkungen.

Vermeidungsmaßnahme/ Occurence: beschreibt eine Maßnahme zur Vermeidung des Fehlers. Dabei kann es sich zum einen um eine konstruktive Maßnahme während der Entwicklung handeln, durch die der Fehler nicht auftreten kann. Andererseits kann dies aber auch Maßnahmen betreffen, die das System zur Laufzeit ergreifen kann, um den Fehler zu vermeiden. Beispiele hierfür sind das Prüfen eines Divisors auf einen Wert ungleich 0 vor einer Division oder das Versetzen des Fahrzeugs in einen Notlaufmodus.

Auftretenswahrscheinlichkeit (A)/ Occurence rating (O): bewertet die angenommene Wahrscheinlichkeit des Auftretens des Fehlers unter Berücksichtigung der Vermeidungsmaßnahmen. Dabei steht 1 für eine geringe Auftretenswahrscheinlichkeit und 10 für eine hohe.

Entdeckungsmaßnahme/ Detektion: beschreibt Maßnahmen, um einen Fehler zu entdecken. Damit ist beispielsweise die Fehlererkennung der Diagnose im Steuergerät gemeint.

Entdeckungswahrscheinlichkeit (E)/ Detection rating (D): bewertet die Wahrscheinlichkeit mit der der betreffende Fehler erkannt werden kann mit einer Zahl von 1 bis 10. Dabei bezeichnet 10 eine geringe Erkennungswahrscheinlichkeit und 1 eine hohe.

Risikoprioritätszahl (RPZ)/ Risk priority number (RPN): die Risikoprioritätszahl ist ein Wert, um einen Fehler besser einschätzen zu können, indem ein Produkt aus der Schwere eines Fehlers, der Auftretenswahrscheinlichkeit der Fehlerursache und der Entdeckungswahrscheinlichkeit des Fehlers gebildet wird. Der Wert RPZ entsteht aus folgender Formel:

$$RPZ = A \cdot S \cdot E$$

4.3.4 Diskussion der Wahl der FMEA als Datenquelle

Nachteile der FMEA

FMEA oft nicht vollständig, da diagnoserelevante Daten nicht erfaßt werden. Wie in Kapitel 3 erwähnt, werden aufgrund des verbesserbaren Informations-Workflows in der Entwicklung sowie dem Bestreben des Zulieferers, wettbewerbsrelevantes Wissen für sich zu behalten, viele wichtige Daten über Fehler und Gegenmaßnahmen nicht erfaßt. Zusätzlich wird eine systematische Analyse des Systems durch die Tatsache erschwert, daß meist weder das Normalverhalten und noch seltener das Fehlverhalten von Funktionen formal spezifiziert werden (vgl. Abschnitt 3.4).

Das führt dazu, daß die systematische Analyse von Fehlern wie sie in der FMEA praktiziert wird, nicht im vollen Ausmaß erfolgen kann. Dieses Problem betrifft aber nicht nur die FMEA, sondern auch andere Qualitätsmethodiken.

Risikoprioritätszahl. Um mögliche Fehler und ihre Auswirkungen vergleichen zu können, ermöglicht die FMEA eine Bewertung von Risiken mittels der *Risikoprioritätszahl* (RPZ). Untersucht wird dabei die *Auftretenswahrscheinlichkeit* (A) einer Fehlerursache, die *Entdeckungswahrscheinlichkeit* (E) den Fehler zu entdecken sowie die *Bedeutung/Schwere* (S) der Fehlerfolge. Die RPZ berechnet sich dann durch Multi-

plikation der erwähnten Faktoren:

$$RPZ = (1 \leq A \cdot S \cdot E \leq 1000)$$

[BR96, KFI99, BK02] merken an, daß diese Bewertung, die auch zur Priorisierung von Fehlermaßnahmen eingesetzt wird, nur eine subjektive Schätzung, nicht aber eine präzise Aussage darstellt. So ergibt sich für die RPZ eines Fehlers F_1 mit $RPZ_{F_1} = (A = 8, S = 8, E = 3) = 192$ der gleiche Wert wie für den eines Fehlers F_2 mit $RPZ_{F_2} = (A = 8, S = 4, E = 6)$, obwohl der Fehler F_1 aufgrund einer höher bewerteten Schwere als potentiell gefährlicher betrachtet werden sollte.

Zusätzlich sei angemerkt, daß die Definition der einzelnen Faktoren je nach FMEA-Standard variieren kann (vgl. [KFI99]).

In der Dissertation werden die Zahlenwerte der Faktoren Entdeckungs- und Auftretenswahrscheinlichkeit durch statistische Daten aus dem Lebenszyklus bestimmt.

Komponentenzentrierte Sichtweise. Eine FMEA wird meist am Anfang des Entwicklungsprozesses durchgeführt. Aufgrund der frühen Phase sind zu diesem Zeitpunkt viele technische Details der späteren Komponente sowie des Gesamtsystems noch offen.

Wird eine FMEA gemäß AIAG-Standard [Aut08] durchgeführt, so werden für Fehler die Daten mittels einer komponentenzentrierten Sicht erfaßt. Die Sicht ist deshalb komponentenzentriert, da weder das Zusammenwirken der Teilsysteme noch die Auswirkungen des Fehlers auf das Gesamtsystem in vollem Maße untersucht werden können.

Ein weiterer Grund für eine oft vorwiegend komponentenzentrierte FMEA ist in der Vergabe der Entwicklung einer Komponente an den Zulieferer zu sehen. Die OEM verpflichten den Zulieferer zur Erstellung einer FMEA im Entwicklungsprozeß der Komponente. Aufgrund des fehlenden Wissens des Zulieferers über die vom OEM zur Verfügung gestellten Schnittstellen zum Gesamtsystem hinaus ist diese FMEA nur eingeschränkt aussagefähig über Auswirkungen von fehlerhaften Verhalten der betrachteten Komponente auf das Gesamtsystem Fahrzeug.

Eine System-FMEA [VDA96] versucht hingegen ein auch komponentenübergreifendes Gesamtsystem zu erfassen und die Auswirkungen von Fehlern auf das Gesamtsystem zu analysieren. Zusätzlich wird bei der System-FMEA untersucht, inwieweit Fehler voneinander abhängig sind. Die Wirksamkeit der System-FMEA und ihrer Erfassung des Gesamtsystems wird aber eingeschränkt durch das technische Verständnis des/r FMEA-Ersteller des Gesamtsystems sowie vom Vorhandensein einer komponentenübergreifenden Spezifikation.

FMEA wird nicht begleitend im Entwicklungsprozeß durchgeführt. [KFI99] verweist auf Bednarz und Marriot, die darlegen, daß die FMEA nicht begleitend im Entwicklungsprozeß durchgeführt werde, sondern einmalig zu spät im Entwicklungsprozeß [BM88]. Ein Ergebnis der späten Durchführung sei dann, daß die im Verlauf der FMEA-Erstellung entdeckten Fehler sich nicht im vollen Maße auf Design-Entscheidungen auswirken können [McK91].

Wird hingegen die FMEA zu früh im Entwicklungsprozeß durchgeführt, so sind wie oben aufgeführt viele Details des Systems, seiner Teilsysteme und der Interaktionen unter Vorbehalt zu betrachten, da sie im Laufe der Entwicklung noch Änderungen unterworfen sein können.

Für einen idealen Nutzen der FMEA-Methodik sollte die FMEA also den ganzen Entwicklungsprozeß begleiten, wie es die DRBFM (vgl. [SK05]) vorsieht.

FMEA-Erstellung zeitaufwendiger und teurer Prozeß. Eine kontinuierliche Begleitung des Entwicklungsprozeß durch die FMEA ist aber unter Kostengesichtspunkten als kritisch zu betrachten. Eine FMEA wird von vielen Experten in einem ermüdenden und zeitaufwendigen Prozeß [OHL91] erstellt. Die Experten setzen sich dabei aus Mitarbeitern aus den Einkaufs-, Entwicklungs-, Logistik- und Qualitätsabteilungen sowie einem Moderator zusammen. Der Zeitaufwand ist abhängig von der Komplexität des zu untersuchenden Systems.

Dieser Aufwand ist um so kritischer zu sehen, da die Daten der FMEA nur eingeschränkt wiederverwendbar sind. Dies liegt zum einen an der fehlenden hierarchischen Struktur einer FMEA gemäß [Aut08] und zum anderen daran, daß die zu untersuchenden Systeme, vor allem Software-Systeme, selten modular oder hierarchisch aufgebaut sind.

In der Dissertation wird ein Ansatz dargelegt, bei dem die Daten der FMEA möglichst umfassend wiederverwendbar sind.

FMEA enthält keine Lebenszyklusdaten und wird nicht gewartet. Es wurde weiter oben angemerkt, daß die FMEA eingeschränkt wiederverwendbar ist. Ein weiterer wichtiger Grund hierfür ist, daß die FMEA nur in der Entwicklungsphase durchgeführt und somit erst im Lebenszyklus auftretende bzw. sichtbare Fehler nicht erfaßt werden. Dies betrifft auch erfaßte Daten in der FMEA, die sich als nicht korrekt im Laufe des Lebenszyklus herausstellen.

Die mangelnde Wartung der FMEA-Daten betrifft auch insbesondere die Faktoren Auftretens- und Entdeckungswahrscheinlichkeit der RPZ. [Gil93] und [KI00] schlagen vor, die geschätzten Wahrscheinlichkeiten eines Fehlers E und A durch Kostendaten sowie durch statistische Daten aus dem Lebenszyklus zu ersetzen.

Der Vorteil dieses Ansatzes liegt darin, daß zum einen die geschätzten Werte durch stets aktuell berechnete relative Häufigkeiten ersetzt werden und zum anderen, daß durch das zusätzliche Einbauen von Kosten sich die Maßnahmen besser vergleichen lassen.

In der Dissertation werden die vorhandenden Diagnose- und Fehlerdaten kontinuierlich durch Lebenszyklusdaten gewartet. Die Wartung betrifft dabei die Aktualisierung der Auftretens- und Entdeckungswahrscheinlichkeiten der RPZ, aber zudem auch die Entdeckung fehlerhafter Zusammenhänge in der FMEA und den Diagnosefunktionen.

FMEA beinhaltet keine Kosten und insbesondere keine Lebenszykluskosten. [KI00, SR03] bemängeln zudem, daß die in der FMEA erstellten Maßnahmen nicht mit Kosten versehen werden. Dieser Nachteil vergrößere sich dadurch, daß nicht die

im Lebenszyklus entstehenden Kosten gesamthaft berücksichtigt werden.
Als Lösungsansatz schlagen sie vor, die einzelnen Kosten eines Fehlers in allen Phasen des Lebenszyklus mitsamt den möglichen Wahrscheinlichkeiten in den entsprechenden Phasen zu betrachten und zu summieren.

Vorteile der FMEA

FMEA ist bekannter, leichtverständlicher Ansatz. Die FMEA wurde zum ersten Mal 1949 veröffentlicht und wird seit den 70er Jahren sehr häufig in den verschiedensten Branchen wie Luft- und Raumfahrt, Automobil oder weiteren sicherheitsrelevanten Gebieten eingesetzt. Dies führt dazu, daß die FMEA in den meisten Unternehmen schon bekannt und akzeptiert ist, sodaß nur geringer Schulungsaufwand notwendig ist.

Systematischer Ansatz. Die FMEA stellt einen systematischen Ansatz zur Erfassung und Analyse von Fehlern dar. So befaßt sich ein großer Teil der vom VDA normierten FMEA [VDA96] mit einer systematischen Vorgehensweise bei der FMEA-Erstellung. Dies soll sicherstellen, daß keine Fehler übersehen werden und basiert auf Untersuchungen, die feststellten, daß bei einer systematischen Fehlersuche und -analyse deutlich mehr Fehler gefunden werden können (vgl. [KFI99, Tabelle 5]), da ohne einen systematischen Ansatz Ingenieure eine subjektive Analyse erstellen, die von ihrem Erfahrungslevel abhinge [BCJS92].

Den systematischen Ansatz der FMEA erkennen auch Sicherheitsnormen wie die ISO 26262 an, die vorschreiben, daß induktive Analysen zur Vermeidung systematischer Fehler durchgeführt werden müssen [ISO09a, Teil 4 Kap. 7.4.3.1].

Strukturiertes Datenformat. Durch die tabellarische Form und die Verknüpfung der FMEA-Elemente untereinander entsteht ein strukturiertes Datenformat. Ein systematisches Datenmodell gewinnt zusätzlichen Wert bei einer Analyse von Fehlern eines hierarchischen Modells. Somit entsteht ein geschichtetes Fehlermodell, das die Wiederverwendbarkeit der diagnoserelevanten Daten deutlich erhöht.

Weiterhin ermöglicht ein strukturiertes Datenformat eine automatisierte Erstellung und Verarbeitung. Dies ist um so positiver einzuschätzen, als daß gerade im Diagnoseumfeld viele Zulieferer darauf bedacht sind, ihr wettbewerbsrelevantes Wissen zu schützen. Die in der Dissertation vorgestellte Methodik ermöglicht durch ein definiertes Datenmodell die Extraktion der relevanten Daten aus einer FMEA eines Zulieferers bei gleichzeitiger Wahrung seines Wissens (vgl. Abschnitt 4.5.4).

FMEA-Durchführung notwendig bei sicherheitsrelevanten Systemen. Die Normen IEC 61508 und ISO 26262 verlangen bei der Erstellung von sicherheitsrelevanten Funktionen eine Analyse der Auswirkungen von Fehlfunktionen.

Diese Analyse wird bei der ISO 26262 *Hazard Analysis and risk assesment* genannt [ISO09a, Teil 3 Kapitel 7]. Für jede Fehlerfunktion wird ein zu erreichendes Sicherheitsziel (safety-goal) definiert, das in Verbindung mit dem zu erreichenden ASIL-Level der Komponente steht. Die Hazard Analysis setzt sich dabei aus der Analyse der

Faktoren *severity*, *exposure* und *controllability* zusammen. Dabei bezeichnet *severity* das Ausmaß des Schadens auf Personen, *controllability* das Vermeiden des spezifizierten Schadens durch eine rechtzeitige Reaktion einer involvierten Person sowie *exposure* die Häufigkeit der zeitgleichen Überschneidung der betrachteten Fehlfunktion mit einem Betriebszustand (Definitionen entnommen aus [ISO09a, Teil I]). Die Definition der Faktoren der Hazard-Analyse zeigt eine große Ähnlichkeit mit der RPZ der FMEA.

Zudem wird in mehreren Abschnitten der ISO 26262 vorgeschrieben, Methodiken durchzuführen, um systematisch mögliche Fehler zu entdecken und zu bewerten. Dabei wird an diesen Stellen explizit die FMEA genannt (vgl. [ISO09a, Teil 3 Kap. 7.4.4.3 und Kap. 8.4.2.3, Teil 4 Kap. 7.4.3.1, Teil 5 Kap. 7.4.3.1 und Kap. 7.4.3.3, Teil 9 Kap. 7.41]) und daß die Ergebnisse der Methodiken dokumentiert werden müssen (vgl. [ISO09a, Teil 9 Kap. 8.4.8]).

4.3.5 Begründung der Wahl der FMEA als Datenmodell der Arbeit

Obwohl, wie Abschnitt 4.3.4 zeigt, einige gewichtige Nachteile gegen die FMEA bestehen, spricht der strukturierte Ansatz und vor allem die Akzeptanz der FMEA im automobilen Umfeld für einen Einsatz. Zudem treffen viele der erwähnten Nachteile auch auf andere Qualitätsmethodiken zu, wie in Kapitel 3 gezeigt wurde.

Ein wichtiger Vorteil der FMEA ist, daß sie bei der Erstellung sicherheitsrelevanter Komponenten durch die ISO 26262 verpflichtender Teil des Entwicklungsprozesses wird. Ziel ist, daß durch den in der Dissertation vorgestellten Ansatz die besprochenen negativen Punkte vermieden werden.

4.3.6 Erweiterung der FMEA

Die FMEA wird um mehrere Elemente erweitert, um so den Anforderungen der Domäne gerecht zu werden, und als Datenmodell konzipiert. Eine Darstellung des Datenmodells sowie Erklärung der Erweiterungen findet sich in Abschnitt 4.5.2. So werden beispielsweise die FMEA-Vermeidungsmaßnahmen in Vermeidungsmaßnahmen unterteilt, die on-board im Steuergerät (Mitigator) sowie off-board in den Werkstätten (Remedy) angewendet werden können. Zusätzlich werden die subjektiven Einschätzungen der Auftretenshäufigkeiten der RPZ-Faktoren durch statistische Daten erhöht. Schließlich wird die FMEA zudem mit einem formalen Fehlermodell verknüpft.

Durch die Hinzunahme der neuen Elemente ändert sich auch der Ablauf der FMEA-Erstellung. Tabelle 4.3 zeigt die Vorgehensweise für die erweiterte FMEA. Dabei werden die Erweiterungen der gegenwärtigen FMEA in kursiver Schrift dargestellt.

4.3.7 Fallbeispiel erweiterte FMEA

Beispiele für die erweiterte FMEA finden sich im Kapitel 6 sowie in Tabelle 6.7.

Phase	Fragestellung	Ausgabe
Identifiziere	Was kann schiefgehen? *Übernahme von FMEA- oder Diagnosedaten möglich?*	mögliche Versagen mit ihren Ursachen und ihren Effekten *ggf. von Vorgänger- oder ähnlicher Komponente*
Analysiere	Wie wahrscheinlich ist ein Versagen und was sind die Folgen? *Wie sind die Fehlerfolgen für Kunde/ Werkstatt/ Entwicklung beobachtbar (siehe Abschnitt 4.4)?*	*Initialschätzung RPZ-Faktoren und statistische Wartung, Fehlereffekte auf den 3 Ebenen, Klassifikation Fehler und Fehlerursache*
Handle	*Wie kann der Fehler auf den einzelnen Ebenen entdeckt werden?* Was kann getan werden, um die Fehlerursache zu eliminieren oder die Schwere zu lindern? *Was kann hierzu im Steuergerät erfolgen? Was off-board?* *Kosten der Maßnahme(n)?*	*Fehlerdetektoren auf den einzelnen Ebenen,* *on- und off-board Maßnahmen zur Fehlervermeidung,* *Kostenbewertung Maßnahmen*

Tabelle 4.3: Vorgehensweise erweiterte FMEA, basierend auf Tabelle 4.1

4.4 Formales Fehlermodell

Im vorigen Abschnitt wurde die Wahl der eingesetzten Qualitätsmethode begründet. Die FMEA wird durchgeführt, um mögliche systematische Fehler einer Komponente sowie die Auswirkungen der Fehler auf das Gesamtsystem und seine Umgebung zu analysieren. Die in diesem Abschnitt verwendeten Fehlerbegriffe basieren dabei auf den von Avižienis et al. definierten Begriffen (vgl. bspw. [ALRL04]).

In diesem Abschnitt wird vorgestellt, wie aus einer FMEA das verwendete, formale Fehlermodell der Arbeit entsteht. Basis für diese Umwandlung ist das in [KB10, Kap. 4] vorgestellte Mapping von FMEA-Elementen auf das Schichtenmodell.

Für den Fokus der Arbeit, verteilte Funktionen, ist besonders die Untersuchung von Fehlern von Interaktionen wichtig. Diese Untersuchung ist jedoch aufgrund eines möglicherweise beschränkten Einblicks in das Steuergerät (Black Box-Sicht) problematisch. Dieses Problem wird durch die Durchführung der FMEA auf den Funktionen und ihren Interaktionen auf verschiedenen Abstraktionsebenen angegangen. Sowohl die Funktionen und ihre Interaktionen als auch die Effekte möglicher Fehler werden in das vorgestellte Ebenenmodell eingepaßt.

Für jeden Fehler wird analysiert, wie er durch die in Abschnitt 4.2 vorgestellten Rollen *Kunde*, *Werkstatt*, *Entwicklung* entdeckt werden kann. Für die Rollen Werkstatt und Entwicklung werden Maßnahmen hinzugefügt. Dabei handelt es sich im Falle der Entwicklung um Maßnahmen, wie die Entstehung eines Fehler bzw. der Zustandsübergang in einen fehlerhaften Zustand verhindert werden kann (präventive Diagnose) und im Falle der Werkstatt um Reparaturmaßnahmen, mit denen die Kom-

ponente bzw. das System wiederhergestellt werden können. Im Gegensatz hierzu ist es dem Kunden meist nicht möglich, ein Versagen zu vermeiden oder zu reparieren. Es gibt natürlich Kunden, die über vertieftes Fahrzeugwissen verfügen, dennoch wird die obige Annahme zur besseren Abgrenzung der Schichten getroffen.

Schon in der Einleitung wurde die Komplexität der Diagnose als eines der größten Probleme dargestellt. Indem im vorgestellten Ansatz die Diagnose auf verschiedenen Abstraktionsebenen durchgeführt wird, kann die Diagnose auch auf einer Ebene mit vergleichsweise geringerer Komplexität durchgeführt werden.

Die Suche nach den Fehlern startet auf der Kundenebene. Es werden dabei die Fehler gesucht, die zu Versagen mit beobachtbaren Effekten führen, die dem beobachtbaren Verhalten entsprechen. Fehler müssen nur dann auf einer technisch tieferen Ebene unterschieden werden, falls ihre zugeordneten Reparaturmaßnahmen auf verschiedene Teile (Hard- oder Software) zugreifen.

Dieses Vorgehen basiert auf der Tatsache, daß aufgrund der limitierten Einflußmöglichkeiten der Werkstatt die Reparatur des Fahrzeugs durch Reparatur sogenannter austauschbarer Einheiten erfolgt (vgl. Abschnitt 2.3.2). Fehler mit gleichen beobachtbaren Effekten, die zudem derselben austauschbaren Einheit zuweisbar sind, müssen deshalb nicht durch die Diagnose mittels zusätzlicher Beobachtungen diskriminierbar sein. Diese Annahme gilt **explizit nicht** für sicherheitsrelevante Systeme. Zusätzlich ist der limitierte Einfluß der Werkstatt in Widerspruch zu den Interessen der Entwicklungsabteilung, für die es aufgrund der Serienbetreuung der Komponente bzw. des Systems von großem Interesse ist, möglichst genau auftretende Probleme zu erfassen. Die Frage nach der Genauigkeit der Diagnose wird im Ausblick der Arbeit in Abschnitt 8.2 näher diskutiert.

Durch die zusätzliche Unterscheidung von Fehlern nur im Falle verschiedener austauschbarer Einheiten wird zum einen angestrebt, die Anzahl der Beobachtungen und somit Komplexität und Kosten der Meßpunkte möglichst gering zu halten, und zum anderen -im Falle der Hinzunahme detaillierterer Beobachtungen- kostspielige „Trial-and-Error“-Reparaturen zu vermeiden.

4.4.1 Aussagenlogik und boolesche Logik

Definition: Syntax der Aussagenlogik (aus [Sch00, S. 14])
Eine *atomare Formel* hat die Form A_i (mit $i = 1, 2, 3, ...$).
Formeln werden durch folgenden induktiven Prozeß definiert:

1. Alle atomaren Formeln sind Formeln
2. Für alle Formeln F und G sind $(F \wedge G)$ und $(F \vee G)$ Formeln.
3. Für jede Formel F ist $\neg F$ eine Formel.

Eine Formel $\neg F$ heißt *Negation* von F, $(F \wedge G)$ heißt *Konjunktion* von F und G, $(F \vee G)$ heißt *Disjunktion* von F und G. Eine Formel F, die als Teil einer Formel G auftritt, heißt *Teilformel* von G.

Weiterhin lassen sich folgende Operatoren definieren, die auf den bisher definierten Operatoren aufbauen:

$$(A_1 \rightarrow A_2) \equiv (\neg A_1 \vee A_2)$$
$$(A_1 \leftrightarrow A_2) \equiv ((A_1 \wedge A_2) \vee (\neg A_1 \wedge \neg A_2))$$
$$(\bigvee_{i=1}^{n} A_i) \equiv (A_1 \vee .... \vee A_n)$$
$$(\bigwedge_{i=1}^{n} A_i) \equiv (A_1 \wedge .... \wedge A_n)$$

Definition: Semantik der Aussagenlogik (aus [Sch00, S. 15])
Die Elemente der Menge $\{0, 1\}$ heißen *Wahrheitswerte*. Eine *Belegung* ist eine Funktion $\mathcal{A} : D \rightarrow \{0, 1\}$ wobei D eine Teilmenge der atomaren Formeln ist.

Durch die Funktion $\mathcal{A}$ werden den Variablen und Formeln die Werte 0 und 1 zugewiesen, wobei im weiteren Verlauf dieser Arbeit der Wert 1 dem Wahrheitswert *wahr* und 0 dem Wahrheitswert *falsch* entspricht.

Definition: Erfüllbarkeit der Aussagenlogik (aus [Sch00, S. 18])
Sei F eine Formel und $\mathcal{A}$ eine Belegung. Ergibt sich für die Belegung der Variablen in der Formel F $\mathcal{A}(F) = 1$, also der Wahrheitswert *wahr*, so ist $\mathcal{A}$ *ein Modell für* F.

Eine Formel ist genau dann *erfüllbar*, falls mindestens eine Belegung $\mathcal{A}$ ein Modell für F darstellt. Stellt keine Belegung $\mathcal{A}$ für F ein Modell dar, so ist F *unerfüllbar*. Eine Formel ist genau dann eine *Tautologie*, falls jede Belegung von $\mathcal{A}$ ein Modell darstellt.

4.4.2 Einordnung FMEA in Schichtenmodell

In diesem Abschnitt wird dargelegt, wie die FMEA-Elemente auf logische Variablen im Schichtenmodell abgebildet werden (aus [KB10, Kapitel 4]). Dadurch ergibt sich für jede Ebene eine logische Formelmenge. Zum besseren Verständnis werden die FMEA-Elemente in fetter Schrift markiert und anhand des späteren Fallbeispiels Fensterheber (vgl. Abschnitt 6.1) aufgezeigt.

Die Erstellung dieses Gesamtmodells ist anfänglich ein manueller Prozeß. Hierbei ist zusätzlich zu beachten, daß aufgrund von unterschiedlichen Vorgehensweisen der FMEA-Erstellung es sich bei den Inhalten der erforderlichen Elemente um textuelle, informelle Beschreibungen handeln kann. Langfristig gesehen ermöglicht jedoch das Schichtenmodell die Erstellung wiederverwendbarer Elemente (meist auf den oberen beiden Ebenen).

System: Name der betrachteten Komponente, beispielsweise Fensterheber-ECU.

Funktion: Name der betrachteten Funktion. Hierbei kann es sich je nach Ebene des Schichtenmodells (vgl. Abschnitt 4.2) um eine *Kundenfunktion*, einen *Funktionsbaustein*, einen *Software-Baustein*, eine *Software-Einheit* oder ein *Hardware-Bauteil* handeln. Für das Fallbeispiel Fensterheber aus Abbildung 4.4 könnte dies entweder die Kundenfunktion *Fenster automatisch schließen*, der Funktionsbaustein *Bestimme Fensterposition*, der Software-Baustein *Entdecke Einklemmen*, die Software-Einheit *Werte Hall-Effekt-Sensoren aus* oder ein *Hall Sensor* sein.

Fehlermodus: Klassifikation eines Fehlers (vgl. Abschnitt 2.3.3) oder eine detaillierte Beschreibung des Fehlers, beispielsweise *thermischer Schaden des Fensterhebermotors*. Die Klassifikation wird später in Abschnitt 4.5.2 im Detail vorgestellt.

Fehlerursache: die Ursache des Fehlers, die zu einem Fehlereffekt führt. Die Ursache wird mit folgenden logischen Formeln beschrieben:

$$\begin{array}{lll} (F_i & \rightarrow & B_i) \quad \bigwedge \\ (F_i & \rightarrow & FE_i) \end{array}$$

F_i, FE_i und B_i sind logische Variablen. F_i wird dabei der Wahrheitswert *wahr* zugewiesen, falls der mit F_i in Verbindung stehende Fehler als vorhanden angenommen wird. B_i wird der Wert *wahr* zugewiesen, falls das mit B_i in Verbindung stehende Verhalten beobachtet werden kann.

Eine Fehlerursache führt zu einem Fehlereffekt, der mit der logischen Variable FE_i korreliert, welche den Wahrheitswert *wahr* zugeordnet bekommt, falls der mit FE_i in Verbindung stehende Fehlereffekt beobachtet werden konnte.

Fehlereffekt: Effekt, den der Fehler auf die Funktion, Komponente, Gesamtsystem oder Umwelt hat. Die Beziehung zwischen Fehlerursache und Effekt läßt sich wie folgt darstellen:

$$F_i \rightarrow FE_i$$

F_i ist dabei die logische Variable für den Fehler und FE_i eine logische Variable, welche den Wahrheitswert wahr zugewiesen bekommt, falls der mit FE_i in Verbindung stehende Fehlereffekt beobachtet werden konnte. Wie erwähnt läßt sich der Effekt eines Fehlers in den drei vorgestellten Ebenen feststellen, weshalb im weiteren Verlauf die Variable FE_i nicht mehr verwendet werden muß.

$$\begin{array}{lll} (FE_i & \rightarrow CC_i) & \bigwedge \\ (FE_i & \rightarrow G_i) & \bigwedge \\ (FE_i & \rightarrow T_i) & \end{array}$$

Dieses beobachtbare Verhalten kann eine Beschwerde des Kunden in der Werkstatt CC_i, ein für die Werkstatt meßbares (Fehl-)verhalten G_i, oder ein im Steuergerät durch Messungen T_i entdeckbares (und später per Fehlerspeichereintrag/ *Diagnostic Trouble Code*, DTC_i, festhaltbares) Fehlerverhalten sein. Jede dieser Variablen erhält den Wert wahr zugewiesen, falls der mit der Variable in Verbindung stehende Effekt festgestellt werden konnte.

Ein Beispiel hierfür ist ein permanenter Defekt des Fensterhebers, der zur Kundenbeschwerde „Fenster bewegt sich nicht“ führt. Es sei an dieser Stelle nochmals darauf hingewiesen, daß die Kodierung einer Kundenaussage mittels einer Variablen zusätzlich den Vorteil hat, Übersetzungsfehler auszuschließen.

Detektion: eine Maßnahme, um einen Fehler zu entdecken. Die Detektion überdeckt sich dabei mit dem Fehlereffekt. Detektionsmaßnahmen existieren auf der technischen Ebene und der Werkstattebene.

Auf der Ebene Entwicklung wird der Variable T_i der Wert wahr zugeordnet, falls die mit T_i in Verbindung stehende Bedingung durch eine Funktion im Steuergerät bzw. *on-board* Detektionsmaßnahme D_{on} beobachtet werden kann:

$$D_{on_i} : T_i \mapsto \{0, 1\}$$

In der Werkstatt wird die Variable G_i verwendet mit dem Wert wahr, falls die mit G_i korrelierende Bedingung von einer Funktion des off-board Testers D_{off} beobachtet werden konnte, also:

$$D_{off_i} : G_i \mapsto \{0, 1\}$$

Beispielhaft hierfür ist die Testerfunktion *Prüfe Kabel zum Fenstermotor auf Strom*, die die dazu korrespondierende Variable auf wahr setzt, falls Strom meßbar ist.

Vermeidungsmaßnahme: schließt das Steuergerät anhand von Messungen T_i und Fehlerspeichereinträgen DTC_i, daß ein verhinderbarer Fehler F_i vorhanden ist, so kann eine Vermeidungsmaßnahme initiiert werden, um den Übergang von einem fehlerhaften Zustand in ein Versagen des Systems zu vermeiden. Durch den Eingriff kann dann aber wiederum ein Systemverhalten entstehen, das vom Kunden als fehlerhaft aufgefaßt und beanstandet wird. Für einen Fehler F_i können mehrere Vermeidungsmaßnahmen existieren. Die Vermeidungsmaßnahmen existieren auf der technischen Ebene.

$$\begin{array}{llll} (T_i & \rightarrow & \bigwedge_i DTC_i) & \bigwedge \\ (\bigwedge_i DTC_i & \rightarrow & F_i) & \bigwedge \\ (F_i & \rightarrow & \bigvee_i CM_i) & \bigwedge \\ (CM_i & \rightarrow & CC_i) & \end{array}$$

Der Variablen CM_i wird der Wert wahr zugeordnet, falls die dazu korrespondierende Maßnahme eingeleitet wurde, nachdem der Fehler entdeckt wurde. Beispielsweise werden im Falle einer Überhitzung des Fenstermotors die Bewegungsfunktionalitäten des Fensters abgeschaltet, so daß der Motor abkühlen kann.

Reparaturmaßnahme: die Aufgabe einer Reparaturmaßnahme ist die Heilung des Systems. Die erkannte(n) Fehlerursache(n) stehen in Verbindung mit einer austauschbaren Einheit, auf die verschiedene Reparaturmaßnahmen RM_i angewendet werden.

$$\bigvee_i F_i \rightarrow \bigvee_i RM_i$$

Falls für einen Fehler mehr als eine Reparaturmaßnahme existiert, müssen diese gewichtet werden. Hierfür wird später eine Funktion $\omega(RM_i)$ definiert, die die Reparaturmaßnahmen anhand der Erfolgsrate der Reparaturmaßnahme $Pr(RM_i)$ und ihren Kosten $costs(RM_i)$ gewichtet. Auf die Gewichtung wird näher in Abschnitt 4.8 eingegangen.

4.4.3 Konstruktion der logischen Formeln

Durch die vorgestellte Abbildung der FMEA auf logische Variablen entsteht ein logisches, formales Fehlermodell auf verschiedenen Abstraktionsebenen bzw. Schichten.

In diesem Abschnitt wird die Konstruktion der logischen Formeln beginnend auf der höchsten Abstraktionsebene vorgestellt.

Es sei an dieser Stelle darauf hingewiesen, daß obwohl das Fehlermodell auf allen drei Ebenen definiert wird, nicht alle Ebenen zwingend notwendig sind für das Gesamtmodell. So können Fehler entstehen, ohne daß sie vom Kunden bemerkt werden. Zudem sind bei mechanischen Fehlern meist keine Fehlerspeichereinträge des Steuergeräts vorhanden.

Funktionsmodell

Eine Funktion, im weiteren Verlauf mit f_i bezeichnet, hat eine Menge von Eingaben, die im einzelnen mit I_i bezeichnet werden, sowie ein internes, mit $beh(f_i)$ bezeichnetes Verhalten, das durch die Menge der Interaktionen der Funktion entsteht. Die Eingaben der Funktion werden verarbeitet und führen zu einer Menge von Ausgaben, die im einzelnen mit O_i bezeichnet werden:

$$f_i(I_i, beh(f_i)) = O_i$$

Fehlermodell auf Kundenebene

In Abschnitt 2.2.1 wurde die von [Avi67] zuerst definierte und später in mehreren Arbeiten erweiterte Fehlerterminologie vorgestellt. [ALRL04] verstehen unter einem Systemversagen, falls das auftretende Verhalten eines Systems oder Dienstes vom spezifizierten Verhalten abweicht. Dies kann durch falsche oder fehlerhafte Eingaben oder ein fehlerhaftes internes Verhalten verursacht werden.

$$(\neg O_i \rightarrow \bigvee_i CC_i)$$

Ein Fehlverhalten einer Funktion, wie bspw. einer Kundenfunktion, führt zu einer unerwarteten Ausgabe, die festgehalten wird, in dem der mit der Ausgabe korrespondierenden Variable $\neg O_i$ der Wahrheitswert wahr zugewiesen wird. Das unerwartete Verhalten führt zu einer Kundenbeschwerde in einer Werkstatt. Dadurch wird der mit der Kundenbeschwerde in Relation stehenden Variable CC_i der Wert wahr zugewiesen.

Fehlermodell auf Werkstattebene

Durch die Kundenbeschwerde in der Werkstatt startet die Inspektion des Fahrzeugs und somit die Reparatur mit der off-board Diagnose. Wie in Abschnitt 2.3 erwähnt, ist die Aufgabe der automobilen Diagnose die Bestimmung der Ursachen der vorhandenen Fehler und die Zuweisung der Fehlerursache(n) auf eine sogenannte austauschbare Einheit. Durch Reparatur oder Austausch dieser Einheit kann dann das fehlerhafte System oder die Komponente geheilt werden.

Falls ein Fehler als präsent angenommen wird, wird der mit dem Fehler verbundenen Variable F_i der Wert wahr zugewiesen.

Die Werkstatt versucht anhand der erfaßten Symptome wie Kundenaussage und gesetzten Fehlerspeichereinträgen auf die vorhandenen Fehler zu schließen. Zusätzlich kann die Werkstatt weitere durch den Tester gesteuerte Messungen und Prüfungen durchführen, die im weiteren Verlauf als Testerfunktionen D_{off_i} bezeichnet werden.

Im Falle von Kundenbeschwerden beginnt die Reparatur in der Werkstatt mit der Analyse und Verifizierung der vorhandenen Beschwerden. Der Grund hierfür ist, daß aufgrund der langen Lebenszyklen viele Kundenbeschwerden der Werkstatt bekannt sind und somit ein schnelles Schließen auf die Fehlerursache möglich ist.

Durch die Analyse der Kundenbeschwerde ergibt sich:

$$\bigvee_i CC_i \rightarrow \bigvee_i F_i$$

CC_i und F_i sind Variablen, denen der Wert wahr zugeordnet wird, falls der mit F_i in Verbindung stehende Fehler vorhanden ist sowie die mit CC_i verbundene Kundenbeschwerde der Werkstatt berichtet wurde. Beispiele hierfür sind bekannte Kundenbeschwerden wie eine gesetzte Warnleuchte oder die Beschwerde, daß der Fensterheber nicht mehr vollständig schließt.

Die Fehlerursache steht in Verbindung mit einer austauschbaren Einheit, auf die eine oder mehrere Reparaturmaßnahme(n) RM_i angewendet werden können.

$$\bigvee_i F_i \rightarrow \bigvee_i RM_i$$

Ziel einer Reparaturmaßnahme ist die anschließende Wiederherstellung des Systems.

Falls die Variable F_i auf verschiedene austauschbare Einheiten verweist, müssen die zugrundeliegenden Fehlerursachen isoliert und durch zusätzliche Beobachtungen diskriminiert werden. Dies geschieht durch die Hinzunahme von Beobachtungen einer tieferen Ebene des Schichtenmodells. Aufgrund des beschränkten Zugriffs der Werkstätten auf die Steuergeräte ist in diesem Fall die Fehlerdetektion wichtiger als die Fehlerdiskriminierung. Dies steht im Gegensatz zu [PBC+02] sowie [SRB+02].

Für die Werkstatt bedeutet dies die Hinzunahme zusätzlicher off-board Beobachtungen durch Tester-Funktionen, die der Variable G_i den Wahrheitswert wahr zuweisen, falls die mit G_i in Verbindung stehende Beobachtung durch die off-board Testerfunktion D_{off_i} observiert werden konnte, also:

$$D_{off_i} : G_i \mapsto \{0, 1\}$$

Steht der Fehler aber in Zusammenhang mit einem Steuergerät, so stehen der Werkstatt eine Menge von Fehlerspeichereinträgen zur Verfügung. Der Variablen DTC_i wird dabei der Wert wahr zugewiesen, falls der mit DTC_i in Verbindung stehende Fehlerspeichereintrag gesetzt wurde.

Somit ergeben sich folgende Formeln auf Werkstattebene:

$$\begin{array}{lcll} (\bigvee_i CC_i & \rightarrow & \bigvee_i F_i) & \bigwedge \\ (\bigvee_i G_i & \rightarrow & \bigvee_i F_i) & \bigwedge \\ (\bigwedge_i DTC_i & \rightarrow & \bigvee_i F_i) & \bigwedge \\ (\bigvee_i F_i & \rightarrow & \bigvee_i RM_i) & \end{array}$$

Es sei an dieser Stelle nochmals darauf hingewiesen, daß mechanische Fehler von den Steuergeräten nicht direkt entdeckt werden können. In diesem Fall sind deshalb keine Fehlerspeichereinträge vorhanden.

Fehlermodell auf technischer Ebene

$$\bigwedge_i T_i \rightarrow DTC_i$$

Die Fehlererkennung im Steuergerät erfolgt durch Auswertungen verschiedener Beobachtungen (vgl. Abbildung 2.12), die mit Variablen T_i in Verbindung stehen. T_i bekommt hier den Wert wahr zugewiesen, falls als Ergebnis die mit der Variablen T_i in Verbindung stehende Beobachtung festgestellt werden kann. Basierend auf den Ergebnissen der Beobachtungen setzt das Steuergerät einen oder mehrere DTC, die in Verbindung mit Variablen DTC_i stehen, die den Wert wahr annehmen, falls der zugehörige DTC gesetzt wurde.

Auch im Steuergerät findet eine Analyse der Fehlerspeichereinträge zur Identifizierung der fehlerhaften Funktion oder Komponente statt. Ziel eines DTC oder DTC-Muster ist es, einen Fehler eindeutig zu identifizieren:

$$\bigwedge_i DTC_i \rightarrow F_i$$

Fehler von sicherheitsrelevanten Steuergerätefunktionen müssen auf Steuergerät- bzw. Systemebene verhindert oder abgeschwächt werden, bevor sie zu kritischen Effekten führen können. Das ist die Aufgabe der präventiven Diagnose. Die präventive Diagnose initiiert Gegen- oder Vermeidungsmaßnahmen, die in Verbindung mit Variablen CM_i stehen, die im Falle der Einleitung der Maßnahme den Wert wahr erhalten.

$$F_i \rightarrow \bigvee_i CM_i$$

Das Auslösen einer präventiven Maßnahme kann zu einer Einschränkung einer kundensichtbaren Funktion führen und somit zu einer Kundenbeschwerde, die in Verbindung mit CC_i steht, die in diesem Falle auf wahr gesetzt wird.

$$CM_i \rightarrow CC_i$$

Beispielhaft für obigen Zusammenhang ist das Entdecken eines Einklemmens des Fensters. Der Einklemmschutz des Fensters ist sicherheitsrelevant, da ein Versagen des Einklemmschutzes zum Einklemmen von Objekten führen kann. Deshalb wird, falls ein Versagen des Einklemmschutzes erkannt wird, das Fenster permanent deaktiviert. Dies kann dann zu einer Kundenbeschwerde, daß das Fenster nicht funktioniert, führen.

4.4.4 Abdeckung anderer Diagnoseansätze

In diesem Abschnitt soll gezeigt werden, daß die Kodierung der Diagnose mittels boolescher Variablen im vorgestellten Ansatz in der Lage ist, die geläufigsten Diagnoseansätze abzudecken.

Modellbasierte Diagnose. Die modellbasierte Diagnose überwacht ein System mittels aussagenlogischer Formeln. Dabei wird oft ein System anhand der Ein- und Ausgaben untersucht. Die Umwandlung der modellbasierten Diagnose für boolesche Diagnose wurde von [Bau06] und [Tri09] gezeigt. Zudem zeigt [KB10] Beispiele hierfür.

Expertenwissenbasierte Diagnose. Die Regeln eines Expertensystems können mittels einer Zuweisung einer Variablen für den rollenbasierten Ansatz eingesetzt werden. So läßt sich die Regel „discretised Hall-effect sensor signal did not change after movement of window" einer Variable T_i zuweisen, die dann ausgewertet werden kann. Das Beispiel für die Kodierung wurde aus [KB10] entnommen.

Quantitative Diagnose. Quantitative Diagnosen können einfach kodiert werden, indem der zu überwachende Ausdruck einer Steuergerätefunktion D_{on_i} zugeordnet wird, die dann ausgewertet werden kann. Soll beispielsweise geprüft werden, ob die gemessene Temperatur eines Bauteils t_i geringer als 70 Grad ist, so kann dies wie folgt dargestellt werden:

$$A_i := (t_i \leq 70)$$
$$D_{on} : T_{A_i} \mapsto \{0, 1\}$$

Qualitative Diagnose. Bei der qualitativen Diagnose [Kui94] werden qualitative Aussagen wie der „Druck Ölpumpe zu hoch" durch die Diagnose geprüft. Da es sich bei der qualitativen Diagnose schon um eine Kodierung handelt, läßt sich diese ohne Weiteres an den rollenbasierten Diagnoseansatz anpassen.

Fehlerverriegelungsmatrix. Die Kodierung einer Fehlerverriegelungsmatrix wurde gezeigt in [Tri09, Kap. 4]. Die durch einen Monitor M_i zu überwachende Bedingung einer Fehlerverriegelungsmatrix aus Zeile i und Spalte j wird mit der Variable B_i kodiert und kann dann dargestellt werden als

$$M(i,j) = B_i = \begin{cases} wahr & \text{falls } B_i \text{ beobachtet werden kann} \\ falsch & \text{sonst} \end{cases}$$

Durch Einordnung in das Rollenmodell wird B_i dann entweder zu CC_i, G_i oder T_i.

Fehlererfassung von zeitveränderlichen Größen in dynamischen Systemen. Bisher wurde im Verlauf dieses Abschnitts gezeigt, wie „statische", zeitunabhängige Fehler erkannt werden können. Im Automobil werden jedoch auch viele *dynamische Systeme* eingesetzt, deren Verhalten von zeitveränderlichen Kenngrößen abhängig ist. Beispielhaft hierfür sind die in Abschnitt 2.1.4 vorgestellten Regelsysteme. Bei solchen Systemen ist somit das Verhalten des Systems über mehrere Zeitpunkte zu prüfen, wie im Folgenden gezeigt wird.

Ein zu überwachendes Verhalten bzw. Wert ist genau dann als fehlerhaft anzusehen, wenn die Bedingung für ein Fehlverhalten länger als eine definierte Zeitdauer vorliegt.

Die Mindestdauer ist notwendig, da der zeitabhängige Wert sich ändern kann, bevor ein Versagen eintritt (vgl. Definition Fehlerzustand in Abschnitt 2.2.1).

Für die Erfassung des fehlerhaften Verhaltens für solche Systeme existieren mehrere Verfahren. Abschnitt 2.4.4 zeigte eine auf einem mathematischen Prozeßmodell des Systems basierende Modellierung. Ein bekannter Ansatz zur formalen Spezifizierung von zeitlichem Verhalten bietet die lineare temporale Logik [Pnu77, Pnu86] oder die darauf aufbauende, für die Diagnose verteilter Systeme entwickelte SALT-Sprache [Bau06]. Prädikatenlogik läßt sich für die Modellierung des zeitlichen Verhaltensablauf nur schwerlich einsetzen, da die Prädikatenlogik das Systemverhalten für einzelne Zeitpunkte und nicht im zeitpunktübergreifenden Verlauf erfaßt.

Der in der Dissertation eingesetzte Ansatz ermöglicht den Einsatz all dieser Verfahren, da sich eine Auswertung solcher Formeln auf boolesche Werte zurückführen läßt (vgl. [Tri09, KB10]). Für eine Entdeckung von zeitkontinuierlichen Fehlern werden diese booleschen Aussagen in bestimmten Zeitrastern ausgewertet, wie Listing 4.1 zeigt.

```
void ECU_function(){ /* Programmlogik einer ECU */
  while (TRUE) { /* erledige ständige ECU Tasks in  verschiedenen Zeitrastern */
    task_1ms();
    task_10ms();
    task_100ms();
     ....;
  }
}
#define FAULT_Fi_ACTIVE_TIME 100 /* 100 ms. muss Fehler vorliegen */

void task_10ms (....){
    ......;   /* Funktionslogik */
  if  (T1 ≥ X) /* Fehlerzustand liegt vor, falls Monitor T1 größer oder gleich Wert X */
    error_active++; /* Fehlerzustandszähler */
  else{  /* Fehlerheilung! */
    if  (error_active  > 0)   /* Vermeide Überlauf */
      error_active−−;
    else
      error_active  = 0;
  }
  if  (error_active  ≥ 10){ /* (FAULT_Fi_ACTIVE_TIME / Zeitintervall) */
    SET_DTC(DTCi,...); /* setze Fehlerspeichereintrag mit Hintergrundinformationen */
    SET_BIT_DTCi(); /* Flag für DTC gesetzt */
  }
  if  ((error_active  < 10)  && (BIT_DTCi_SET())){ /* DTC war gesetzt, aber Heilung! */
    ERASE_DTC(DTCi);
    CLR_BIT_DTCi(); /* Flag für DTC gelöscht */
  }
}
```

Listing 4.1: Beispiel für dynamische Fehlerentdeckung im Steuergerät

4.4.5 Zusammenfassung

In diesem Abschnitt wurde gezeigt, wie das formale Diagnosemodell aufgebaut ist. Es entsteht durch eine Abbildung der für die Diagnose relevanten FMEA-Daten in ein

formales, logisches Schichtenmodell. In den einzelnen Ebenen des Schichtenmodells werden die Fehler der FMEA und ihr beobachtbares Verhalten mittels logischer Variablen beschrieben. Die Darstellung der Diagnoseformeln mit logischen Variablen ermöglicht die Einbindung der verschiedensten Diagnoseansätze.

4.5 Datenmodell

Das Diagnosedatenmodell beinhaltet die in Abschnitt 4.2 vorgestellte hierarchische Funktionsstruktur und erweitert diese um ein Datenmodell zur Erfassung der mit MSC spezifizierten funktionalen Interaktionen in Abschnitt 4.5.1.

Abschnitt 4.5.2 zeigt dann das Datenmodell der erweiterten FMEA, die zur Erfassung der möglichen Fehler der Funktionen und ihren Interaktionen durchgeführt wird. Dabei werden die definierten Vermeidungs- und Entdeckungsmaßnahmen entsprechend der im letzten Abschnitt gezeigten Abbildungsvorschrift in logische Formeln als Teil des Datenmodells transformiert. Die Daten werden durch die Einbindung von Lebenszyklusdaten zentral gewartet und dann an die einzelnen Tester übertragen.

Das Datenmodell ist Grundlage für die automatisierte Erstellung der Diagnose. Zudem ermöglicht das Datenmodell eine Erstellung einer Zulieferersicht (vgl. [BGK+09]) und somit eine frühzeitige Einbindung von Zulieferern in die Entwicklung der Diagnose.

4.5.1 Datenmodell MSC

Neben der Darstellung der Datenstruktur für die hierarchische Funktionsstruktur werden in diesem Abschnitt die mit MSC spezifizierten, möglicherweise komponentenübergreifenden, Interaktionen in einem Datenmodell erfaßt. Dabei standen verschiedene auf der MSC-Empfehlung [MSC04] basierende Ansätze zur Wahl. Für die Spezifikation der Interaktionen im Rahmen der Arbeit wurde jedoch der Ansatz von Krüger et al., das *Managed Service Concept* [AKMP05, EHK+07, EKM+07], aufgrund seines großen Funktionsumfangs gewählt.

Die Darstellung der komplexen, komponierbaren Interaktionen basiert dabei auf Arbeiten von Krüger et al. [Krü00, AKMP05], die für den Kontext der Arbeit erweitert werden. Abbildung 4.6 zeigt die Erweiterung des Funktionsmodells um diese Interaktionen, die für die Erfassung der verteilten Funktionen benötigt werden. In blau gehaltene Elemente stellen dabei Erweiterungen des logischen Aufbaus aus [AKMP05] dar. Die einzelnen Elemente werden in diesem Abschnitt vorgestellt. Dabei handelt es sich um eine verkürzte Darstellung des Kapitels 4.4 aus [Krü00].

Interaction: Interaction stellt zum einen die Verbindung zum hierarchischen Funktionsmodell aus Abbildung 4.3 dar und zum anderen die Obermenge der durch MSC spezifizierten Interaktionen. Die Interaktionen setzen sich aus ein- oder zweistelligen Operatoren zusammen, die verknüpft werden können. Die Interaktionen kommunizieren über Kanäle in Form von Nachrichtenübertragungen. Neben den Funktionen werden auch die Interaktionen mittels der erweiterten FMEA analysiert. Interaktionen bestehen entweder aus Nachrichten oder aus komponierten Operatoren.

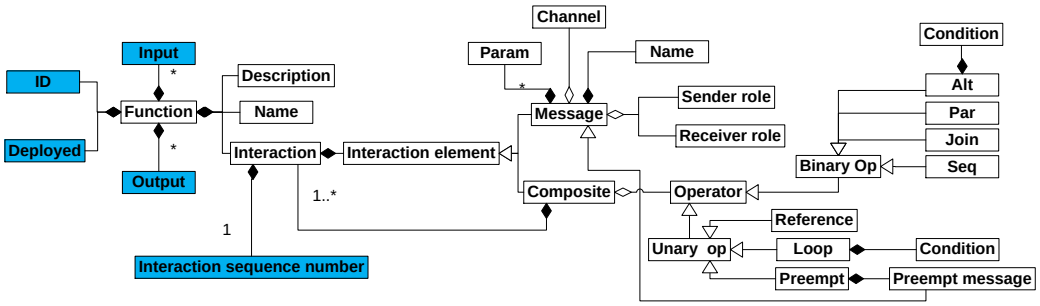


Abbildung 4.6: Erweitertes Datenmodell MSC. Quelle: basierend auf [AKMP05] und Abbildung 4.3

Message/ Nachricht: die Komponenten kommunizieren mittels asynchroner Nachrichten miteinander. Die Nachrichten bestehen dabei aus **Parametern**, **Channel** bzw. **Kanälen**, einem **Namen** für die Nachricht sowie aus einer **Sender Role** und einer **Receiving Role**. Das Konzept des Kanals stammt dabei aus FOCUS [BS01].

Composite: Composite stellt den Verknüpfungsoperator dar, mit dem Interaktionen komponierbar sind.

Operator: besteht aus **einstelligen** bzw. **zweistelligen Operatoren**.

Unary Operator bzw. einstelliger Operator: bedeutet, daß der Operator über genau einen Operand verfügt.

Reference: der Reference-Operator (vgl. [Krü00, S. 128]) referenziert auf einen anderen MSC. Dadurch können MSC wiederverwendet und gekapselt werden.

Loop: mit Loop werden die Schleifen dargestellt. Krüger unterscheidet dabei mit *guarded loops*, *bounded loops*, *unbounded loops* und *infinite loops* vier Arten von Schleifen, mit denen *for-*, *do..while-* und *while*-Schleifen abgebildet werden können [Krü00, S. 126ff.].

Preempt: der Preempt-Operator (vgl. [Krü00, S.128 und Kap. 4.7.6]) ermöglicht die Realisierung von Präemptionen, mit denen der Ablauf von Prozessen unterbrochen und später wiederaufgenommen werden kann. Dies kann ohne Zustimmung des zu unterbrechenden Prozesses geschehen. Ein Beispiel für eine Präemption stellt das automatische Öffnen des Fensterhebers dar, das jederzeit durch ein Drücken des Fensterschalters unterbrochen werden kann.

Binary Operator bzw. zweistelliger Operator: bedeutet, daß der Operator über genau zwei Operanden verfügt.

Alt/Alternative: der Alt-Operator [Krü00, S.120] ermöglicht die Auswahl zwischen Entscheidungen und somit strukturierte Anweisungen wie bspw. *if-* und *switch...case-*Anweisungen.

Par/ Parallel: der Par-Operator erlaubt die Darstellung von gleichzeitig ablaufenden Ereignissen.

Join/ Verknüpfung: mittels des Join-Operators [Krü00, S. 123] ist es möglich, zwei Operanden eines MSC, die Nachrichten über denselben **Channel** mit den gleichen Labels austauschen, miteinander zu verknüpfen.

Seq/ Sequential Composition: der Sequential Operator [Krü00, S. 119] ermöglicht eine genauere Verzahnung bzw. Komposition zweier MSC, indem gewählt werden kann, welcher MSC zuerst ausgeführt werden soll. Es handelt sich hierbei um eine Erweiterung der sequentiellen Komposition aus [Hoa04, Kap. 5].

Neue Elemente: **Input/ Output:** die Einbindung von Ein- und Ausgaben des MSC verknüpft den MSC mit FOCUS [BS01]. Ist ein Zulieferer darauf bedacht, sein Know-how zu schützen, so ist er daran interessiert, das Innenleben seiner Funktionsumfänge zu kapseln und für sich zu behalten. Die Bedatung der Ein- und Ausgaben ermöglicht dem OEM aber in diesem Fall eine Black Box-Sicht auf die Funktionalität und er kann die Komponente trotzdem diagnostizieren. Das heißt, gegeben die Eingaben einer Komponente lassen sich die Ausgaben derselben Komponente bestimmen, ohne daß zwingend Wissen über das Innenleben vorhanden sein muß.

Interaction Sequence Number: die Interaction Sequence Number versieht jede Interaktion mit einer eindeutigen Nummer. Aufgrund der Diagnose kann die Notwendigkeit bestehen, in die Funktionalität und Ablauf einer mit MSC zu beschreibenden Funktion präventiv einzugreifen, um bspw. den MSC um Anweisungen für ein Vermeiden eines Systemversagens zu erweitern. Aufgrund der Eindeutigkeit dieser Zahl können so die für die Diagnose notwendigen Funktionalitäten an der richtigen Stelle eingefügt werden. Durch die Sequenznummer entsteht so ein Ablaufplan für die Generierung von Programm-Code aus dem MSC.

4.5.2 Datenmodell erweiterte FMEA

Abbildung 4.7 zeigt das Datenmodell der erweiterten FMEA. Das Datenmodell zielt sowohl auf die erkannten Probleme der FMEA als auch auf die in Kapitel 3 erwähnten Top-Probleme. In diesem Abschnitt wird der Aufbau des Datenmodells genauer dargelegt.

Übernommene Elemente vom Funktionsmodell: mehrere Elemente des Funktionsmodell aus Abbildung 4.3 wurden übernommen. Es handelt sich dabei um die Attribute **Function**, **Input**, **Output**, **Deployed** sowie **Refined**. Refined verweist dabei auf die unteren Ebenen der Funktionshierarchie aus Abschnitt 4.2.1.

Übernommene Elemente vom MSC-Funktionsmodell: aus dem Datenmodell der MSC-Struktur (Abb. 4.6) wurde zusätzlich die Obermenge der möglichen Interaktionen, **Interaction**, übernommen.

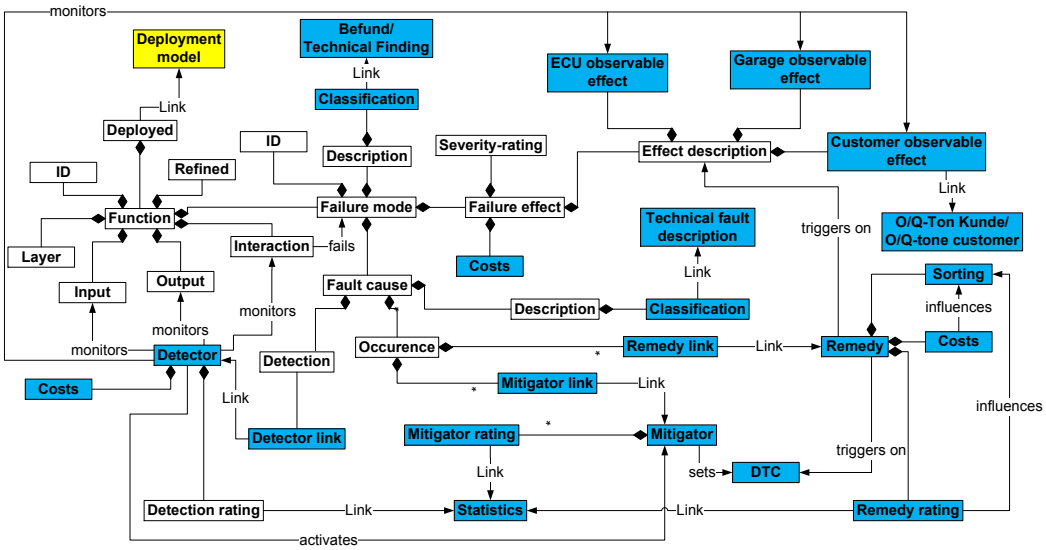


Abbildung 4.7: Datenmodell erweiterte FMEA. Quelle: eigene Darstellung, basierend auf [DIN80] sowie Abbildungen 4.3 und 4.6

Übernommene Elemente aus der FMEA-Struktur: aus der Standard-FMEA wurden **Failure Mode** mitsamt Beschreibung, **Fault cause** mitsamt Beschreibung, **Failure effect** mitsamt Beschreibung, **Severity rating**, **Detection**, **Detection rating** sowie **Occurence/Vermeidungsmaßnahme** übernommen. Nicht übernommen wurde das **Occurence rating**, das durch das **Mitigator rating** sowie **Remedy rating** ersetzt wurde. Eine genaue Erklärung der aus der Standard-FMEA übernommenen Attribute findet sich in Abschnitt 4.3.3.

Neue Elemente: die neuen Elemente des Datenmodells sind in blauer Farbe gehalten und werden im Folgenden detaillierter dargelegt.

Kosten: ermöglicht eine Kostenbewertung der einzelnen Fehlereffekte. Somit können nicht nur die monetären Effekte präventiver Maßnahmen, sondern auch von Nichtqualität verglichen werden. Dabei werden die Kosten über den gesamten Lebenszyklus erfaßt (vgl. [KI00]).

ID: ein Fehlermodus wird mit einer ID versehen, um den Fehlermodus mitsamt seinen Daten genau zu identifizieren. Zudem ermöglicht die ID, den Fehler in ähnlichen Komponenten durch Referenzieren wiederzuverwenden.

Classification: die Klassifizierung einer Fehlerursache wird an die **ID** der Funktionsstruktur angelehnt und zusätzlich mit einer komponenten- und produktübergreifenden einheitlichen Klassifizierung eines Fehlers K_i erweitert:

$$FC_iAK_iFB_iSB_iSU_i - K_i$$

Die Klassifizierung beschreibt dabei die Art des Fehlers sowie gegebenenfalls die Fehlerbedingung. Ein Beispiel hierfür ist die Funktion *Fensterheber-Fenster automatisch öffnen-Detect Jamming-Read Jamming Detection Sensors-Analyse Hall Sensors* mit der Fehlerursache *Kurzschluß an Masse*. Dieses Verfahren ermöglicht nicht nur eine

Strukturierung der Fehlermodi, sondern auch eine Einordnung in vorhandene Daten des Qualitätsmanagements. So ist es mittels Datenauswertungen möglich, schnell einen Überblick über die Fehlerlandschaft zu erhalten.

Technical finding/ Befund: ist die in der Werkstatt erstellte technische Beschreibung des gefundenen Mangels des Fahrzeugs. Der Befund wird dabei an die **Klassifikation des Fehlers** gekoppelt. Ein weiterer Vorteil dieser numerischen Erfassung liegt darin, daß Befunde sprachenunabhängig erfaßt werden können.

Q-Ton-Kunde: aufgrund der Tatsache, daß viele Fehler nur in Zusammenarbeit mit dem Kunden entdeckt und gelöst werden können, ist eine Erfassung der Aussage des Kundens notwendig. Aufgrund des weltweiten Einsatzes von Fahrzeugen und somit verschiedener Sprachen sowie des differierenden technischen Wissens der Kunden läßt sich eine Aussage des Kunden am besten in numerischer Darstellung erfassen. Für diese Erfassung muß die Kundenaussage in die Klassifizierung eingepaßt werden. Somit wird die Kundenaussage qualifiziert.

Statistische Daten: für die Wartung der Diagnose werden statistische Daten aus dem Lebenszyklus eingebunden. Auf der einen Seite dienen sie als Einfluß zur **Sortierung** der Werkstattmaßnahmen, zum anderen werden sie verwendet, um die Maßnahmen der FMEA durch statistische Daten anstelle von geschätzten Auftretenshäufigkeiten (siehe auch [KI00]) zu bewerten. Auf die Erstellung und Wartung durch statistische Daten wird genauer in Abschnitt 4.8 eingegangen.

Technische Fehlerbeschreibung: bezeichnet die Fehlerbeschreibung durch die Entwickler, die das technisch tiefste Verständnis haben. Die technische Fehlerbeschreibung orientiert sich dabei auch an der Fehlerklassifikation.

Customer observable effect: bezeichnet, wie die Rolle Kunde den Fehler und seine(n) Effekt(e) entdecken kann (vgl. auch Einbindung Kundenaussage in Abschnitt 3.9).

Garage observable effect: bezeichnet, wie die Rolle Werkstatt den Fehler und seine(n) Effekt(e) entdecken kann.

ECU observable effect: bezeichnet, wie die Rolle Entwicklung den Fehler und seine(n) Effekt(e) entdecken kann.

Durch die beobachtbaren Effekte werden die Fehlereffekte der FMEA in das Rollenmodell der Arbeit eingeordnet. Dabei lassen sich die Effekte, genauso wie die Interaktionen mitsamt ihren Ein- und Ausgaben, formal spezifizieren. Eine genauere Darstellung der Einordnung der Fehler und ihrer Effekte in das formale Fehlermodell wird in Abschnitt 4.4 dargelegt.

Detector: überwacht die Interaktionen sowie **Ein-** und **Ausgaben** der Funktion auf Abweichungen vom spezifizierten Verhalten. Es handelt sich also hierbei um die Fehlererkennung der Diagnose. Da die Einsicht in Steuergeräte eingeschränkt sein kann (vgl. Abschnitt 3.3), muß die Fehlererkennung in der Lage sein, ein System oder Funktion anhand seiner Ein- und Ausgaben zu diagnostizieren, ohne Kenntnisse vom Innenleben zu benötigen. Zudem bezieht der Detektor auch das beobachtete Verhalten auf den Ebenen des Rollenmodells ein (**Customer/ Garage/ ECU observable effect**). Detektoren können sowohl im Steuergerät als auch im Tester eingesetzt werden. Wird ein Fehler im Steuergerät erkannt, kann die on-board Diagnose die präventive Diagnose bzw. **Mitigator** starten.

Mitigator link: verweist auf eine präventive Diagnosefunktion, **Mitigator**, deren Aufgabe es ist, Auswirkungen von Fehlern einzuschränken oder zu vermeiden. Diagnosefunktionen und FMEA-Elemente werden bewußt getrennt, um die Wiederverwendbarkeit sowohl der FMEA als auch der Diagnosefunktionen zu erhöhen.

Mitigator: der Mitigator wird von der Diagnose gestartet, um einen Fehler zu vermeiden oder abzuschwächen. So kann der Mitigator beispielsweise eine Funktion einschränken oder das Fahrzeug in den Notlaufmodus versetzen. Dieser Vorgang wird auch als präventive Diagnose bezeichnet. In den meisten Fällen setzt der Mitigator aber einen Fehlerspeichereintrag, um den gefundenen Fehler dauerhaft festzuhalten und zu dokumentieren.

Mitigator rating: das Standard-FMEA-Element Occurence rating bzw. Auftretenswahrscheinlichkeit wird wie erwähnt ersetzt durch **Mitigator-** und **Remedy rating**. Das Mitigator-Rating bewertet die statistische Wahrscheinlichkeit des Auftretens des Fehlers unter Berücksichtigung der Vermeidungsmaßnahmen. Da ein kundenwirksames Eingreifen eines Mitigators mit einem Fehlerspeichereintrag gesichert werden muß, kann die statistische Wahrscheinlichkeit auf der Ebene des Steuergeräts über die Anzahl der jeweiligen DTC bestimmt werden. Das Mitigator-Rating wird über die Lebenszyklusdaten automatisch gewartet.

Remedy link: verweist auf eine Reparaturmaßnahme, **Remedy**, in der Werkstatt. Die Aufgabe von Reparaturmaßnahmen ist die Heilung bzw. Wiederherstellung des defekten Systems. Die Reparaturmaßnahme und die FMEA-Elemente werden bewußt getrennt, um die Wiederverwendbarkeit zu erhöhen.

Remedy: Remedy bezeichnet eine Werkstattmaßnahme. Diese triggert auf den gespeicherten DTC, aber auch auf andere beobachtbare Effekte eines aufgetreten Fehlers.

Remedy rating: mit dem Remedy-Rating wird bewertet, wie häufig eine Werkstattmaßnahme erfolgreich durchgeführt wurde. Dieses Rating wird automatisch über die Lebenszyklusdaten gewartet.

Sorting: da mehrere Werkstattmaßnahmen für einen Fehler vorhanden sein können, werden die Maßnahmen anhand des Produkts aus ihrer Erfolgsrate und Kosten sortiert. Die Kosten bleiben meist relativ gleich über den Lebenszyklus, die Erfolgsrate wird jedoch automatisch über den gesamten Lebenszyklus hindurch gewartet (vgl. Abschnitt 4.7).

4.5.3 Datenmodell DTC

In diesem Abschnitt wird das Datenmodell für einen Fehlerspeichereintrag (DTC) [ISO05b] vorgestellt. Das Datenmodell wird dabei während der Entwicklungsphasen sowohl vom Zulieferer als auch vom OEM befüllt. Abbildung 4.8 zeigt das Datenmodell eines DTC. Dabei werden auf das Steuergerät nur die farbig markierten Elemente gespeichert. Das komplette Datenmodell dient zur Dokumentation der Fehlerspeichereinträge durch den Zulieferer und zur Validierung der DTC in den Test- und Integrationsphasen.

DTC-Number/ DTC-Nummer: gibt die Nummer des Fehlerspeichers mit einer hexadezimalen Zahl an. Jeder DTC wird mit einer eindeutigen Nummer versehen, die sich aus einer hexadezimalen Zahl für das Steuergerät, das den DTC setzte, sowie der

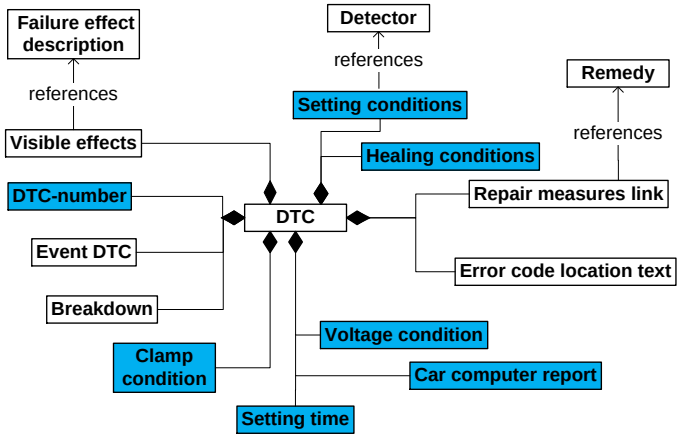


Abbildung 4.8: Datenmodell eines Fehlerspeichereintrags. Quelle: aus [ISO05b]

Nummer des DTC selber zusammensetzt.

Error code location text/ Fehlerspeichereintragstext: beschreibt den Fehlerspeichereintrag genauer.

Setting conditions/ Setzbedingungen: definiert, unter welchen Bedingungen der Fehlerspeicher gesetzt wird. Die Setzbedingungen werden im formalen Fehlermodell mit den on-board Monitoren kodiert. Dadurch ergibt sich auch die Verlinkung der Setzbedingungen mit der Detektionsmaßnahme aus der erweiterten FMEA. Zudem werden in den Integrationsphasen die DTC über die Setzbedingungen validiert.

Healing conditions/ Fehlerspeicherheilung: ein Fehlerspeichereintrag wird gesetzt, falls eine fehlerhafte Bedingung über einen bestimmten Zeitraum vorlag. Dieser Zeitraum wird in den Setzbedingungen mit angegeben. Das System kann sich aber auch in bestimmten Fällen selbst heilen, ohne daß eine Reparatur in der Werkstatt notwendig ist, bspw. durch die präventive Diagnose. Kann sich das System also von einem bestimmten Fehler selbst heilen, so muß der Fehlerspeichereintrag gelöscht werden, da der Fehler dann ja nicht mehr vorhanden ist (und um unnötige Reparaturen zu vermeiden). Die Bedingungen, unter denen sich das System heilen kann, werden mit den **Healing conditions** angegeben und referenzieren auf einen **Mitigator** aus dem FMEA-Datenmodell. Ein Beispiel für eine Fehlerheilung findet sich in der Auswertung eingehender Busnachrichten durch die überwachenden Diagnosefunktionen des ACC. Stellt diese Funktion eine fehlerhafte Nachricht fest, wird der erkannte Fehlerzustand mit einem DTC festgehalten. Eine solche fehlerhafte Nachricht kann beispielsweise durch Unterspannung entstehen. Um solche Fehler vermeiden zu können, erfolgt der Nachrichtenaustausch vor allem bei sicherheitsrelevanten Systemen zyklisch. Ist die Nachricht im nächsten Zyklus fehlerfrei, so liegt die Fehlerbedingung nicht mehr vor und der Fehlerspeichereintrag muß zurückgesetzt werden.

Event DTC/ Ereignisfehlerspeicher: unter einem Ereignisfehlerspeicher wird verstanden, daß ein Steuergerät A feststellt, daß das mit ihm kommunizierende Steuergerät B fehlerhaft ist und diese Erkenntnis speichert. Hintergrund ist, daß sich Steuergeräte, die zur Erfüllung einer sicherheitsrelevanten oder verteilten Funktion zusammenarbeiten, gegenseitig überwachen. Um nun aber zu verhindern, daß in der Werkstatt ein Steuergerät getauscht oder repariert wird, das einen Fehler nur beobachtete, wird das

Event DTC Flag eingeführt.

Visible effects/ Sichtbare Effekte: verlinkt den Fehler mit der Fehlereffektbeschreibung aus dem Datenmodell der erweiterten FMEA. Dadurch kann der Fehler in das Rollenmodell der Diagnose eingeordnet werden.

Breakdown/ Panne: gibt an, ob der Fehler pannenrelevant ist, d. h. ob der Fehler zu einem Liegenbleiben des Fahrzeugs führen kann.

Setting time/ Setzzeit: die Setzzeit hält den Zeitpunkt fest, zu dem der DTC gesetzt wurde. Die Zeit wird dabei in der Einheit 1 Millisekunde erfaßt.

Car computer report: gibt an, ob der vorhandene Fehler im Armaturenbrett oder im Car Computer angezeigt werden muß. Ist der Fehler emissionsrelevant, so muß der Fehler im Armaturenbrett aufgrund der OBD-Gesetzgebung mittels spezieller sogenannter *OBD-Lampen* angezeigt werden (vgl. [Cal94]).

Clamp condition/ Klemmenbedingung: gibt an, welche *Klemmenbedingungen* vorliegen müssen, damit ein Fehlerspeicher gesetzt wird. Der Begriff Klemme stammt noch aus der Steckerbezeichnung der allgemeinen Elektrotechnik und wurde mit der Elektrifizierung des Fahrzeugs übernommen. Bei der Klemme handelt es sich um einen Anschluß, der im Zustand „Klemme an“ einen dauerhaften Kontakt sicherstellt. Dies wird durch mechanische Fixierung der Klemme mit einem leitfähigen Gegenstand, beispielsweise der Fahrzeugmasse, erreicht. So bedeutet die Klemmenbedingung „Klemme 30 an“ einer elektronischen Komponente, daß die Komponente auch ohne aktivierte Zündung einschaltbar ist, da ein Dauerplus der Autobatterie das Bordnetz mit permanenter Spannung versorgt. Eine über Klemme 15 geschaltete Komponente hingegen läßt sich nur bei eingeschalteter Zündung aktivieren (aus [Rei06, Kap. 4.2.2]). Die Klemmenbezeichnungen in der Fahrzeugelektronik sind durch die DIN 72552-2 standardisiert [DIN71].

Voltage condition/ Spannungsbedingung: definiert die notwendige Bordnetzspannung bzw. die vorhandene Mindestspannung, die notwendig ist, damit ein DTC gesetzt werden kann. Dabei kann es sich um Ruhestrom, Standverbrauch, Start, Leerlauf und Fahrtzustände handeln [Rei06, S. 218].

Remedy link: verweist auf eine Reparaturmaßnahme, **Remedy**, in der Werkstatt. Die Aufgabe von Reparaturmaßnahmen ist die Heilung bzw. Wiederherstellung des defekten Systems. Die Reparaturmaßnahme und die FMEA-Elemente werden bewußt getrennt, um die Wiederverwendbarkeit zu erhöhen.

4.5.4 Extraktion

Aufgrund des beschriebenen strukturierten Aufbaus ist es nun dem Zulieferer möglich, die Daten in dem *Extensible Markup Language* (XML)-Datenformat [BPS+08] zu befüllen, aus dem dann die Daten mittels eines Parsers wie bspw. XPath [CD99] für den OEM extrahiert werden können.

Dabei sind vom Zulieferer zwingend alle Daten der erweiterten FMEA und DTC zu befüllen.

4.5.5 Modelltransformation

Die logischen Formeln werden entweder auf den Tester oder das Steuergerät gespeichert. Zur Laufzeit wird dann das gespeicherte Modell mit Meßwerten erweitert und die Variablen erhalten Wertzuweisungen gemäß ihrer beschriebenen Definition. Die Variablen mit ihren Belegungen stellen somit ein Erfüllbarkeitsproblem dar, das mit einem SAT-Solver gelöst werden kann. Jede mögliche Lösung des Solvers stellt dabei einen Diagnosekandidat oder Erklärung der Fehlerursache dar.

SAT-Solver benötigen als Eingabeformat ein spezielles Speicherformat für die booleschen Formeln, in den meisten Fällen das auf der *konjunktiven Normalform* (CNF) basierende DIMACS-CNF-Format [DIM93]. Eine Formel ist genau dann in konjunktiver Normalform, falls sie folgenden Aufbau hat:

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} l_{i,j} = (l_{1,1} \vee ... \vee l_{1,m}) \wedge .... \wedge (l_{n,1} \vee ... \vee l_{n,m}).$$

Eine Formel in konjunktiver Normalform besteht aus den atomaren Formeln, die auch als *Literale* bezeichnet werden. $l_{i,j}$ ist dabei ein *positives Literal*, $\neg l_{i,j}$ ein *negatives Literal*. Eine disjunktive Verknüpfung zweier Literale l_1, l_2 zu $(l_1 \vee l_2)$ wird als *Klausel* bezeichnet.

Zur Umwandlung der Formeln in die CNF wird dabei folgender Algorithmus aus [RN10, S. 253ff] verwendet:

1. Eliminiere $\leftrightarrow$. Ersetze $\alpha \leftrightarrow \beta$ durch $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$

2. Eliminiere $\rightarrow$. Ersetze $\alpha \rightarrow \beta$ durch $\neg\alpha \vee \beta$

3. Äquivalenzumformungen. Der Aufbau der CNF verlangt, daß $\neg$ nur bei einzelnen Literalen vorkommen darf. Deshalb wird das $\neg$ mittels Äquivalenzumformungen durch wiederholtes Anwenden folgender Äquivalenzen nach innen verschoben:

3.1 $\neg(\neg\alpha) \equiv \alpha$

3.2 $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ (De Morgan-Gesetz)

3.3 $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ (De Morgan-Gesetz)

4. Anwenden des Distributivgesetzes zur Verteilung von $\wedge$ über $\vee$. Ersetze:

4.1 $(a \wedge b) \vee c \equiv (a \vee c) \wedge (b \vee c)$

4.2 $a \vee (b \wedge c) \equiv (a \vee c) \wedge (b \vee c)$

Das DIMACS-Format basiert auf der CNF und hat folgenden definierten Aufbau: Kommentarzeilen werden mit einem kleinen c eingeleitet. Die mit *p* beginnende Zeile definiert mit *cnf* das Format der Datei gefolgt von zwei Zahlen, die angeben, wie viele Variablen vorhanden sind und wie viele auszuwertende Zeilen die Datei hat. Zeilen werden mit der Zahl 0 beendet. So stellt Tabelle 4.4 das DIMACS-CNF-Format für die Formel $((\neg x_1 \vee x_2) \wedge \neg x_3 \wedge x_4)$ dar.

c Beispiel DIMACS-CNF-Format
c abgebildete Formel: $((\neg x_1 \vee x_2) \wedge \neg x_3 \wedge x_4)$
p cnf 4 3
-1 2 0
-3 0
4 0

Tabelle 4.4: Beispiel für DIMACS-CNF-Datei. Quelle: basierend auf [DIM93]

4.6 Deployment-Modell

4.6.1 Deployment-Modell off-board Diagnose

Das Deployment-Modell für die off-board Diagnose gestaltet sich relativ einfach. Das Testsystem in den Werkstätten enthält, wie die Abbildung 4.9 zeigt, das Diagnosemodell im DIMACS-CNF-Format, bspw. in Form einer Textdatei. Die DIMACS-CNF-Datei dient als Eingabedatei für den SAT-Solver und wird zur Laufzeit mit den Beobachtungen erweitert. Durch die Beobachtungen können die booleschen Variablen mit Wahrheitswerten belegt werden. Dadurch ergibt sich ein boolesches Erfüllbarkeitsproblem, das durch den Einsatz eines SAT-Solvers gelöst werden kann. Eine genauere Darstellung hierzu findet sich in Abschnitt 4.7.4.

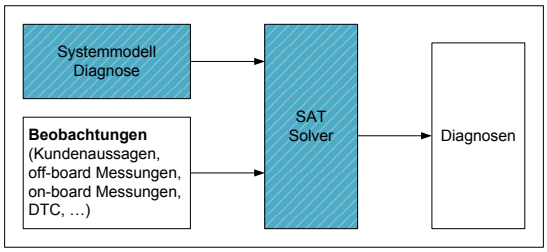


Abbildung 4.9: Deployment-Modell off-board Diagnose.Quelle: eigene Darstellung

4.6.2 Deployment-Modell on-board Diagnose

Das Deployment-Modell für die on-board Diagnose unterscheidet sich deutlich von dem des Testers. Der Grund hierfür liegt in der Tatsache begründet, daß in der Werkstatt nur eine nachträgliche, „post-mortem“ Diagnose stattfindet.

Im Gegensatz hierzu ermöglicht ein präventives Eingreifen der Diagnose das Vermeiden des Übergangs eines fehlerhaften Zustands in ein Versagen des Systems und verhindert so kunden- oder sicherheitsrelevante Versagen. In Abschnitt 2.2.1 wurde in der Fehlerterminologie dargelegt, daß der Übergang eines fehlerhaften Zustands in ein Versagen nicht zwingend ist, da beispielsweise der fehlerhafte Zustand in Form eines Variablenwerts überschrieben werden kann. Zusätzlich kann in vielen Fällen ein sich anbahnendes Versagen schon erkannt werden, bevor eine Aktion, die zu einem Versagen führen würde, durchgeführt wird (vgl. die Trendanalyse des FDI-Ansatzes in Abschnitt 2.4.4). In diesem Fall greift die Diagnose also präemptiv in die Funktion ein. Beispielhaft hierfür ist die Erhitzung des Fensterhebermotors bei einem automa-

tischen Fensterlauf, die zu einer Deaktivierung des Fensters führen kann, bevor der Motor erneut angesteuert wird, um so einen thermischen Schaden des Motors zu vermeiden.

Wie die Beispiele zeigen, ist die Einbettung einer zu überwachenden Funktion zusammen mit ihrer Diagnose in einen Funktionsrumpf, analog zum *Facade Pattern* aus [GHJV95] möglich, aber nicht immer ausreichend, um alle Fehler abzudecken.

Deshalb werden aufgrund der präventiven Diagnose die fehlervermeidenden Elemente der Diagnose in die Funktion integriert. Daraus folgt die Notwendigkeit, das bisherige Datenmodell um die Anteile der on-board Diagnose zu erweitern.

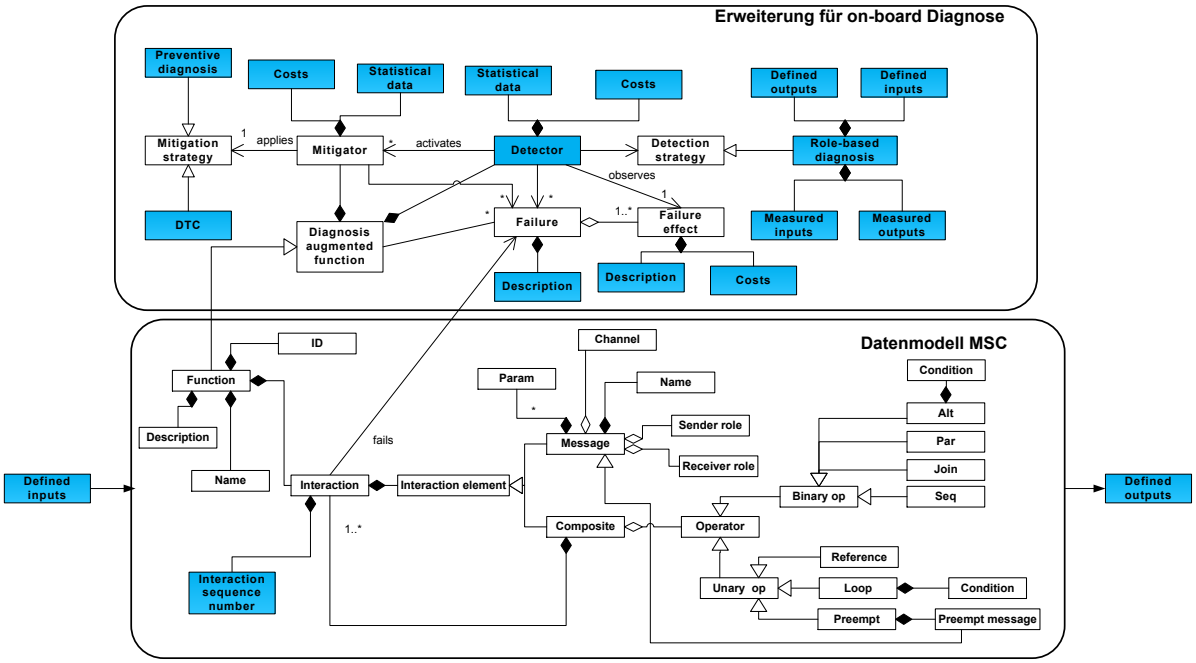


Abbildung 4.10: Erweiterung Datenmodell für on-board Diagnose.
Quelle: basierend auf [EHK+07, EKM+07]

Abbildung 4.10 zeigt die Erweiterung des MSC-Datenmodells aus Abschnitt 4.5.1 um die für die on-board Diagnose notwendigen Anteile. Die Abbildung basiert dabei auf Vorarbeiten aus [EHK+07, EKM+07] und wird um mehrere Elemente erweitert, die in blauer Farbe markiert sind.

Die Verknüpfung des Datenmodells für die „normalen“ MSC um die für die on-board Diagnose notwendigen Umfänge erfolgt über eine Erweiterung der Funktion sowie zusätzlich über das Datenelement **Interaktion** und einem möglichen Versagen der Interaktion. Eine Beschreibung der einzelnen Elemente des Datenmodells findet sich in Abschnitt 4.5.

An dieser Stelle sei nochmals die Wichtigkeit des Attributes **Interaction Sequence Number** erwähnt. Da die präventive Diagnose in eine Funktion eingreifen muß, müssen die Abläufe der Funktion durch die Diagnoseumfänge erweiterbar oder änderbar sein. Die Interaction Sequence Number ermöglicht das Einordnen der Diagnoseanteile.

Der **Detector** setzt dabei das Rollenmodell der Diagnose mitsamt seinen logischen For-

meln ein und aktiviert bei einem erkannten Fehler bzw. Fehlerzustand den Mitigator. Der Mitigator kann dann entweder das Systemversagen durch einen Eingriff verhindern oder -falls dies nicht möglich ist- einen DTC für die spätere Fehlerbehandlung in der Werkstatt setzen.

Für eine automatisierte Umwandlung des erweiterten Funktionsmodells in auf einem Steuergerät speicherbaren Code sind zwei Schritte notwendig. Im ersten Schritt muß das logische Funktionsmodell mit der gewählten Zielarchitektur des Steuergeräts und deren Aufbau bzw. Struktur gekoppelt werden. Diese Abbildung geschieht durch das **Deployment-Modell**.

Im zweiten Schritt muß das gekoppelte Modell automatisiert in Programm-Code **transformiert** werden, der dann wiederum nach einer Kompilierung durch einen Target-Compiler auf der Ziel-Hardware gespeichert werden kann.

Deployment-Modell

Die Aufgabe des Deployment-Modells für die on-board Diagnose ist die Abbildung des Modells der logischen Funktionsarchitektur aus Abbildung 4.10 auf die technische Architektur des Steuergeräts. Die einzelnen Beziehungen zwischen den Elementen der funktionalen und technischen Architektur zeigt Tabelle 4.5.

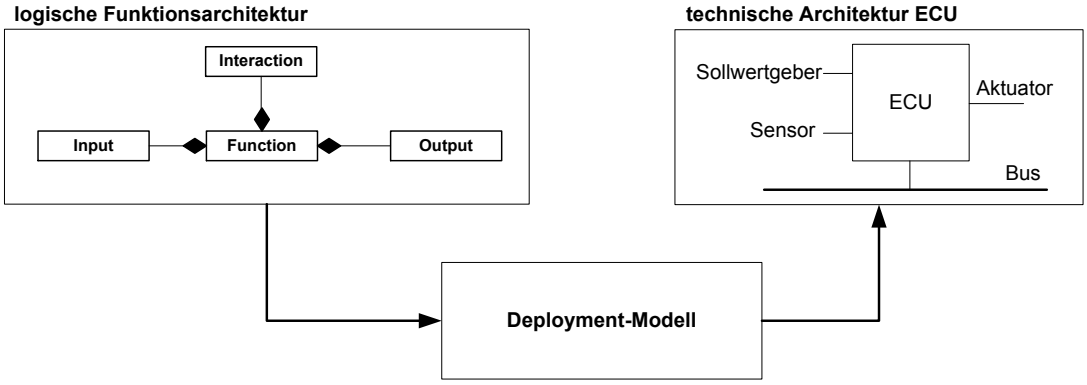


Abbildung 4.11: Deployment-Modell der on-board Diagnose.
Quelle: basierend auf [Sch08, Abb. 11] sowie [EKM+07, Abb. 4]

Die Relation N:M zwischen einer Funktion und dem Steuergerät ergibt sich aus der Tatsache, daß Kundenfunktionen über mehrere Steuergeräte verteilt sein können. So wurde beispielsweise in Abbildung 2.6 gezeigt, daß die Kundenfunktionen des ACC auf fünf Steuergeräten sowie weiteren Bauteilen verteilt sind. Deshalb erfolgt die Partitionierung der Funktionen auf ein Steuergerät ab der Ebene Funktionsbaustein.

Das Deployment-Modell ermöglicht also die Zuordnung der Funktionen mitsamt ihren einzelnen Ein- und Ausgaben zu den Bauteilen der technischen Architektur der Ziel-Hardware und wird vom Funktionsverantwortlichen in der Entwicklung (bspw. in Phase Systemarchitekturfunktionsdesign, vgl. Abschnitt A.3) durchgeführt.

Element logischer Architektur	Element technischer Architektur	Relation
Funktion	ECU	N:M
Input	Sensor, Bus, Stichleitung, Sternkoppler, Sollwertgeber, Pin, Kabel	N:1
Output	Aktuator, Stichleitung, Pin, Kabel	1:N
	Bus, Sternkoppler	N:M

Tabelle 4.5: Abbildung der funktionalen auf die technische Architektur.
Quelle: basierend auf [Sch08, Tab. 3] sowie [EKM+07, Kap. 2.3]

Transformation des Modells in speicherbaren Code

Nachdem das Modell an die Hardware des Zielsteuergeräts angepaßt wurde, ist eine Transformation des Modells in speicherbaren Code notwendig, der auf dem Zielsteuergerät gespeichert werden kann. Hierbei handelt es sich eigentlich um zwei Unterschritte, nämlich der Umwandlung des Modells in Source-Code einer kompilierbaren Hochsprache, die anschließend durch einen Compiler in Maschinen-Code umgewandelt werden kann, der auf einem Steuergerät speicherbar ist.

Dabei wird der schon in [Sch08] verfolgte Ansatz eingesetzt, das *Eclipse Modelling Framework* (EMF) (vgl. [SBPM08]) für die Transformation einzusetzen.

EMF stellt ein Modellierungs-Framework für die Programmiersprache Java dar. Mit EMF werden sogenannte *ECore-Metamodelle* erstellt, die sich unter anderem in Java-Klassen umwandeln oder als XML serialisieren lassen. Das ECore-Metamodell kann aus XML-Schemas oder wie im vorliegenden Fall aus UML- bzw. MSC-Modellen erstellt werden. Der anschließende Einsatz der *Java emitter templates* ermöglicht beispielsweise die Generierung von C-Code aus dem Modell.

In der Arbeit werden Tools eingesetzt, die das EMF zur Generierung von C-Code aus Message Sequence Charts nutzen. Die MSC werden in Java-ECore-Klassen umgewandelt, aus denen sich dann Code in der Zielsprache C [ISO99b] generieren läßt. Der C-Code kann dann anschließend durch einen Target-Compiler in ein auf einem Steuergerät speicherbares Format umgewandelt werden.

4.7 Effiziente Diagnose zur Laufzeit

„Veritas consistit in adaequatione intellectus et rei“
(Die Wahrheit besteht in der Übereinstimmung von Verstand und Sache)

(Thomas von Aquin - Quaestiones disputatae de veritate)

In diesem Abschnitt wird dargestellt, wie aus dem beobachteten Systemverhalten und dem erstellten Diagnosemodell zur Laufzeit die Diagnose(n) des betrachteten Systems bestimmt werden.

4.7.1 Erfüllbarkeit und SAT-Solver

Das erstellte formale Fehlermodell aus Abschnitt 4.4, das entweder auf Steuergerät oder Tester gespeichert werden kann (Abschnitt 4.6), wird zur Laufzeit mit den erstellten Beobachtungen erweitert. Durch Auswerten der Beobachtungen werden die Meßwerte mit Wahrheitswerten wahr/falsch belegt. Dabei gilt für eine Beobachtung B_i der Wahrheitswert wahr, falls das mit B_i in Verbindung stehende Verhalten beobachtet werden konnte, also:

$$\mathcal{A} : B_i \mapsto \{true, false\}$$

mit Auswertungsfunktion $\mathcal{A}$ (vgl. [Tri09]). Dabei bildet $\mathcal{A}$ die Variablen direkt auf die Wahrheitswerte ab. Zusätzlich verwendet Tripakis als Ergebnis von $\mathcal{A}$ den Wert ?, falls die Bedingung nicht überprüft werden konnte.

Durch die Belegung der Variablen ergibt sich somit die Frage, für welche der verbleibenden, nicht belegten Variablen die gesamte Formel wahr wird. Diese Fragestellung wird als *boolesches Erfüllbarkeitsproblem* bzw. *boolean satisfiability* bezeichnet und oft mit *SAT* abgekürzt.

SAT ist ein sehr ausgiebig untersuchtes Forschungsthema. Cook zeigte 1971, daß das Erfüllbarkeitsproblem der Aussagenlogik NP-vollständig ist [Coo71]. SAT stellt somit das erste als NP-vollständig erkannte Problem dar. Obwohl deshalb die zu lösenden Erfüllbarkeitsprobleme theoretisch in exponentieller Zeit gelöst werden können, sind sie in der Praxis meist gut lösbar (vgl. [GPFW96]).

Eines der ersten Verfahren für die Lösung von SAT stellt der Algorithmus von Davis und Putnam [DP60] dar, der zwei Jahre später von Davis et al. weiterentwickelt wurde [DLL62] und oft als DPLL oder DLL-Algorithmus bezeichnet wird. Eine Übersicht über weitere mögliche Verfahren findet sich in [GPFW96].

Die meisten heutigen Verfahren basieren jedoch auf dem DPLL-Algorithmus. Aufgrund seiner Relevanz für den weiteren Verlauf dieses Abschnitts wird der DPLL-Algorithmus anhand des folgenden Pseudo-Codes im Detail beschrieben:

bool DLL(Formelmenge $\mathcal{C}$, Belegung $\mathcal{A}'$){
  /∗ Folgere aus der bisher gewählter Belegung die weiteren notwendige Zuweisungen (Unit Clause Rule)∗/
  $\mathcal{A}_{ness}$ = deduce($\mathcal{C}$, $\mathcal{A}$);

```
  A = A' ∪ A_ness; /* füge notwendige Belegung der vorhandenen bisherigen zu */
  if ( satisfiable (C, A))
    return TRUE; /* Formel erfüllbar mit gewählter Belegung */
  }else{
    if ( is_conflict (C, A))
      return FALSE; /* Formel nicht erfüllbar mit gewählter Belegung */
  }
  l = choose_free_literal (C, A); /* wähle freies, noch nicht belegtes Literal l */
  A_1 = A ∪ assign(l, 1) ; /* weise l wahr zu und füge Belegung hinzu */
  result1 = DLL(C, A_1); /* rekursiver Aufruf */
  if (result1 == TRUE)
    return TRUE;
  A_2 = A ∪ assign(l, 0); /* weise l 0 zu, da l = wahr zu Konflikt führte (Conflict Rule) */
  return DLL(C, A_2);
}
```

DPLL-Algorithmus, aus [Zha03, S. 21]

Wie der Algorithmus zeigt, wählt DLL iterativ ein bisher nicht belegtes Literal l, belegt es mit Wahrheitswert wahr und prüft dann rekursiv, ob die Formel mit der bisherigen Belegung und l erfüllbar ist. Dabei verwendet der Algorithmus zwei grundlegende Verfahren, die *Unit Clause Rule* sowie die *Conflicting Rule*.

Die *Unit Clause Rule* besagt, daß falls in einer bislang unerfüllten Klausel alle belegten Literale 0 bzw. falsch ergeben, dann muß das freie Literal l mit 1 bzw. wahr belegt werden, da sonst eine Klausel und somit die gesamte Formel nicht erfüllbar ist (vgl. [DP60]). Die *Conflicting Rule* setzt an diesem Fall an. Falls alle Literale einer Klausel 0 ergeben, so besteht ein Konflikt und für die gewählte Belegung ist die Formel unerfüllbar. Der Algorithmus muß also zurückspringen (*backtracking*) und andere Belegungen für die Literale wählen, sodaß die Formel erfüllbar wird.

Der dargestellte Algorithmus wurde in den letzten Jahren enorm erweitert und verbessert. Dabei steht besonders die Wahl des Literals *choose_free_literal()* im Fokus der Forschung. Diese Wahl ist besonders wichtig, um die Anzahl der Rekursionen zu beeinflussen. Ein Beispiel hierfür ist eine Heuristik für die Wahl des Verzweigungselements, die *Variable State Independent Decaying Sum* (VSIDS) genannt wird. VSIDS stellt einen Häufigkeitszähler für Variablen dar mit einer höheren Gewichtung von Variablen, die für den Suchverlauf aktueller sind [MMZ+01]. Eine genauere Darstellung möglicher Verfahren für die Wahl des Verzweigungselements findet sich in [Zha03, Kap. 2.3.2].

Ein anderer Ansatz für die Laufzeitoptimierung stellt die Folgerung von Belegungen aus den vorhandenen bisherigen dar (*deduce*-Funktion). Werden die noch nicht belegten Variablen iterativ mit dem Wert wahr belegt, so spricht man von *Boolean Constraint Propagation* (BCP). [Zha03, Kap. 2.3.3] zeigt mehrere Verfahren, die für die Deduktion in Frage kommen.

Auch die Wahl des Rückspringens bzw. *backtracking* bei Belegungen, für die sich die Formel nicht erfüllt, bietet Optimierungspotentiale. Hier bieten sich vor allem nicht-chronologische Verfahren an, wie [Zha03, Kap. 2.3.4] zeigt.

Abschließend sei eines der größten Optimierungspotentiale eines SAT-Solvers genannt, das Lernen von Klauseln, die zu Konflikten führen, dem sogenannten *conflict-driven clause learning* (vgl. [SS96] sowie [ES04b, Kap. 4.4]). Befindet sich beispielsweise in der

Belegung eine Klausel $\{x_1, x_2\}$ mit den Literalen x_1, x_2, für die gerade ein Konflikt festgestellt wurde, so wird analysiert, wieso und durch welche andere(n) Klauseln die zum Konflikt führende Belegung propagiert wurde. Der Solver stellt fest, daß der Grund des bestehenden Konflikts x_1 ist und daß x_1 durch die sogenannte *Konfliktklausel* $C_{conf} = (\neg x_3 \vee x_4)$ mit den Literalen $\neg x_3, x_4$ propagiert wurde. Aus diesem Konflikt „lernt" der Solver die negierte Konflikt-Klausel $\neg C_{conf}$ sowie den Grund des Konflikts. Dadurch ergibt sich folgende Belegung für $\mathcal{A} \cup \neg C = \{\neg x_1, x_2, x_3, \neg x_4\}$.

4.7.2 Anpassen des SAT-Solvers

Für den weiteren Verlauf der Arbeit ist die Wahl der Belegung von freien Variablen, wie sie im DPLL-Algorithmus vorgeschlagen wird, nur eingeschränkt einsetzbar. Der Grund hierfür ist, daß bestimmte Variablen mit festen Ergebnissen zu belegen sind, da sie ja die Ergebnisse von vorliegenden Beobachtungen oder Meßergebnissen abbilden.

Eine Möglichkeit besteht darin, die ausgewerteten Beobachtungen als einzelne Klauseln dem auszuwertenden Modell hinzuzufügen.

Eine andere Möglichkeit ist, den möglichen Suchraum des SAT-Solvers durch die Ebenen des Rollenmodells von vorneherein zu begrenzen oder den SAT-Solver soweit anzupassen, daß die Ergebnisse der Beobachtungen als feste Annahmen der Lösungsmethode des SAT-Solvers als Parameter mitgegeben werden (vgl. [ES04b, Kap. 2: Funktion *unit_assumptions()*]) oder indem die Ergebnisse der festgestellten Beobachtungen als gelernte Klauseln verwendet werden.

4.7.3 Effiziente Diagnose im Steuergerät

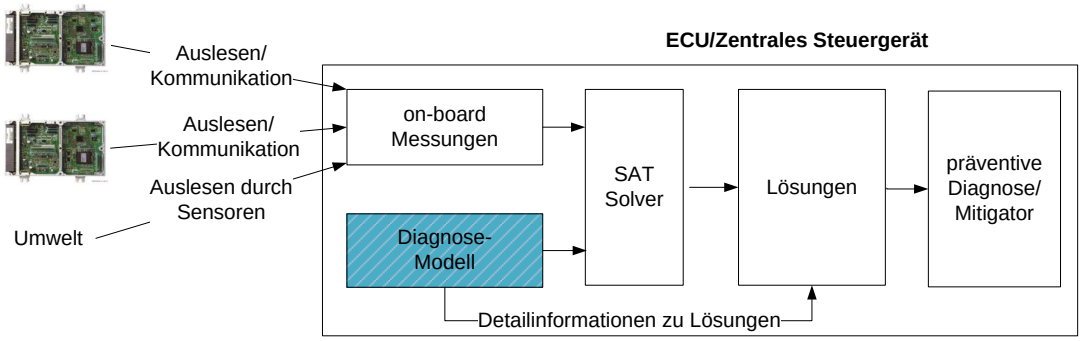


Abbildung 4.12: Effiziente Diagnose zur Laufzeit im Steuergerät.
Quelle: eigene Darstellung

Abbildung 4.12 zeigt den Ablauf der rollenbasierten Diagnose zur Laufzeit im Steuergerät. Das Steuergerät erstellt und wertet interne Messungen aus. Die Beobachtungen entsprechen dabei nur der Rolle Entwicklung. Dazu gehören auch Interaktionen der kommunizierenden Steuergeräte oder erfaßte Umwelteinflüsse durch Sensoren.

Durch die Auswertung der Beobachtungen ergibt sich für die booleschen Variablen des Diagnosemodells eine Belegung mit wahr oder falsch. Da die beobachteten Werte tatsächlich eintrafen, werden die Ergebnisse als einzelne Klauseln entsprechend ihrem

Wert dem Diagnosemodell hinzugefügt. Somit ergibt sich das im vorigen Abschnitt beschriebene Erfüllbarkeitsproblem, das durch den SAT-Solver gelöst wird.

Sind für die bestimmten Fehlerursachen bzw. Lösungen präventive Diagnosemaßnahmen möglich, werden diese anschließend gestartet. Die Information über die möglichen Vermeidungsmaßnahmen erfolgt durch eine Kopplung mit dem gespeicherten Diagnosemodell. Die Vermeidungsmaßnahmen werden von den Entwicklern konzipiert und in einer (erweiterten) FMEA festgehalten. Ziel muß sein, zum Serienanlauf alle präventiven Maßnahmen umgesetzt zu haben. Dennoch können durch die in Abschnitt 4.6 beschriebene Kopplung der erweiterten FMEA an das Diagnosemodell entwickelte Software-Maßnahmen jederzeit in Code umgewandelt werden und durch Flashen auf schon im Feld eingesetzte Komponenten verteilt werden.

Das dynamische Speicherproblem aufgrund Rekursionen

Der DPLL-Algorithmus aus Abschnitt 4.2 und der darauf aufbauende SAT-Solver benutzen Rekursionen zur Bestimmung der Lösungen des SAT-Problems. Sind die Anzahl der Variablen des zu lösenden Problems a priori bekannt, läßt sich der benötigte Speicherplatz des SAT-Solvers und der notwendige Heap berechnen, da die Rekursionstiefe und somit die notwendige Speicherallokation durch die Anzahl der Variablen festgelegt ist. Der Grund hierfür ist, daß im DPLL-Algorithmus ein freies, noch nicht verwendetes Literal für die Rekursion gewählt wird.

Es ist jedoch nicht immer möglich, die genaue Variablenanzahl und somit den benötigten Speicher zu bestimmen. In diesem Fall muß Speicher dynamisch angefordert werden. Dynamische Speicherallokation steht aber im Widerspruch zur MISRA-C-Regel 20.4, „dynamic heap memory allocation shall not be used“ [Mot04]. MISRA (Motor Industry Software Reliability Association) ist ein Programmierstandard in der Automobilindustrie und zielt darauf ab, die Laufzeitsicherheit des erstellten Codes zu erhöhen sowie Qualität und Portabilität des Codes zu verbessern. Die Automobilhersteller schreiben die Einhaltung der MISRA-Richtlinien ihren Lieferanten zwingend vor. Abweichungen von MISRA-Regeln sind nur in bestimmter Form zulässig (vgl. [Mot04, Abschnitt 4.3.2]). Der Verzicht auf dynamische Speicheranforderung soll Speicherfragmentierungen oder -erschöpfungen sowie nicht deterministisches Verhalten der anfordernden Funktionen vermeiden (vgl. [Mot04, S. 83], [Sak08], [Nat09, S. 10]).

Deshalb wird für den SAT-Solver der Diagnose ein festgelegter Speicherbereich zur Verfügung gestellt, der per *memory pool* ([Bro02], [LY03, Kap. 13.3]) verwaltet wird. Der Speicher des Memory pool wird initial allokiert, weshalb dynamische Speicheranforderungen zur Laufzeit nicht notwendig sind. Der Pool besteht aus vielen Speicherblöcken einheitlicher Größe, was eine Fragmentierung und Erschöpfung des Speichers vermeidet. Ist der verfügbare Speicherbereich für den Pool durch den vorhandenen Speicher stark begrenzt, sollten die Blöcke des Pools zyklisch zurückgesetzt werden. Kann für die Diagnose dennoch nicht genügend Speicher angefordert werden, so wird dies per DTC festgehalten und ausreichend Blöcke des Pools zurückgesetzt. Kann nun die Diagnose durchgeführt werden, wird der gesetzte DTC gelöscht. Der Fall, daß die Diagnose des Systems aufgrund Speichermangels oder -problemen über einen längeren Zeitraum nicht durchgeführt werden kann, sollte detaillierter untersucht werden, bspw. mittels der (erweiterten) FMEA.

4.7.4 Effiziente Diagnose in den Werkstätten

Die Diagnose in den Werkstätten läuft ähnlich zu der im Steuergerät ab. Auch hier wird das auf dem Tester gespeicherte Systemmodell durch die Beobachtungen der Rollen Entwicklung, Werkstatt und Kunde erweitert. Dadurch können aber im Gegensatz zum Steuergerät auch mechanische Fehler erfaßt werden, wie Abbildung 4.13 zeigt.

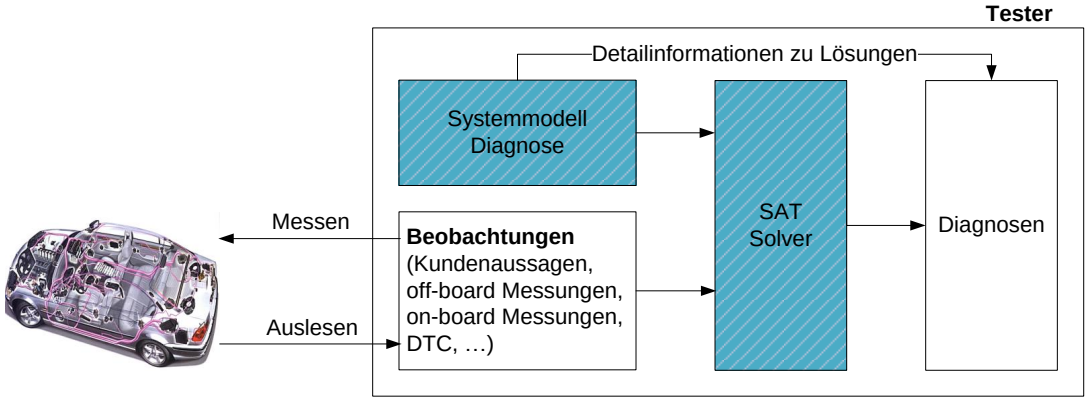


Abbildung 4.13: Effiziente Diagnose zur Laufzeit in der Werkstatt.

Der genaue Ablauf eines Diagnosefalls in der Werkstatt wird in Abbildung 2.11 in Abschnitt 2.3.4 geschildert.

4.8 Wartung der Diagnose im Lebenszyklus

„Einen Fehler machen und sich
nicht bessern:
das erst heißt fehlen“

(Konfuzius - Lùn Yǔ/ Analekten, Buch XV:29)

In den vorigen Abschnitten dieses Kapitels wurde eine Methodik für die Erstellung der Diagnose gezeigt. Vor allem die stark zunehmende Vernetzung der Funktionen führt dazu, daß die Aufgabe der Fehlererkennung, die Unterscheidung zwischen Symptomen eines Fehlers, seiner Fehlerursache und hierzu nicht korrelierenden Daten, enorm schwierig, komplex und umfangreich ist. Berücksichtigt man hierzu noch die in Kapitel 3 aufgezeigten Potentiale, so ist es nicht verwunderlich, daß viele Diagnosefunktionen in nicht optimaler Qualität ins Feld gelangen. So kann beispielsweise eine Diagnosefunktion zum einen nicht zielführende Maßnahmen vorschlagen oder zum anderen Beobachtungen erfassen, die für einen zu entdeckenden Fehler irrelevant oder gar irreführend sind.

Diese Problematik wird gesteigert durch die Pflicht des OEM, die Diagnose des Fahrzeugs und seiner Komponenten bis zu 15 Jahre nach Ende seiner Serienproduktion zu unterstützen (vgl. Abschnitt 2.3.2). Bei einem geschätzten Anteil der Diagnose von 30 - 50% an den automobilen Software-Funktionen (vgl. Abschnitt 1.2) ergibt sich somit ein Umfang von ungefähr 3 Millionen Zeilen Diagnose-Code, der bis zu 15

Jahre gewartet werden muß.

Beide Faktoren zeigen, daß die Wartung dieser umfangreichen Funktionen über einen so ausgedehnten Zeitraum eine enorm schwierige und kostspielige Aufgabe ist. In diesem Abschnitt wird ein effizientes, automatisiertes Verfahren zur Wartung der Diagnose vorgestellt, das auch entwickelte fehlerhafte Diagnosefunktionen einbezieht.

Die Automobilhersteller haben in den letzten Jahren IT-Systeme aufgebaut, die die Daten aller Werkstattfälle an den OEM zurückübertragen (vgl. [Koh06]). Ein wichtiger Grund hierfür war die Etablierung eines geschlossenen Regelkreises für das Qualitätsmanagement [Edl01] durch eine Rückkopplung der Vorfälle im Feld mit den Entwicklungs- und Qualitätsabteilungen. Zum anderen geschieht dies, um Anforderungen an die funktionale Sicherheit zu genügen. Wie schon in Abschnitt 2.3.2 erwähnt, fordert die ISO 26262, daß die Hersteller für sicherheitsrelevante Komponenten einen „Field Monitoring"-Prozeß für die funktionale Sicherheit etablieren müssen [ISO09a, Teil II, Kapitel 7.4.5], der ein Reporting über alle Vorfälle sicherheitsrelevanter Steuergeräte mitsamt der ergriffenen Maßnahmen durch Hersteller und/oder Werkstätten vorschreibt.

Dabei handelt es sich um bis zu 5 Gigabyte an täglichen Daten (vgl. [MPEE10]), die für die Wartung der Diagnose verarbeitet werden müssen. Auf diese mit dem *Librarian Law* beschriebene Problematik, daß je mehr Wissen vorhanden ist, desto mehr Aufwand auf die informationsverarbeitenden Prozesse aufgewendet werden muß ([ER03, Kap. 10.3.5], aus [Deg00]), wurde schon in Abschnitt 3.9 hingewiesen.

Aufgrund dieses enormen Datenumfangs bietet sich ein Einsatz von statistischen Verfahren zur Wartung an. Zudem ermöglicht eine statistische Filterung die von der ISO 26262 vorgeschriebene Untersuchung der gesammelten Daten um bisher unbekannte, sicherheitsrelevante Probleme zu entdecken [ISO09a, Teil VII, Kapitel 6.4.2.1].

In diesem Abschnitt werden multivariate Statistikverfahren eingesetzt, um aus der Gesamtmenge der Diagnosefunktionen die zu entdecken, die gewartet werden müssen. Eine Diagnosefunktion ist dann zu warten, falls sie irrelevante oder fehlerhafte Beobachtungen erfaßt oder falls sie fehlerhafte oder selten erfolgreiche Reparaturmaßnahmen vorschlägt. Zudem entdeckt das vorgestellte Verfahren bisher unbekannte Fehler, also Lücken in der Diagnose. Durch den Tester verursachte Fehler werden jedoch nicht berücksichtigt. Die Wartung erfolgt zentral. Die aktualisierten Diagnosedaten werden anschließend auf alle Testsysteme übertragen.

Das in diesem Abschnitt vorgestellte Verfahren basiert auf [Koh06] und [KKP+11] und ermöglicht eine Steigerung der Effizienz der Diagnosewartung sowie eine deutliche Erhöhung der Qualität der Diagnosefunktionen und vor allem der Reparaturmaßnahmen. Abbildung 4.14 zeigt eine Übersicht der Wartung der Diagnose zur Laufzeit.

4.8.1 Modellierung der Domäne

In diesem Abschnitt wird die Modellierung der Domäne dargelegt. Die Variablen des formalen Fehlermodells aus Abschnitt 4.4 werden mit stochastischen Variablen gekoppelt und die stochastischen Variablen entsprechend deren Belegung gesetzt. Durch diese Kopplung und Kumulation der stochastischen Variablen werden die

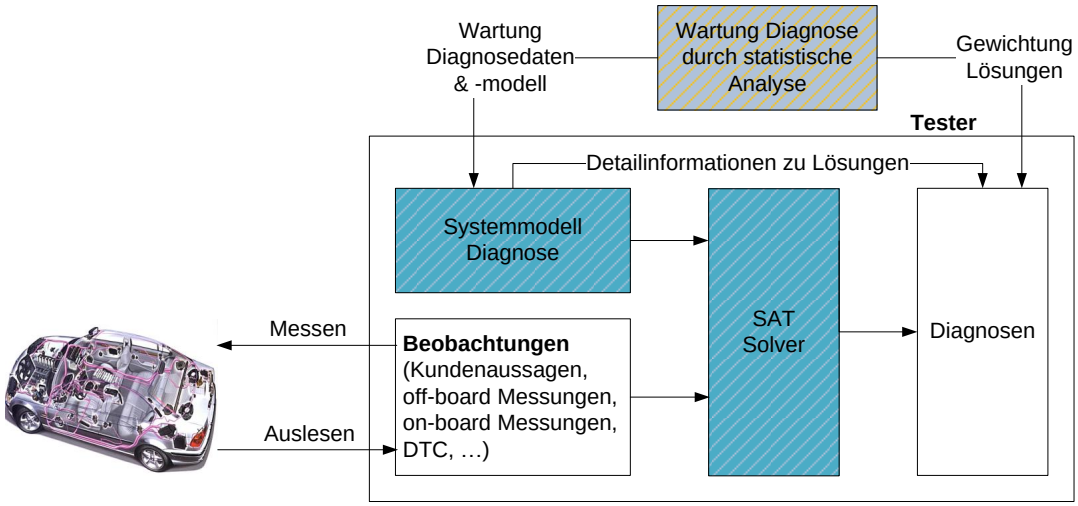


Abbildung 4.14: Wartung der Diagnose im Lebenszyklus.
Quelle: eigene Darstellung, basierend auf Abbildung 4.13

relativen Häufigkeiten gebildet, die für statistische Analysen notwendig sind.

Aufgrund der enormen Datenmenge von ungefähr 5 GB täglich läßt sich so für die übertragenen Daten unter Ausnutzung des *Gesetz der großen Zahlen* (siehe [Ros06, Kap. 8.2 und 8.4] sowie [Loè77, S.14, S.20]) sagen, daß die relativen Häufigkeiten der Daten gegen ihre absoluten Wahrscheinlichkeiten konvergieren und somit die statistischen Ergebnisse aussagekräftig sind.

Als erster Schritt der statistischen Wartung der Diagnose wird ein Modell erstellt, das die Beziehungen zwischen den auswertbaren Daten erfaßt. Dabei handelt es sich um die Daten auf den drei Ebenen des formalen Fehlermodells (vgl. Abschnitt 4.4) sowie den von der Werkstatt erfaßten Gewährleistungsdaten.

Übernommen werden die Variablen $CC_i, G_i, T_i, DTC_i, F_i, RM_i$. Dabei stellen CC_i, G_i, und T_i Variablen dar, die in Beziehung stehen mit Beobachtungen auf den Ebenen des Rollenmodells der Diagnose und den Wert wahr zugewiesen bekommen, falls die Beobachtung eingetreten ist. Die Variable F_i wird mit wahr belegt, falls der mit F_i in Verbindung stehende Fehler als präsent angenommen wird. RM_i erhält den Wert wahr zugewiesen, falls die mit RM_i in Verbindung stehende Reparaturmaßnahme das System von dem mit F_i korrelierenden Fehler heilen konnte. Zusätzlich werden Umgebungsvariablen E_i hinzugefügt, um bisher unbekannte Einflußgrößen zu erfassen. Diese Variablen werden in Abschnitt 4.8.3 genauer dargestellt.

Da sich die Reparatur meist auf Ebene Werkstatt abspielt, ergibt sich somit folgendes Modell für eine off-board Diagnosefunktion D_{off} (vgl. Abschnitt 4.4.3):

$$\begin{array}{lcl} \bigvee_i CC_i & \rightarrow & \bigvee_i F_i \\ \bigvee_i G_i & \rightarrow & \bigvee_i F_i \\ \bigwedge_i DTC_i & \rightarrow & \bigvee_i F_i \\ \bigwedge_i E_i & \rightarrow & \bigvee_i F_i \\ \bigvee_i F_i & \rightarrow & \bigvee_i RM_i \end{array}$$

Das vorgestellte Modell erlaubt die Verbindung technischer Daten wie eines Fehlerspeichereintrags DTC_i oder Werkstattmessungen G_i mit textuellen Daten wie

Kundenbeschwerden CC_i und Reparaturmaßnahmen RM_i.
Es sei an dieser Stelle darauf hingewiesen, daß nicht alle oben aufgeführten Zusammenhänge der Diagnosefunktion D_{off_i} initial bekannt sein müssen, beispielsweise aufgrund transitiver Abhängigkeiten von verteilten Funktionen. Eine Aufgabe des hier vorgestellten Ansatzes ist die Erkennung unbekannter Abhängigkeiten und Fehler.

Im zweiten Schritt werden diskrete stochastische Variablen $\hat{X}_i$ eingeführt und mit den logischen Variablen X_i des Fehlermodells gekoppelt. Einer stochastischen Variable wird der Wert 1 zugewiesen, falls die korrespondierende logische Variable den Wahrheitswert wahr hat und sonst der Wert 0. Es handelt sich bei $\hat{X}_i$ somit um eine *Indikatorvariable* mit folgender Definition:

$$\hat{X}_i = \begin{cases} 1 & \text{falls } X_i = wahr \\ 0 & \text{sonst} \end{cases}$$

Basierend auf dieser Formel läßt sich dann die relative Häufigkeit der Indikatorvariable $\hat{X}_i$ über ihre Auftrittshäufigkeit in den N übertragenen Datensätzen aus den Werkstätten bilden. Aufgrund der enormen Datenmengen läßt sich der Übergang von der relativen Häufigkeit zu der stochastischen Auftretenswahrscheinlichkeit $\Pr$ bilden:

$$\Pr(\hat{X}_i) = \frac{1}{N} \sum_{i=1}^{N} \hat{X}_i$$

4.8.2 Diskussion der verfügbaren statistischen Techniken

Im vorigen Abschnitt wurde der Übergang von den logischen Variablen des Fehlermodells zu stochastischen Variablen gebildet. Dies ermöglicht die statistische Analyse der Daten. Die Wahl der statistischen Analysemethode wird stark durch die speziellen Eigenschaften der automobilen Domäne beeinflußt. In diesem Abschnitt werden die geläufigsten statistischen Analysetechniken kurz auf einen Einsatz diskutiert und dann begründet, wieso in der Arbeit eine multivariate Split-Analyse eingesetzt wird. Die Diskussion ist dabei aus [KKP+11, Kap. IV-A] entnommen.

Die **univariate Analyse** (vgl. [JKK93]) ist eine relativ einfache statistische Analyse. Sie konzentriert sich auf einzelne Variablen, die in Verbindung mit verschiedenen Kategorien gestellt werden können, beispielsweise das Baujahr eines Fahrzeugs. Die univariate Analyse ist nicht für die Domäne anwendbar, da sie nicht in der Lage ist, gleichzeitig mehrere Variablen wie bspw. on- und off-board Meßwerte zu untersuchen. Zudem kann sie nicht Zusammenhänge zwischen vorhandenen Daten erkennen oder berechnen.

Da wie das Diagnosemodell zeigt, verschiedene Werte miteinander in Verbindung stehen können, wie beispielsweise die Meßwerte des Steuergeräts mit einem Fehlerspeichereintrag, müssen im Kontext der Arbeit **multivariate Analysen** (vgl. [TF06]) eingesetzt werden. Bei einer multivariaten Datenanalyse wird eine zu untersuchende Teilmenge innerhalb der gesamten Datenmenge definiert. Dies geschieht durch Beschränkung der Wertebereiche der Datenattribute auf bestimmte Werte oder Intervalle. Beispielhaft ist hierfür eine Teilmenge von Fahrzeugen, die innerhalb ausgewählter Monate mit bestimmten Fehler- oder Beobachtungsmustern in die Werkstatt kamen.

Bei der enormen Datenmenge von ca. 5 GB pro Tag (vgl. [MPEE10]) bedarf es einer Analysemethode, die keine Datenredundanzen für die Auswertung erstellen muß, wie es beispielsweise das **Entscheidungsbaumlernen/ decision tree learning** [BFOS84] muß.

Obwohl in den vorhandenen Daten Zusammenhänge zwischen den Variablen gesucht werden, liegt der Fokus auf vorgegebenen einzelnen Variablen, was den Einsatz einer **Cluster-Analyse** [JD88] ausschließt. Die Cluster-Analyse ist genauso wie **neuronale Netzwerke/ neural networks** [TK08] eine statistische Methode, die auf *unüberwachtes Lernen/ unsupervised learning* setzt. Unüberwachte Lernverfahren erstellen ein statistisches Modell, das die durch den Ansatz gelernten Abhängigkeiten darstellt. Dieser Ansatz ist zwar vorteilhaft bei der Entdeckung unbekannter Zusammenhänge, ist aber in der Zieldomäne nur problematisch einsetzbar.

Die Problematik beim Einsatz dieser Verfahren läßt sich anhand eines weiteren unüberwachten Lernverfahrens verdeutlichen. Die **Assoziationsanalyse** [AIS93] konzentriert sich auf die Entdeckung kurzer Datenmuster mit einer hohen relativen Häufigkeit (Support) wie beispielsweise „Kunden, die A kauften, kauften oft auch B“. Diese Muster sind mit dem von Agrawal et al. vorgestelltem *A-priori-Algorithmus* [AS94] gut entdeckbar. Im vorliegenden Anwendungsgebiet wird jedoch nach deutlich komplexeren Mustern von 20 oder mehr Eigenschaften gesucht, die aus der Anzahl der zu betrachtenden Variablen einer Diagnosefunktion, zusätzlichen Daten über die Fahrzeugkonfiguration und weiteren Informationen bestehen können.

Hinzu kommt erschwerend, daß die für die Wartung relevanten, zu suchenden Datenmuster mit einer geringen Häufigkeit auftreten. Eine Analyse der vorhandenen Daten im Rahmen der Erstellung von [KKP+11] ergab, daß die relevanten Diagnosedaten eine Spreizung von ungefähr 90% und eine Auftretenswahrscheinlichkeit von unter 50% besitzen. Diese Werte ergeben sich aus der enormen Anzahl an voneinander unabhängigen Variablen. Da die meisten statistischen Ansätze für Untersuchungen von Daten konzipiert wurden, deren Auftretenswahrscheinlichkeit höher als 80% und deren Konfidenzwert höher als 60% ist [LHM99], erschwert dies eine statistische Analyse deutlich. Aus der niedrigen relativen Häufigkeit der relevanten Daten ergibt sich die Problematik, daß bei einem unüberwachten statistischen Lernen falsche Zusammenhänge, sogenannte *spurious correlations* gelernt werden. Eine spurious correlation wird auch oft mit dem Satz „correlation is no proof of causation“ [Sim54] beschrieben. Als Beispiel für eine solche Korrelation sei die Untersuchung einer Studie in [Huf94, S. 87ff.] genannt. Die Studie ergab, daß unter den untersuchten College-Studenten in den USA die rauchenden Studenten schlechtere Noten erhielten. Aus den Daten wurde dann das fehlerhafte Studienergebnis geschlossen, daß Rauchen dümmer mache. Die Autoren der Studie rieten dann, daß man das Rauchen aufgeben solle für bessere College-Noten. Huff fügt dem spaßend hinzu, daß es wohl auch sein könne, daß Studenten aufgrund des Frusts durch schlechte Noten rauchen.

Aufgrund der spurious correlations sowie der enormen Größe der zu untersuchenden Variablen müssen folglich spezifische Werte, sogenannte *scored values*, wie beispielsweise bestimmte Beobachtungen, für die Analyse vorgegeben werden. Der Einsatz von scored values entspricht aber einem überwachten Lernen.

Mit einer **split analysis** bzw. **test-control data analysis** [Nik74] lassen sich Hypothesen über die Eigenschaften eines Datensatzes verifizieren. Dabei werden die vorhan-

denen Daten in einen *Test-Datensatz*, der die zu verifizierende bestimmte Eigenschaft aufzeigt, und einen *Control-Datensatz*, der diese Eigenschaft nicht hat, aufgeteilt. Der χ^2-Test wird dann auf die Test-Daten durchgeführt, um zu prüfen, ob die Test-Daten sich signifikant von den Kontrolldaten unterscheiden. Ein Beispiel hierfür ist der Vergleich der durchschnittlichen Gewährleistungskosten eines Fahrzeugs oder die in dem vorgestellten Ansatz verfolgte Bestimmung relevanter Daten einer Reparaturmaßnahme.

4.8.3 Wartung der Diagnose

In diesem Abschnitt wird im Detail gezeigt, wie die Diagnose durch Anwenden der gewählten statistischen Techniken gewartet wird. Es wird dabei vorausgesetzt, daß die zu untersuchenden Daten aus den Werkstätten übertragen und zentral gespeichert werden und die relativen Häufigkeiten der einzelnen Daten berechnet wurden. Abbildung 4.15 zeigt dabei eine Übersicht der einzelnen Schritte, die regelmäßig durchgeführt werden, um die Diagnose zu warten. Die einzelnen Schritte werden im Folgenden genauer dargelegt.

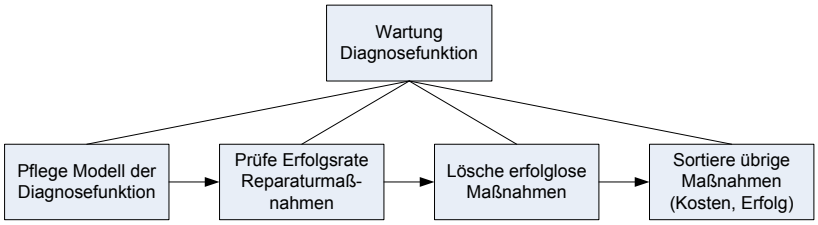


Abbildung 4.15: Ablauf der Diagnosewartung. Quelle: eigene Darstellung

Pflege Modell der Diagnosefunktion

In diesem Abschnitt wird beschrieben, wie das beschreibende Modell einer Diagnosefunktion aus Abschnitt 4.4 gewartet wird. Dabei wird das Modell auf fehlende, hinzuzufügende sowie auf fehlerhafte, zu löschende Elemente untersucht. Für das vorgestellte Verfahren ist die Bestimmung eines repräsentierbaren Datensatzes elementar. Hierzu wird in [KKP+11, Kap. IV.B und IV.C] ein Verfahren vorgestellt, das in diesem Abschnitt skizziert wird.

Die **split analysis** wird eingesetzt, um Beziehungen zwischen den vorliegenden Daten zu entdecken. Dabei handelt es sich um die in Abschnitt 4.8.1 modellierten anfallenden Daten der Rolle Werkstatt (vgl. 4.4.3). Das split analysis-Verfahren sei anhand des Beispiels eines Fehlerspeichereintrags einer beliebigen on-/off-board Diagnosefunktion erklärt. Für jeden Fehlerspeichereintrag DTC_i wird ein Test-Datensatz gebildet, der die Daten aller Reparaturen enthält, bei denen der gewählte DTC_i auftrat. Daraus ergibt sich auch gleich der Kontrolldatensatz, der alle Reparaturen enthält, bei denen der DTC_i *nicht* auftrat.

Findet nun die Split Analyse eine signifikante Korrelation zwischen den Vorkommen eines DTC und einer anderen Variable, beispielsweise einer (wiederholten) Anwendung einer Reparaturmaßnahme RM_i, dann scheint der Fehlerspeicher für die Repa-

raturmaßnahme „relevant“ zu sein. Wird keine signifikante Korrelation festgestellt, dann scheint DTC_i „irrelevant“ für die Reparaturmaßnahme zu sein und sollte aus der Diagnosefunktion dieser Maßnahme gelöscht werden.

[KKP+11] weist auf das Problem hin, daß in der Realität die Test- und Kontrolldaten in vielen Aspekten differieren können. Dies liege an den vielen *Umgebungsvariablen* $E_1, \ldots, E_n$, die teilweise a priori unbekannt sind. Durch das Analyseverfahren werden die Umgebungsvariablen genauer bestimmt und können dann anschließend in das Schichtenmodell der Dissertation eingeordnet werden, abhängig von der Ebene, auf der die Umgebungsvariable beobachtbar ist. Die Umgebungsvariablen können beispielsweise verschiedene Fahrzeugtypen (auf Kundenebene sichtbar), Zusatzausstattungen eines Fahrzeugs (auf Kunden- und Werkstattebene beobachtbar) oder unbekannte funktionale Zusammenhänge (im Steuergerät erfaßbar) sein.

Die Berücksichtigung dieser Variablen ist wichtig, da sie Auswirkungen auf das Verhalten des Fahrzeugs oder gar sein beobachtbares fehlerhaftes Verhalten haben können. Vor allem aber ist die Integration dieser Variablen wichtig, um die erwähnten *spurious correlations* zu vermeiden. So kann es beispielsweise sein, daß die split Analyse die Auswirkung eines DTC_i auf einen bestimmten Fehler F_i entdeckt, aber sowohl DTC_i als auch F_i durch einen dritten Parameter wie beispielsweise Unterspannung verursacht wurden, der sich aufgrund der Vernetzung des Fahrzeugs ausbreitete. In diesem Falle besteht zwischen DTC_i und F_i nur eine nicht-kausale Verbindung. Aufgrund solcher teilweise unbekannten Einflußfaktoren kann auch keine einfache Untersuchung des Lift-Faktors, wie sie die Assoziationsanalyse [AIS93, AS94] ermöglicht, durchgeführt werden. Zudem ergibt sich aus der Notwendigkeit der Einbindung der Umgebungsvariablen die Herausforderung, potentielle Nebeneffekte und nichtkausale Verbindungen, die Einfluß auf die Zieleigenschaft der split analysis haben können, herauszufiltern.

Im Folgenden wird kurz der in [KKP+11, Kap. IV.C] vorgestellte Algorithmus dargelegt, der einen repräsentativen Test-Datensatz erstellt unter Berücksichtigung aller möglichen Einflußfaktoren E_i, die in Verbindung mit den zu untersuchenden Daten des Fehlermodells stehen können. Repräsentativer Testdatensatz heißt dabei, daß die Verteilung von E_i innerhalb Test- und Kontrolldatensatz identisch ist.

Wie oben erwähnt ist der vorhandene Kontrolldatensatz möglicherweise mit großer Wahrscheinlichkeit nicht repräsentativ für die Testdaten. Da es zudem auch unbekannte Zusammenhänge geben kann, die gelernt werden sollen, werden alle Variablen einbezogen. Der Algorithmus bildet für jede Umgebungsvariable E_i die Differenz zwischen seiner Verteilung im Testdatensatz und der im Kontrolldatensatz und summiert die Differenzen aller Umgebungsvariablen. Solange diese Gesamtsumme durch Entfernung eines Datensatzes reduziert werden kann, so wird der Datensatz entfernt, der die Gesamtdifferenz maximal reduziert. Der Algorithmus stoppt, falls die Gesamtdifferenz unterhalb einer bestimmten, signifikanten Grenze fällt oder die Größe der Kontrolldaten eine definierte Größe unterschreitet. Die signifikante Größe läßt sich über einen χ^2-Signifikanztest bestimmen mit der Nullhypothese, daß alle Variablen E_i in den Test- und Kontrolldaten eine identische Verteilung haben. Die Hypothese wird dann bei einem Konfidenzlevel größer als 95% abgelehnt (aus [KKP+11, Kap. IV.C]).

Der vorgestellte Algorithmus hat also zwei Ergebnisse: zum einen wird aus den Daten

Rauschen, wie beispielsweise irrelevante Symptome für ein Fehlerbild, entfernt, zum anderen durch die Analyse aller möglichen Daten neue Abhängigkeiten entdeckt. Diese werden anschließend dem Diagnosemodell hinzugefügt.

Prüfe Erfolgsrate der Reparaturmaßnahmen

In diesem Abschnitt wird die Wartung der Reparaturmaßnahmen beschrieben.

Eine erfolgreiche Reparaturmaßnahme zeichnet sich dadurch aus, daß sie die defekte Komponente oder das defekte Fahrzeug heilt. Durch die Reparatur wird das Fahrzeug also wieder in den Normalzustand versetzt. Die Wiederherstellung der Funktionsfähigkeit wird direkt nach der Reparatur von den Werkstätten geprüft. Bei einer erfolgreichen Anwendung einer Reparaturmaßnahme RM_i wird dann die mit RM_i in Verbindung stehende stochastische Variable $\hat{RM}_i$ um 1 erhöht. Über die Indikatorvariable für eine Reparaturmaßnahme läßt sich dann die statistische Erfolgsrate einer Reparaturmaßnahme RM_i wie folgt bestimmen:

$$\Pr(RM_i) = \frac{\sum \text{Erfolgreiche Anwendungen } RM_i}{\sum \text{Anwendungen von } RM_i}$$

Dennoch kann es sein, daß eine Reparatur nur kurzfristig hilfreich war und der Fehler somit nicht behoben wurde. Hinzu kommt, daß die fehlgeschlagene Reparatur nicht sofort auffallen muß, beispielsweise bei einer Reparatur des Cabrio-Steuergeräts im Winter.

Es wird also geprüft, ob nach einer als erfolgreich gewerteten Reparatur innerhalb einer bestimmten Zeitspanne Δ_t oder einer bestimmten Laufleistung Δ_{km} das Fahrzeug mit den gleichen Beobachtungen und angenommenen Fehlern wieder in eine Werkstatt kommt. Geschieht dies, war die Anwendung aller Reparaturmaßnahmen $RM_1, ..., RM_n$ für den Fehler innerhalb des Zeitraums Δ_t der Reparatur und/oder Laufleistung des Fahrzeugs Δ_{km} nicht erfolgreich.
In diesem Falle werden die mit $RM_1, ..., RM_n$ in Verbindung stehenden stochastischen Variablen verringert und somit die statistische Erfolgsrate der Reparaturmaßnahme $\Pr(RM_i)$ reduziert.

Das heißt im Umkehrschluß, daß eine Reparatur nur dann als erfolgreich gilt, falls ein Fahrzeug nicht innerhalb der beiden definierten Grenzen eine Werkstatt mit denselben Symptomen aufsucht (vgl. [Koh06, Kap. 4.4]). Auch hier leistet der Algorithmus aus dem vorigen Abschnitt wichtige Unterstützung bei der Identifizierung der wirklich relevanten Symptome.

Lösche erfolglose Maßnahmen

Der vorige Wartungsschritt deckt auch mehrfache „trial-and-error"-Reparaturversuche oder Wiederholreparaturen ab. Eine Vermeidung dieser erfolglosen Reparaturen ist ein wichtiger Faktor zur Steigerung der Kundenzufriedenheit bei einem Werkstattfall [MPEE10].

Deshalb wird für jede Reparaturmaßnahme einer Diagnosefunktion geprüft, ob die Erfolgsrate $\Pr(RM_i)$ überhalb eines definierten Wertes $\Pr_{min}$ liegt. Falls sie darunter liegt, wird die Maßnahme aus der Diagnosefunktion gelöscht.

Die Entfernung von Reparaturmaßnahmen gilt nur für Diagnosefunktionen mit mehr als einer Reparaturmaßnahme. Hat eine Diagnosefunktion nur eine Maßnahme, die zudem erfolglos ist, werden die Entwickler der Diagnosefunktion benachteiligt.

Sortiere übrige Maßnahmen

Nachdem die nicht erfolgreichen Reparaturmaßnahmen einer Diagnosefunktion gestrichen wurden, werden die restlichen Diagnosemaßnahmen sortiert. Die Sortierung benutzt dabei die Funktion $\omega(RM_i)$, die die Reparaturmaßnahme RM_i anhand der Erfolgsrate der Reparaturmaßnahme $\Pr(RM_i)$ und den Kosten $cost(RM_i)$ gewichtet (vgl. Abschnitt 4.4.2). Die Gewichtungsfunktion sieht dabei wie folgt aus:

$$\omega(RM_i) = \omega_{Pr} \Pr(RM_i) \cdot \omega_c cost(RM_i)$$

Durch die Verwendung einzelner Faktoren ω_{Pr} und ω_c wird eine wählbare Gewichtung ermöglicht. So ist es möglich, daß beispielsweise kostengünstigere Maßnahmen gegenüber erfolgreichen bevorzugt werden können. Die Gewichtungsfunktion kann zusätzlich erweitert werden durch die Zeitdauer einer Reparaturmaßnahme, die ja auch kostenrelevant ist.

Durch die Gewichtungsfunktion $\omega(RM_i)$ ergibt sich ein Wert, anhand dessen die Reparaturmaßnahmen sortiert werden können. Die Sortierreihenfolge wird dabei als Wert dem Sorting-Element der erweiterten FMEA (vgl. Abbildung 4.7) zugewiesen. Sowohl die Anwendung der Gewichtungsfunktion als auch die Sortierung der Maßnahmen erfolgt zentral und wird anschließend als Update den Testern zur Verfügung gestellt.

Im Ausblick der Arbeit (Abschnitt 8.2) wird kurz ein deutlich umfangreicheres Verfahren zur Bewertung und Gewichtung der Maßnahmen vorgestellt.

4.9 Verwandte Arbeiten

Architekturmodelle Für die Arbeit standen mehrere Architekturmodelle zur Verfügung mit Wild et al. [WFH+06, BFG+08, BGK+09] sowie East-ADL [EAD10].

Für das Themengebiet der Dissertation sind die untersuchten Arbeiten jedoch nur eingeschränkt verwendbar. [BFG+08] verfügt über zu wenige Schichten für eine Betonung des Wiederverwendungsgedankens. In [BGK+09] fehlt ein Datenmodell für die Funktionen und es gibt zudem keine Hinweise, wie das funktionale Verhalten formuliert werden soll. [EAD10] erfaßt das Verhalten des Systems und bietet genügend Schichten für eine sinnvolle Wiederverwendung. [EAD10] erfaßt zudem noch die Interaktionen des Systems mit seiner Umwelt und hat einen starken Fokus auf AUTOSAR [AUT09, VGSS09]; dies zeigt sich darin, daß das Modell der Implementierungsschicht direkt in AUTOSAR eingebunden werden kann.

Dennoch fehlt diesen Ansätzen die Erfassung fehlerhaften Verhaltens, das für die Diagnose essentiell ist, sowie eine Einbindung der für die Diagnose relevanten Rollen, um alle möglichen Fehler abzudecken (vgl. Abschnitt 2.3.3).

Spezifikation von Interaktionen. Für die Spezifikation der Interaktionen der (verteilten) Funktionen bestehen mehrere Ansätze.

EADL definiert Interaktionen in [EAD10, Kap. 4.1] auf der höchsten Abstraktionsebene als Features und in Abschnitt [EAD10, Kap. 6] Interaktionen auf Funktionsebene. Nachteilig ist hier die enge Ausrichtung an AUTOSAR, das zum Zeitpunkt der Erstellung dieser Arbeit nicht in allen Steuergeräten eingesetzt wird.

FOCUS [BS01] stellt ein mathematisch exaktes Verfahren zur Spezifikation von Interaktionen dar, mit dem oder seiner Erweiterung AUTOFOCUS [HSSS96, BHS99] schon mehrere Komponenten spezifiziert wurden (vgl. [FP95, BBSS97, BS02]). Nachteilig ist hier, daß die Beschreibungstechnik für Nicht-Mathematiker nicht unbedingt einfach zugänglich ist.

Multivariate Statistik. Es wurden schon verschiedene statistische Ansätze für eine Analyse der Diagnosedaten eingesetzt, von denen einige hier zusammengefaßt werden. Die Zusammenfassung ist aus [KKP+11, Kap. III] entnommen.

Buddhakulsomsiri und Zakarian [BZ09] setzen Data Mining-Verfahren für die in den Werkstätten erstellten Gewährleistungsdaten ein, aber schließen nicht Daten der on-board Diagnose wie DTC in ihre Analyse ein. Die Gewährleistungsdaten enthalten dabei die von den Werkstätten durchgeführten Reparaturen, Befunde und Kosten. Die Kosten setzen sich dabei aus Materialkosten und Arbeitsaufwand zusammen.

Blumenstock et al. [BSML08] konzentrieren sich hingegen auf Daten der on-board Diagnose, um spezifische Fehlerbilder zu untersuchen, die in Fahrzeugen auftreten. Für die Analyse dieser Fehlerbilder werden jedoch keine Gewährleistungsdaten untersucht.

Müller et al. [MKL+09] verbinden die Diagnosedaten mit den Gewährleistungsdaten, erstellen aber keine Regeln für die Entdeckung von Wiederholreparaturen, wie in Abschnitt 4.8 gezeigt wurde. Der Ansatz zielt darauf ab, signifikante Muster in den Werkstatthandlungen zu entdecken, um so Regeln zu entwickeln, die an andere Werkstätten verteilt werden können. Auch wird in der Arbeit nicht ausreichend auf die Möglichkeit eingegangen, daß eine Reparatur nur kurzfristig erfolgreich ist. Zudem können viele Regeln nur über eine Untersuchung einer ausreichenden Datengrundlage erstellt werden.

Sankavaram et al. [SKMP10] integrieren in ihre Analyse zwar alle verfügbaren Daten der on- und off-board Diagnose inklusive Kundenbeanstandungen und Befunde. Problematisch ist hier jedoch der Einsatz von unüberwachten Lernverfahren. In Abschnitt 4.8.2 wurde dargelegt, daß aufgrund vieler inkorrekter und unbekannter Zusammenhänge zwischen Beobachtungen und Reparaturmaßnahmen falsche Korrelationen zwischen den Daten, *spurious correlations*, gelernt werden können.

4.10 Zusammenfassung

In diesem Kapitel wurde ein Diagnoseansatz für verteilte Funktionen von Automobilsteuergeräten vorgestellt. Der Ansatz basiert auf den im vorigen Kapitel erarbeiteten Potentialen der Diagnose.

Der vorgestellte Diagnoseansatz unterteilt sich dabei in ein Verfahren zur formalen Spezifikation und Erstellung von modellbasierten Diagnosefunktionen für die on- und off-board Diagnose sowie ein statistisches Verfahren zur Analyse der erstellten Diagnosefunktionen auf Fehler und Verbesserungen.

Der Diagnoseansatz orientiert sich dabei an den Rollen eines Anwendungsfalls Diagnose. Dies ist nötig, um alle möglichen Fehler in der Automobildomäne abdecken zu können. Es wurde zudem das bekannte Analyseverfahren FMEA erweitert für eine strukturierte Erfassung aller systematischen Fehler der Domäne.

Die erstellten Diagnosefunktionen können sowohl on-board im Steuergerät als auch off-board in einem Testsystem für die Werkstätten gespeichert werden.

Um den möglichen enormen Zustandsraum bei verteilten Software-Systemen zur Laufzeit effizient auf vorhandene Fehlverhalten durchsuchen zu können, wird für die Inferenz zur Laufzeit ein SAT-Solver eingesetzt.

Weiterhin wurde ein statistisches Verfahren zur Analyse der Diagnose vorgestellt. Die Diagnosefunktionen werden sowohl auf fehlerhafte Funktionen, die nicht zu einer erfolgreichen Reparatur führen bzw. nicht relevante Beobachtungen erfassen, als auch auf optimierbare Reparaturmaßnahmen untersucht. Das Verfahren ermöglicht somit eine automatisierte Wartung der Diagnose.

Der vorgestellte Ansatz wird im nächsten Kapitel in einen Prozeß eingebunden. Als Datenschnittstelle dient die vorgestellte erweiterte FMEA. Der Ansatz wird in Kapitel 6 anhand von Fallbeispielen evaluiert.

Kapitel 5

Effizienter Diagnoseprozeß über den Fahrzeuglebenszyklus

„If you can't describe what you are doing as a process, you don't know what you're doing."

(W. Edwards Deming)

In diesem Abschnitt wird gezeigt, wie die im vorigen Kapitel vorgestellte Methodik in einen Vorschlag eines Diagnoseprozesses eingebettet wird. Der vorgestellte Diagnoseprozeß wird zur übersichtlicheren Darstellung in einen Entwicklungsprozeß sowie einen Wartungsprozeß über den gesamten Lebenszyklus aufgegliedert. In diesem Kapitel werden die beiden Prozesse beschrieben, eine detaillierte Darstellung der einzelnen Schritte findet sich in Anhang A.3.

Ziel der vorgeschlagenen Diagnoseprozesse ist die Gleichstellung der Diagnoseentwicklung mit der Entwicklung der System- und Kundenfunktionen im Fahrzeugentstehungsprozeß. Um diesem Ziel gerecht zu werden, wird der Diagnoseprozeß in den Entwicklungsprozeß eingebettet.

Übersicht

5.1 Neuer Entwicklungsprozeß Diagnose bis Serienanlauf

„The past is a foreign country:
they do things differently there“

(L. P. Hartley - The Go-Between)

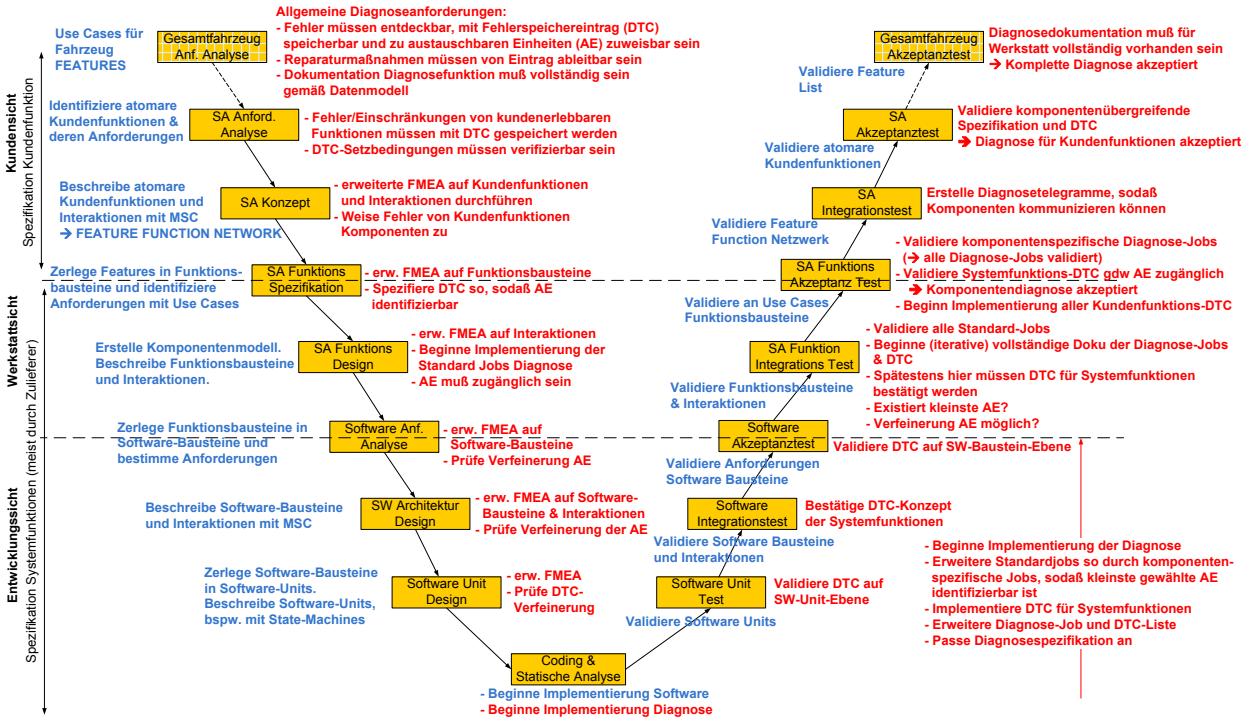


Abbildung 5.1: Vorschlag Entwicklungsprozeß für die Diagnose verteilter Funktionen. Quelle: eigene Darstellung, basierend auf [BMW06] und [BMW07]

Abbildung 5.1 zeigt die einzelnen Schritte des vorgeschlagenen modellbasierten Prozesses der Arbeit für die Diagnoseentwicklung. Dabei sind die diagnoserelevanten Schritte in rot gehalten, die Schritte für die Nominalfunktionen in blau. Das vorgestellte Prozeßmodell ergibt sich aus der Einbettung des in Abschnitt 4 vorgestellten Diagnoseansatzes in den bisherigen, generischen Diagnoseprozeß in Abschnitt 2.3.5, der wiederum auf [Koo92] sowie [BMW06] basiert. Unterschiede zum bisherigen Diagnoseprozeß bestehen dabei in der formalen Spezifizierung der Diagnose auf den drei Abstraktionsebenen des Rollenmodells sowie der Verwendung einer erweiterten FMEA als Datenschnittstelle zwischen Zulieferer und OEM. Zudem ermöglicht die formale Spezifikation der Diagnose mitsamt den gestellten Anforderungen eine Validierung der Diagnose im Verlauf der Entwicklung.

Eine genaue Darstellung der einzelnen Schritte findet sich im Anhang A.3. Zudem wird der Entwicklungsprozeß in Abschnitt 6.2 anhand eines Fallbeispiels veranschaulicht.

Der vorgestellte Prozeß beginnt in der Phase **Gesamtfahrzeuganforderungsanalyse**. In dieser Phase werden die Funktionalitäten des späteren Fahrzeugprodukts bestimmt. Das Verhalten der Funktionalitäten wird mit *Use Cases* (vgl. [Rum05, Kap. 6 sowie S.

668ff.]) erfaßt. In dieser Phase werden auch die allgemeinen Diagnoseanforderungen aus Tabelle 2.2 als Eingangsgrößen der Diagnose festgelegt.

In der **Systemarchitekturanforderungsanalyse** werden aus den erstellten Use Cases die Kundenfunktionen extrahiert. Diese werden verfeinert bis hin zur *atomaren Kundenfunktion*, die die kleinste für den Kunden erlebbare Funktion darstellt. Die Diagnoseanforderungen werden auf dieser Ebene um die Anforderung erweitert, daß kundenerlebbare Fehler oder fehlerhaftes Verhalten einer Kundenfunktion mit einem Fehlerspeichereintrag (DTC) gesichert werden müssen. Dies gilt auch für Einschränkungen der Kundenfunktionen, die durch die spätere präventive Diagnose entstehen können. Zudem müssen die Setzbedingungen der Fehlerspeicher verifizierbar sein.

Die Beschreibung der Interaktionen der atomaren Kundenfunktionen findet in der Phase **Systemarchitekturkonzept** statt. Die Spezifikation des Verhaltens der atomaren Kundenfunktionen kann hierbei mit MSC erfolgen (vgl. [MSC04] sowie [Krü00]). In diesen Phasen wird zum ersten Mal auf die Möglichkeit der Verteilung der Funktionen eingegangen. Durch die Beschreibung der gegenseitigen Abhängigkeiten der atomaren Kundenfunktionen ergibt sich das *Feature Function Network / Funktionsnetzwerk*, einer ersten Abbildung des späteren Bordnetzes. In dieser Phase muß auch die Möglichkeit einer Erweiterung des Fahrzeugs durch verschiedene Systemausstattungen sowie im Rahmen einer späteren Modellüberarbeitung vorgehalten werden. Auf das Funktionsnetzwerk wird nun die erweiterte FMEA (vgl. Abschnitt 4.3.6, Datenmodell in Abschnitt 4.5.2) durchgeführt. Dadurch werden die möglichen Fehler sowohl der Funktionen als auch der Interaktionen systematisch erfaßt.

In der nächsten Phase, der **Systemarchitekturfunktionsspezifikation**, werden die Kundenfunktionen in *Funktionsbausteine* zerlegt und die Anforderungen der jeweiligen Funktionsbausteine mit Use Cases bestimmt. In dieser Phase findet der Übergang von der Kundensicht zur Werkstattsicht dar. Auch auf die spezifizierten Anforderungen der Funktionsbausteine wird die erweiterte FMEA durchgeführt. Zudem werden ab dieser Ebene die Fehlerspeicher *austauschbaren Einheiten* zugewiesen, die für die Werkstatt zugänglich sein müssen. Dennoch sind die Fehlerspeicher auf dieser Ebene noch relativ groben Einheiten zugewiesen und können deshalb später verfeinert werden.

Das Komponentenmodell stellt einen ersten Schritt Richtung Deployment-Modell (vgl. Abschnitt 4.6) dar und wird beginnend mit der Phase **Systemarchitekturfunktions-Design** erarbeitet. In dieser Phase werden zudem auch die Funktionsbausteine und ihre Interaktionen formal beschrieben. Auf die spezifizierten Funktionsbausteine und ihre Interaktionen wird die erweiterte FMEA durchgeführt. Ab dieser Phase werden die grundlegenden Diagnose-Funktionen, die sogenannten *Standard-Jobs Diagnose* (vgl. [BMW08]), erstellt. Dabei handelt es sich um grundlegende Diagnosefunktionen, die jedes Steuergerät verstehen muß. Das betrifft die Kommunikation mit dem Tester über das UDS-Protokoll [ISO06] sowie Funktionen wie *Fehlerspeicher_löschen()* oder *Steuergerät_identifizieren()*. Es wird zudem nochmals geprüft, ob die identifizierten Fehler der Funktionsbausteine und vor allem die ihrer Interaktionen einer austauschbaren Einheit zuweisbar sind und daß diese Einheit auch für die Werkstatt zugänglich ist.

In der Phase **Software-Anforderungsanalyse** findet der Übergang von der Werkstattsicht zur Entwicklungssicht statt. Die Funktionsbausteine werden in *Software-Bausteine*

zerlegt und ihre Anforderungen aus den vorigen abgeleitet und genauer bestimmt. Die Anforderungen der Software-Bausteine werden durch die erweiterte FMEA erweitert. Zudem wird geprüft, ob die bisher gewählte austauschbare Einheit verfeinerbar ist. Dadurch entsteht zwar ein Mehraufwand durch die notwendige detailliertere Diagnose, dafür läßt sich im Fehlerfall eine kleinere Einheit der bisherigen austauschbaren Einheit identifizieren. Natürlich gilt auch für die verfeinerte Einheit, daß sie für die Werkstatt zugänglich sein muß.

Die Software-Bausteine und ihre Interaktionen werden in der Phase **Software Architektur Design** spezifiziert. Auf diese Spezifikation wird die erweiterte FMEA durchgeführt.

Die Software-Bausteine werden nun in der Phase **Software Unit Design** in *Software Units* verfeinert. Die Software Units stellen die unterste Ebene des Funktionsmodells dar und lassen sich beispielsweise mit State Machines (vgl. [Rum05, Kap. 7 sowie S. 604ff]) beschreiben. Auch auf diese Software Units wird die erweiterte FMEA durchgeführt.

Die Implementierung startet mit der Phase **Codierung und statische Analyse**. Es werden sowohl die Funktionsumfänge als auch die Diagnoseumfänge implementiert, die zur Erhöhung der Programmierqualität dem MISRA-Standard [Mot04] genügen müssen. Die Programmierung der Diagnoseumfänge unterteilt sich in on- und off-board Diagnose. Dabei unterscheidet sich die on-board Diagnose in die erwähnten Standarddiagnosefunktionen aller Steuergeräte sowie komponentenspezifische Diagnosefunktionen. Beginnend mit dieser Phase werden die komponentenspezifischen Diagnosefunktionen entwickelt. Die komponentenspezifischen Funktionen enthalten dabei sowohl die Fehlerdetektion (Abschnitt 4.4) als auch die präventive Diagnose sowie die Mitigatoren, die Fehlerspeichereinträge setzen können. Bei den DTC wird zwischen *Systemfunktions-* und *komponentenübergreifenden DTC* unterschieden. *Systemfunktions*-DTC werden eingesetzt, um interne Fehler des Steuergeräts, beispielsweise auf Bauteilebene, festzuhalten. *Komponentenübergreifende* DTC beschreiben dabei Fehler von Funktionen, die sich auf technisch höherer Ebene abspielen und auch andere Steuergeräte betreffen können. Dennoch gilt für alle DTC, daß sie so konzipiert sein müssen, sodaß die kleinste gewählte austauschbare Einheit identifizierbar ist. Da Komponenten oft von Zulieferern entwickelt werden, müssen die Diagnoseumfänge dokumentiert werden, sodaß der OEM dies nachvollziehen kann. Zum einen aufgrund der Sicherheitsrelevanz der Diagnose sowie Auswirkungen eines Komponentenfehlers auf das Gesamtsystem, zum anderen da die on-board Diagnose mit der off-board Diagnose für die Werkstätten kooperieren muß, die meist vom OEM beginnend in dieser Phase erstellt wird.

Mit dem **Software Unit Test** beginnt die Integrations- und Validierungsphase. In dieser Phase werden sowohl die Spezifikation der Software Units als auch die Diagnosespezifikation, insbesondere die DTC auf Ebene Software Unit, validiert.

In der Phase **Software Integrationstest** wird die Architektur der Software-Bausteine mitsamt ihren Interaktionen validiert. Zusätzlich wird auf dieser Ebene das Fehlerspeicherkonzept der Systemfunktionsfehlerspeicher bestätigt.

Mit dem **Software Akzeptanztest** werden die Anforderungen der Software-Bausteine validiert. Zusätzlich werden auf dieser Ebene die auf der Ebene Software-Bausteine gewählten DTC validiert.

In der Phase **Systemarchitekturfunktionsintegrationstest** werden die Funktionsbausteine und ihre Interaktionen validiert. Seitens der Diagnose wird durch die Validierung aller Standarddiagnosefunktionen für sowohl die on- als auch off-board Diagnose das Grundgerüst der Diagnose abgeschlossen und freigegeben. Zudem müssen spätestens in dieser Phase die DTC der Systemfunktionen bestätigt werden, da ab der folgenden Phase die komponentenübergreifenden Funktionen stärker in den Fokus geraten.

Die durch die Use Cases erstellten Anforderungen der Funktionsbausteine werden in der Phase **Systemarchitekturfunktionsakzeptanztest** validiert. Hinsichtlich der Diagnose werden in dieser Phase alle DTC der Systemfunktionen genau dann validiert, falls die durch den DTC identifizierbare austauschbare Einheit für die Werkstatt zugänglich ist. Zusätzlich werden die *komponentenspezifischen Diagnosefunktionen* validiert. Dadurch ist die gesamte Komponentendiagnose validiert und die Implementierung der *komponentenübergreifenden* bzw. *Kundenfunktions-DTC* kann beginnen.

In der Phase **Systemarchitekturintegrationstest** wird das spezifizierte Funktionsnetzwerk validiert. Zusätzlich werden in dieser Phase zwischen den Zulieferern und dem OEM die Diagnosetelegramme abgestimmt, sodaß die verteilten Funktionen bezüglich der gegenseitigen Überwachung und Fehlervermeidung miteinander kommunizieren können.

Die atomaren Kundenfunktionen werden in der Phase **Systemarchitekturakzeptanztest** validiert. Dadurch sind alle Kundenfunktionen des Fahrzeugs freigegeben. Indem die beiden Anforderungen validiert werden, daß alle Fehler oder Einschränkungen von Kundenfunktionen mit DTC gesichert und die Setzbedingungen der DTC verifizierbar sind, ist die gesamte Diagnose der Kundenfunktionen validiert.

Mit der Phase **Gesamtfahrzeugakzeptanztest** werden die Features des Gesamtfahrzeugs validiert. Dadurch kann das Fahrzeug zur Produktion und zum anschließenden Verkauf freigegeben werden. Die Diagnoseentwicklung wird genau dann abgeschlossen, falls die generellen Diagnoseziele erfüllt sind und die Dokumentation der Diagnose vollständig für die Werkstätten ist.

5.2 Lebenszyklusprozeß Diagnose

Der Lebenszyklusprozeß der Diagnose wird in zwei Unterprozesse unterteilt.

Der erste Prozeß stellt dabei die **Fahrzeugmodellüberarbeitung** dar, deren Entwicklung ungefähr 3 Jahre nach Produktionsstart des Originalfahrzeugs beginnt und ungefähr zwei Jahre dauert (vgl. Abbildung 2.1). Die Modellüberarbeitung wird auch oft als *Facelift* bezeichnet, da unter anderem das Fahrzeugdesign geändert wird. Zusätzlich werden aber auch Verbesserungen wie neue Versionen der Originalkomponenten ins Fahrzeug eingebaut. Hier bietet sich der bereits erwähnte Vorteil, aus den bisherigen Erfahrungen des Feldes mit der Komponente zu lernen und so sowohl die Qualität der Funktionen als auch der Diagnose zu erhöhen.

Der zweite Prozeß ist der **kontinuierliche Verbesserungsprozeß** der Diagnose einer Komponente, beginnend mit dem Ersteinsatz der Komponente. In diesem Prozeß wird anhand der Feedback-Daten aus dem Feld kontinuierlich mittels statistischer Ver-

fahren geprüft, ob die Diagnosefunktionen für eine Komponente sowie die komponentenübergreifenden Funktionen funktionieren. Festgestellte Mängel werden verbessert sowie Optimierungspotentiale gehoben.

5.2.1 Diagnoseprozeß für die Fahrzeugmodellüberarbeitung

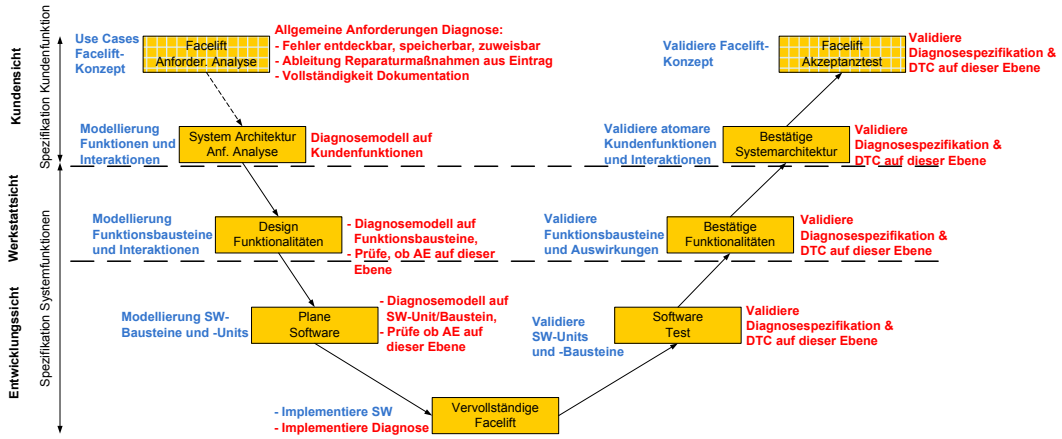


Abbildung 5.2: Vorschlag Lebenszyklusprozeß für die Diagnose verteilter Funktionen. Quelle: eigene Darstellung, basierend auf [Koo92, BMW06]

Abbildung 5.2 zeigt eine Übersicht des vorgeschlagenen Lebenszyklusprozeß der Diagnose, der auf [Koo92] sowie [BMW06] basiert. Diagnoserelevante Schritte sind in rot gehalten, die Entwicklungsschritte für die Funktionalumfänge in blau. Eine detaillierte Darstellung der einzelnen Prozeßschritte findet sich im Anhang A.4. Wie die Abbildung zeigt, findet dieser Prozeß auf der Systemarchitektur- und Komponentenebene statt.

In der Phase **Facelift Anforderungsanalyse** werden *Use Cases* (vgl. [Rum05, Kap. 6 sowie S. 668ff.]) zur Definition des Funktionsumfangs des Facelift erstellt. Dabei wird zum einen festgelegt, welche vorhandenen Funktionalitäten von dem Originalfahrzeug oder anderen Fahrzeugen übernommen werden sollen (*Synergie-* oder *Gleichteile*) und zum anderen, welche *Features* neu entwickelt werden sollen. Für alle Features gelten die allgemeinen Diagnoseanforderungen aus Tabelle 2.2.

Die nächste Phase, die **Systemarchitekturanforderungsanalyse**, beginnt mit der Modellierung der neuen Kundenfunktionen und ihren Interaktionen und untersucht gleichzeitig, ob die neuen *atomaren Kundenfunktionen* Auswirkungen auf bestehende, übernommene Kundenfunktionen haben. Falls ja, werden die Spezifikationen der vorhandenen Kundenfunktionen angepaßt. Zusätzlich wird auf die neuen Kundenfunktionen die erweiterte FMEA durchgeführt und die vorhandenen Diagnosedaten der geänderten Kundenfunktionen soweit nötig angepaßt.

Die Kundenfunktionen werden in der Phase **Design Funktionalitäten** verfeinert in die *Funktionsbausteine*. Die Funktionsbausteine und ihre Interaktionen werden beispielsweise mittels des MSC-Ansatzes spezifiziert (vgl. [Krü00, MSC04]). Auch hier wird geprüft, ob die Funktionsbausteine der neuen Funktionen Auswirkungen auf vorhandene Bausteine haben und falls ja, werden die Spezifikationen der übernom-

menen Funktionen überarbeitet. Auf alle neuen und geänderten Bausteine wird die erweiterte FMEA durchgeführt. Weiterhin wird geprüft, ob die zu bestimmende *austauschbare Einheit* für die Werkstätten auf dieser Ebene sein soll.

In der Phase **Plane Software** werden die Funktionsbausteine in *Software-Bausteine* und *Software Units* verfeinert. Auch hier werden wieder die Bausteine und Units sowie ihre jeweiligen Interaktionen formal spezifiziert und geprüft, welche der übernommenen Bausteine oder Units angepaßt werden müssen. Die erweiterte FMEA wird auf alle neuen und geänderten Bausteine und Units sowie deren Interaktionen durchgeführt. Zudem wird untersucht, ob die in der letzten Phase definierten austauschbaren Einheiten verfeinert werden sollen.

Die Phase **Vervollständige Facelift** bezeichnet die Implementierungsphase, in der sowohl die Diagnose als auch die restlichen Funktionalitäten implementiert werden.

In der **Software Test**-Phase werden die Software Units und -Bausteine gegen ihre Spezifikation validiert. Das gleiche gilt für die Diagnoseumfänge auf dieser Ebene. Zusätzlich wird geprüft, ob die gewählten austauschbaren Einheiten auf den beiden Ebenen identifizierbar durch DTC sind.

Die nächste Phase wird als **Bestätige Funktionalitäten** bezeichnet. Die Funktionsbausteine und Interaktionen werden hier gegen ihre Spezifikationen geprüft. Die Diagnose wird auch gegen ihre Spezifikation validiert und zudem wird geprüft, ob alle DTC auf austauschbare Einheiten verweisen und diese Einheiten zugänglich für die Werkstätten sind.

In der Phase **Bestätige Systemarchitektur** erfolgt die Validierung der atomaren Kundenfunktionen und Interaktionen, ebenso für die Diagnosespezifikation dieser Umfänge. Weiterhin wird geprüft, ob alle Fehler oder Einschränkungen einer kundenrelevanten Funktion mittels DTC festgehalten werden und die DTC funktionsfähig sind.

Das gesamte Facelift-Konzept wird abschließend in der Phase **Facelift Akzeptanztest** zur Produktion freigegeben, falls die Anforderungen der Features erfolgreich validiert werden konnten. Die Diagnose wird dann vollständig akzeptiert, falls die allgemeinen Anforderungen der Diagnose erfüllt sind.

5.2.2 Kontinuierlicher Diagnoseverbesserungsprozeß

„To strive, to seek, to find,
and not to yield“

(Alfred Tennyson - Ulysses)

In diesem Abschnitt wird der kontinuierliche Verbesserungsprozeß der Diagnose dargestellt. Dieser Prozeß nutzt die Datenrückführung von Lebenszyklusdaten des Fahrzeugs für die kontinuierliche Überprüfung der vorhandenen Diagnosefunktionen. Dabei werden zum einen unbekannte Fehler gefunden, die folglich Lücken in der vorhandenen Diagnose darstellen und zum anderen die vorhandenen Diagnosefunktionen optimiert. Der Prozeß setzt dabei den in [KKP+11] vorgestellten und in Abschnitt 4.8 erweiterten Ansatz ein.

Abbildung 5.3 stellt eine Übersicht des kontinuierlichen Verbesserungsprozeß mitsamt den ergriffenen Maßnahmen dar. Die gelb markierten Kästen stellen dabei den

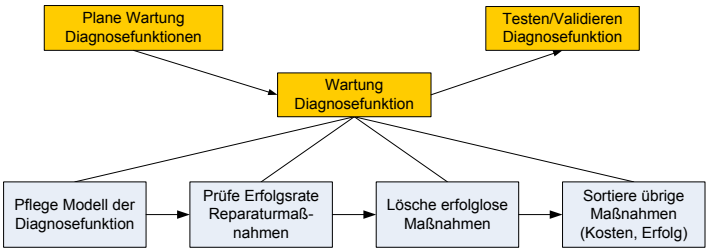


Abbildung 5.3: Kontinuierlicher Verbesserungsprozeß Diagnose mit Maßnahmen. Quelle: basierend auf [BMW07, Kap. 5.2], [KKP+11] und Abb. 4.14

Modifizierungsprozeß aus [BMW07] dar, die türkisen Kästen die in Abschnitt 4.8 dargestellte Wartungsmethodik.

In der ersten Phase erfolgt die **Planung der Wartung**. Die einzelnen Feedback-Daten werden zwar in verschiedenen Zeitabständen, aber dennoch regelmäßig an den OEM übertragen. Hierbei ist zu berücksichtigen, daß die Reparaturdaten natürlich nur nach Reparaturen anfallen und somit saisonalen Effekten, bspw. bei Reparaturen von Cabrios, unterliegen können. Aufgrund des langen Wartungszeitraums von bis zu 15 Jahren nach Produktionsende eines Fahrzeugtyps fallen jedoch genügend Daten zur Bildung der statistischen Daten an, auch für saisonal-bedingte Ausfälle.

Die **Durchführung der Wartung** geschieht durch eine statistische Auswertung der gesammelten Daten. Dabei wird eine *multivariate split analysis* auf den Daten ausgeführt. Die Diagnosefunktionen werden zuerst darauf untersucht, ob sie die richtigen Beobachtungen erstellen. Falsche Beobachtungen werden aus der Funktion gestrichen, fehlende hinzugefügt. Anschließend wird geprüft, ob die vorgeschlagenen Reparaturmaßnahmen erfolgreich sind und erfolglose Maßnahmen gestrichen. Die verbleibenden Maßnahmen können dann anhand Kosten oder Erfolgsrate sortiert werden.

Da die Diagnosefunktionen geändert wurden, erfolgt anschließend das **Testen/ Validieren der Diagnosefunktionen**.

Kapitel 6

Fallstudien

In diesem Abschnitt werden anhand ausgewählter Umfänge von Fallbeispielen der vorgestellte Ansatz sowie der vorgeschlagene Prozeß der Dissertation evaluiert.

Der Fensterheber ist ein gern verwendetes Fallbeispiel (vgl. bspw. [Str06, SWLL06, JRT08, VLG10]), da er ein relativ verständliches, leicht erklärbares System darstellt. Dennoch handelt es sich beim Fensterheber um ein verteiltes System, das über zwei verschiedene Bussysteme kommuniziert und zudem über sicherheitsrelevante Umfänge verfügt. Die vorgestellte Methodik wird anhand ausgewählter Fehler des Fensterhebers beispielhaft dargestellt.

Anschließend wird am Beispiel eines ausgewählten Fehlers des Fensterhebers der Entwicklungsprozeß Diagnose, wie er in der Arbeit vorgeschlagen wird, gezeigt.

Eine Darstellung des Wartungsprozesses der Diagnose anhand von Reparaturmaßnahmen für das ACC schließt das Kapitel ab.

Übersicht

6.1 Fallstudie Fensterheber

6.1.1 Beschreibung des Fensterhebers

Die Anwendung des Rollenmodells der Diagnose auf einen Fensterheber wurde schon in [KB10] dargestellt, aus dem große Teile dieses Fallbeispiels entnommen sind. Das Fallbeispiel in der Dissertation wird jedoch um die Erkennung zweier durch Software verursachte Fehler erweitert.

Der Fensterheber im Automobil ist ein verteiltes System, dessen in Software realisierte Funktionen über verschiedene Busprotokolle kommunizieren. Die Hauptaufgaben des Fensterhebers sind das *stufenweise* bzw. manuelle sowie *automatische Öffnen und Schließen* des Fensters. Automatisches Fensteröffnen oder -schließen bedeutet, daß das Fenster bis zu seinem unteren oder oberen Anschlag fährt, solange es nicht durch eine Fensterbewegung in die andere Richtung gestoppt wird.

Der Fensterheber wird in allen Derivaten einer Produktlinie eingebaut und muß deshalb spezielle Funktionen bereitstellen, die für die Derivate benötigt werden. Ein Beispiel hierfür ist die Funktion *Kurzhub Öffnen* bei rahmenlosen Fenstern, die beispielsweise in allen Cabrios oder Coupés eingebaut werden. Bei diesen rahmenlosen Türen arretiert das Fenster direkt in der Dichtung des Daches. Um nun die Tür eines geschlossenen Cabrios öffnen zu können, muß das Fenster um eine kleine, bestimmbare Grenze gesenkt werden, sobald die Anforderung des Türöffnens entdeckt wird. Ebenso gibt es die Funktion *Kurzhub Schließen*, die das Fenster beim Abschließen des Fahrzeugs kurz anhebt, so daß das Fenster in der Dachdichtung arretiert, um Einbrüche oder bspw. das Eintreten von Wasser bei starkem Regen zu verhindern. Eine bekanntere Funktion ist das Anwenden der automatischen Fenster öffnen oder schließen Funktion, falls der Fahrzeugschlüssel länger als eine definierte Zeitspanne gedrückt wird (vgl. Abbildung 4.5). Diese Funktion wird auch *Komfortöffnen* bzw. *Komfortschließen des Fahrzeugs* genannt.

Der Funktion *automatisches Schließen Fenster* ist besondere Aufmerksamkeit zu widmen, da beim Schließen des Fensters Gegenstände eingeklemmt werden können. Das Erkennen des Einklemmens von Gegenständen ist aufgrund einer möglichen Gefährdung für Insassen sicherheitsrelevant (ASIL-Stufe A, vgl. [ISO09a]). Diese *Einklemmschutz* genannte Funktion kann jedoch in Notfallsituationen vom Fahrer überstimmt werden, beispielsweise im Falle eines Überfalls auf den Fahrer. Deshalb wird dieses Überstimmen als *Panic Mode* bezeichnet.

Die Bewegungsfunktionen des Fensters können durch die Fensterhebersteuerung deaktiviert werden, beispielsweise falls ein Überhitzen des Fensterhebermotors entdeckt oder falls erkannt wird, daß ein Einklemmen aufgrund fehlerhafter Hardware oder Spannung nicht mehr erkannt werden kann.

6.1.2 Technische Architektur Fensterheber

Abbildung 6.1 zeigt ein abstraktes Modell des Fensterhebers in einem Fahrzeug mit vier Fenstern. Der Fahrer ist in der Lage, alle Fenster des Fahrzeugs zu kontrollieren. Hierzu steht ihm im *Fensterblock Fahrer* für jedes Fenster ein *Fensterschalter* zur Verfügung. Die Schalter übertragen die Befehle des Fahrers über analoge Kabel oder

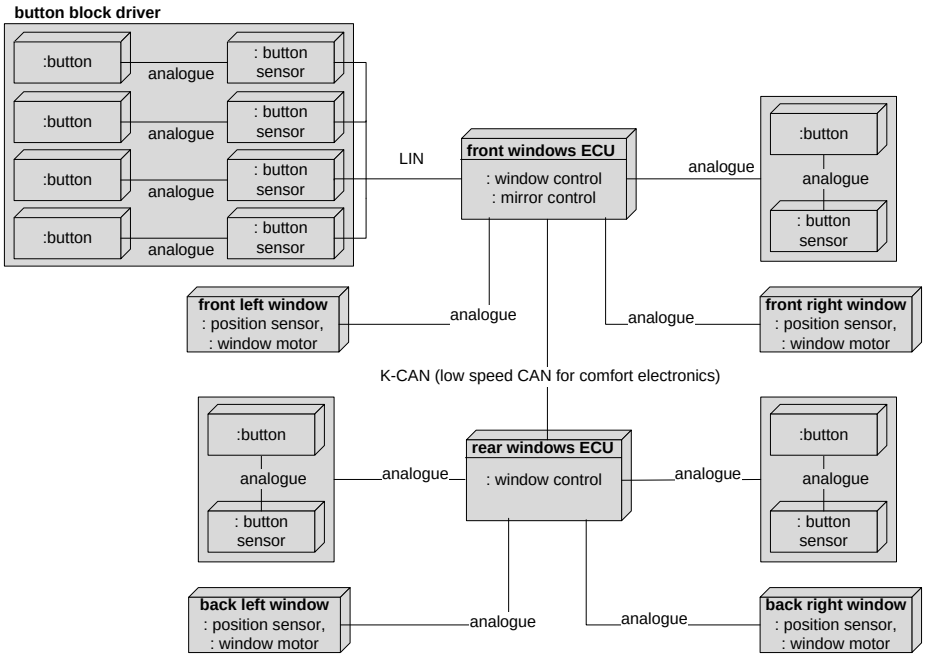


Abbildung 6.1: Fallbeispiel Fensterheber: technische Architektur eines Fensterhebers in einem Fahrzeug mit 4 Fenstern. Quelle: [KB10]

mittels des *Local Interconnect Network* (LIN, vgl. [WR11, S.220ff]) an das Steuergerät für die Vorderfenster. Das LIN-Netzwerk ersetzt somit die einzelnen Stichleitungen zum Steuergerät der vorderen Fenster, das auch *Front Electronic Module* genannt wird. Befehle an die hinteren Fenster, soweit vorhanden, werden von diesem Steuergerät über einen *low-speed-CAN-Bus* an das Steuergerät für die hinteren Fenster, dem *Rear Electronic Module* weitergeleitet. Eine Übersicht über die Bussysteme des Fahrzeugs findet sich in [NHB05] und [ZS08].

Jedes Fenster verfügt über einen *Fensterhebermotor*. An den Motor ist ein *Seilzug* angeschlossen, der mit dem *Fensterglas* so verbunden ist, so daß eine Bewegung des Fenstermotors das Fenster in die gewünschte Richtung bewegt. In den Motorblock sind zusätzlich zwei *Hall-Effekt-Sensoren* (vgl. [Ram06]) integriert. Die Hall-Sensoren werden als *Positionssensoren* eingesetzt, aus deren Signal die Programmlogik des Fensterhebers die Positionsänderung und zusätzlich Drehrichtung sowie Drehzahl des Motors bestimmen kann (vgl. [Rei06, Kap. 3.6.6 sowie Kap. 10.6.5]).

6.1.3 Hierarchisches Funktionsmodell Fensterheber

Abbildung 6.2 zeigt das schon in Abschnitt 4.2.1 dargestellte Beispiel des hierarchischen Funktionsmodells für die Funktion *automatisches Fensterschließen*. Dabei stellen die Knoten Funktionen dar und in fetter Schrift markierte Funktionen verteilte Funktionen. Dieser Aufbau ist für beide Fenstersteuergeräte gleich, nur empfängt das Rücksitzmodul seine Eingaben entweder über die beiden hinteren Schalter oder den low-speed-CAN. Der genaue Ablauf wird im nächsten Abschnitt beschrieben.

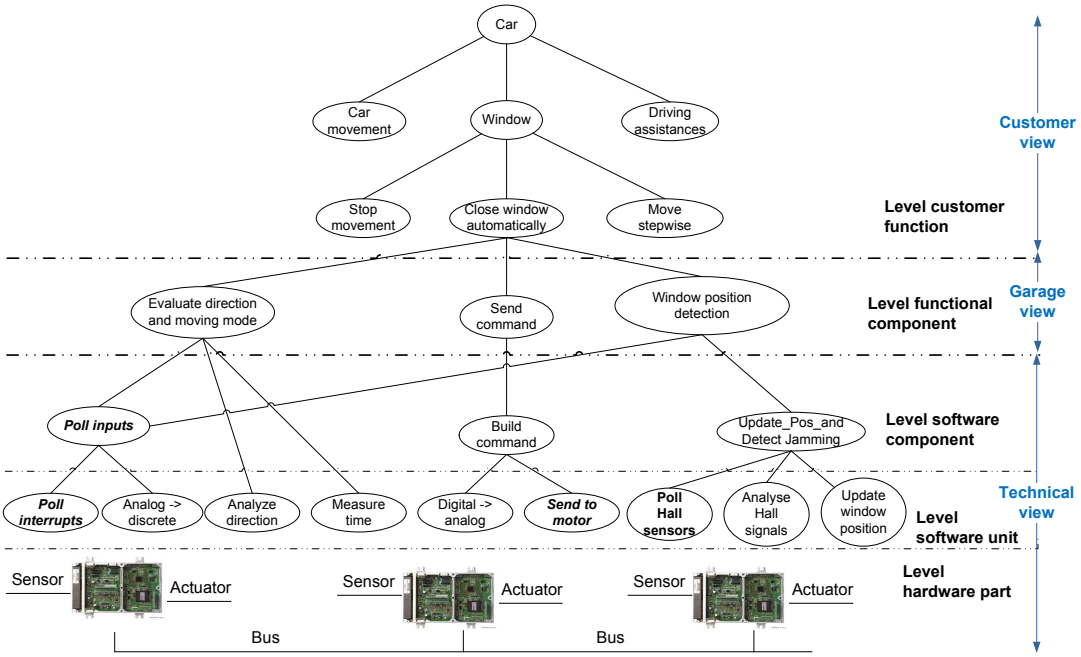


Abbildung 6.2: Fallbeispiel Fensterheber: hierarchisches Funktionsmodell für Fensterheberfunktion *automatisches Schließen*. Quelle: eigene Darstellung

6.1.4 Interaktionsmodell Fensterheber

Abbildung 6.3 zeigt das Interaktionsmodell des Fensterhebers auf den verschiedenen definierten Ebenen des Rollenmodells Diagnose. Die Ebenen sind getrennt und stellen die Perspektive der einzelnen Rollen Kunden-, Werkstatt- und technische Sicht auf den Fensterheber dar.

Im Bild wird unterschieden zwischen für die jeweilige Ebene beobachtbaren und nicht beobachtbaren Interaktionen. Fett markierte Pfeile stellen für die Ebene beobachtbare Ereignisse dar, nicht beobachtbare Ereignisse sind mit gestrichelten Pfeilen dargestellt. Graue Rechtecke entsprechen dabei Hardware-Teilen, die in diesem Fall gleichzeitig die austauschbaren Einheiten darstellen (vgl. Abbildung 6.1). Weiße Rechtecke sind die einzelnen Komponenten, die die Funktionsumfänge erfüllen.

Je nach Tiefe der Abstraktionsebene nimmt der Detailgrad des Modells zu. Dies zeigt sich einerseits in der verfügbaren Detailtiefe der betrachteten Hardware-Teile, aber vor allem in der Anzahl der möglichen Beobachtungen, die mit Indizes C_i, G_i und T_i für die jeweilige Ebene versehen sind. So ergeben sich für die Werkstattebene fünf mögliche Observationen, für die technische Ebene hingegen 12.

Im weiteren Verlauf dieses Abschnitts wird anhand der Beobachtungen und Interaktionen der Fensterheber im Detail erklärt.

Kundensicht. Dem Kunden ist nur eine Black Box-Sicht auf den Fensterheber möglich. Der Kunde kontrolliert das Fenster mit dem Schalter (C1), wodurch sich eine beobachtbare Aktion des Fensterhebers ergibt (C2).

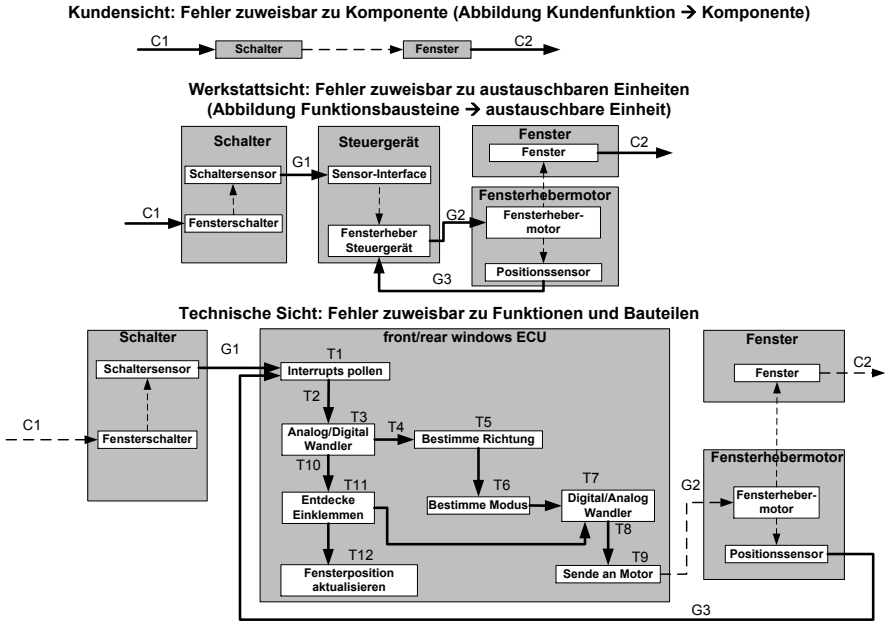


Abbildung 6.3: Fallbeispiel Fensterheber: beobachtbare Interaktionen.
Quelle: basierend auf [KB10, Abbildung 3]

Werkstattsicht. Aus Sicht der Werkstätten besteht der Fensterheber aus vier austauschbaren Einheiten mit dem Schalter, dem Fenster, dem Fensterhebermotor sowie einem Steuergerät, auf dem die Software-Funktionen zur Steuerung des Fensterhebers gespeichert sind. Der Schalter besteht wiederum aus einem Sensor und einem mechanischen Teil und ist über ein analoges Kabel mit dem Steuergerät verbunden. Über diese Verbindung werden die Kommandos des Schalters in Form von analogen, kontinuierlichen Signalen (G1) an das Steuergerät gesendet. Diese Signale können von der Werkstatt gemessen werden, um die Funktionsfähigkeit des Fensterhebers zu prüfen.

Durch Analyse des Eingangssignals ermittelt das Steuergerät, in welche Richtung (oben/unten) das Fenster bewegt werden soll und ob ein stufenweises oder automatisches Bewegen des Fensters verlangt wird. Aus diesen beiden Parametern wird ein Kommando gebildet, das dann über ein analoges Signal zum Fensterhebermotor gesendet wird. Der Motor bewegt einen angeschlossenen Seilzug und somit das Fenster. Auch diese Verbindung zwischen Steuergerät und Fensterhebermotor kann von der Werkstatt gemessen werden (G2).

Ein besonderes Augenmerk muß auf die Funktion automatisches Schließen des Fensters geworfen werden, da durch das Ausführen dieser Funktion Gegenstände eingeklemmt werden können. Deshalb müssen mögliche Einklemmungen erkannt und im Falle eines Einklemmens der Fensterlauf sofort reversiert werden. Üblicherweise setzen Fensterheber heutzutage für den Einklemmschutz ein auf Hall-Effekt-Sensoren basierendes Verfahren ein. Auf der Motorwelle des Fensterhebermotors ist ein Magnetrad angebracht, dessen Magnetfeld zwei zueinander orthogonal angebrachte Hall-Sensoren überstreicht. Bei jeder Bewegung des Motors ergibt sich eine Änderung des Magnetfelds, das die beiden Hall-Sensoren auslesen können. Hall-Effekt

Sensoren nutzen dabei den physikalischen Hall-Effekt und erzeugen eine meßbare Ausgangsspannung, aus der sich über die Amplitude die Drehzahl in Umdrehungen pro Minute des Fensterhebermotors messen läßt. Bei Hinzunahme eines weiteren Hall-Sensors und einer zueinander orthogonalen Ausrichtung beider Sensoren an dem Motor läßt sich zudem die Drehrichtung des Motors bestimmen [Rei06, Kap. 10.6]. Eine detaillierte Beschreibung der Hall-Sensoren findet sich in [Ram06].

Nach jeder Bewegung des Fensterhebermotors prüft das Steuergerät die von beiden Hall-Sensoren anliegende Spannung (G3), um aufgrund Drehrichtung und -zahl die Position des Fensters zu bestimmen und zu aktualisieren. Daraus läßt sich, wie weiter unten im Text gezeigt wird, ein mögliches Einklemmen erkennen. Das Signal der Hall-Sensoren an das Steuergerät ist auch für die Werkstatt meßbar.

Technische Sicht. Die technische Sicht auf den Fensterheber ist deutlich detaillierter als die beiden vorangegangen. Die einzelnen ablaufenden Funktionen finden sich auf der Ebene Software Unit in Abbildung 6.2.

Die erste durchgeführte Funktion ist das Pollen der Eingangssignale des Fensterschalters (T1). Falls ein analoges Signal des Fensterschalters entdeckt wird, so wird das analoge, kontinuierliche Signal in einen diskreten Wert umgewandelt (T2, T3) und der Funktion *Evaluate direction and moving mode* übergeben, die die gewünschte Bewegungsrichtung bestimmt (T5). Da das Eingangssignal kontinuierlich ist, wird mit der Signalerkennung ein Timer gestartet, um herauszufinden, ob der Kunde einen stufenweisen oder automatisierten Fensterlauf haben möchte (T6). Die Zeitmessung wird gestoppt, sobald das Signal nicht mehr anliegt oder eine andere Richtung anzeigt.

Nachdem Modus und Bewegungsrichtung bestimmt wurden, muß ein Kommando erstellt und an den Fensterhebermotor weitergegeben werden. Da der Motor über ein analoges Signal an das Steuergerät angeschlossen ist, muß das diskrete Signal in ein analoges gewandelt werden (T7) und kann anschließend als Kommando (T8, T9) über das analoge Kabel an den Fensterhebermotor geschickt werden (G2).

Um ein Einklemmen zu vermeiden, müssen die Signale der Hall-Sensoren ausgewertet werden (G3) und die neue Fensterposition bestimmt werden. Hierzu werden die Hall-Sensor Signale über einen Interrupt ausgelesen (T1) und in diskrete Werte umgewandelt (T3). Durch Auswertung von T3 läßt sich die Drehrichtung und Drehzahl des Fenstermotors und somit die aktuelle Position sowie ein mögliches Einklemmen entdecken.

Das Erkennen eines Einklemmens durch die Funktion *Detect jamming* (T11) basiert auf der Auswertung der Drehzahländerung des Fensterhebermotors. Im Falle eines Einklemmens blockiert ein Gegenstand den Fensterlauf, wodurch sich die Geschwindigkeit und somit die Drehzahl des Fensterlaufs plötzlich ändert. Die Drehzahl des Fensterhebermotors läßt sich aus dem Kehrwert der diskretisierten Hall-Sensor-Signale (T3) bestimmen. Die Drehzahländerung wird dann bestimmt aus dem Gradienten von aufeinanderfolgenden Drehzahlen. Die Drehzahländerung ist zwar der Basiswert für die Einklemmerkennung, wird aber in der Praxis aufgrund von Erfahrungswerten um weitere Faktoren erweitert. So wird, um beispielsweise temporäre Effekte wie Frost oder Kälte auszuschließen, eine gemittelte Drehzahländerung als Basis für die Bestimmung eines Einklemmens genommen. Weiterhin wird oft auch im Falle einer erkannten Verringerung der Motordrehzahl während des Fensterlaufs die Motorspan-

nung untersucht, um ein Einklemmen zweifelsfrei zu erkennen. Hintergrund hierfür ist das aufgrund des Widerstands, den ein eingeklemmter Gegenstand dem Fensterlauf bietet, mehr Kraft durch den Motor notwendig ist und somit eine Erhöhung der Stromstärke nach sich zieht.

Im Falle eines Einklemmens wird das Fenster sofort reversiert (beobachtbar mittels C2), andernfalls wird die Fensterposition aktualisiert (T12). Da die Funktion *automatisches Schließen* mitsamt dem zugehörigen Einklemmschutz aufgrund möglicher Gefahren für Insassen sicherheitsrelevant ist (Einstufung ASIL A), wird im Falle eines erkannten Fehlers der Hall-Sensoren der Fensterlauf durch das Steuergerät deaktiviert.

6.1.5 Erweiterte FMEA

Tabelle 6.1 zeigt die erweiterte FMEA für das Fallbeispiel Fensterheber.

6.1.6 Hierarchische Fehlermodellierung

In diesem Abschnitt wird gezeigt, wie aus der FMEA in Tabelle 6.1 das formale Fehlermodell der rollenbasierten Diagnose entsteht.

Die rollenbasierte Diagnose wird auf drei verschiedenen Abstraktionsebenen durchgeführt und besteht aus aussagenlogischen Formeln, die das spezifizierte Verhalten aus der Sicht der einzelnen Rollen darstellen. Die technische Sicht wird auf das Steuergerät und die Werkstattsicht auf dem Tester gespeichert.

Zur Laufzeit wird das Modell durch das beobachtete Verhalten erweitert, wodurch sich ein Erfüllbarkeitsproblem ergibt, das durch einen SAT-Solver gelöst werden kann.

Zuerst werden den für das Rollenmodell benötigten FMEA-Inhalten Variablen zugewiesen. Die FMEA wurde auf der Funktion *automatisches Schließen Fenster* durchgeführt. Dabei wurden mögliche Fehlerursachen untersucht, die zu den beobachtbaren Fehlereffekten *Fenster bewegt sich permanent nicht* sowie *Fensterlauf stoppt abrupt* führen. Tabelle 6.2 zeigt die Zuweisung von Variablen zu den einzelnen Fehlerursachen.

Fehlermodell auf Kundenebene

$$(\neg C2 \rightarrow CC_1 \vee CC_2)$$

Der Kunde erwartete ein automatisches Schließen des Fahrerfensters, nachdem er den Fensterschalter länger drückte, aber die Reaktion des Fensters war nicht das erwartete Schließen. Bewegt sich das Fenster auch nach mehrfachen Versuchen über einen längeren Zeitraum nicht wie erwünscht, so berichtet der Kunde das Verhalten des Fensterhebers in der Werkstatt. Die Werkstatt erfaßt die im weiteren Verlauf mit CC_1 kodierte Beschwerde des Kunden „Fenster bewegt sich permanent nicht“. Ein weiteres Fehlverhalten des Fensterhebers, ein „abruptes Stoppen des Fensterlaufs“ wird mit CC_2 kodiert. In diesem Fall erwartete der Kunde einen automatischen Fensterlauf bis zum oberen oder unteren Anschlag des Fensters, der aber plötzlich unterbrochen wurde.
Es sei an dieser Stelle erneut darauf hingewiesen, daß einige der Formeln im weiteren

System: Fensterheber				
Funktion	**Ursache**	**Effekt (auf Rolle)**	**Entdeckung**	**Maßnahme on-board**
automatisches Schließen Fensterheber	Kabel von Schalter zu ECU defekt	K: Fenster bewegt sich permanent nicht, W: kein ECU-Eingangssignal	W: Kabel von Schalter zu ECU ohne Strom	T:-
	Kabel von ECU zu Fensterhebermotor defekt	K: ” ”, W: keine Hall-Sensoren-Eingänge	W: Kabel ECU zu Fenster ohne Strom und DTC_1 gesetzt, T: keine Wertänderung Hall-Sensoren nach Fensterlauf	T: setze DTC_1
	falsche Polarität der Hall-Sensoren	K: ” ”, T: aktuelle Position sowie Einklemmen können nicht erkannt werden	W: Kabel Hall-Sensor zu ECU ohne Strom und DTC_3 gesetzt, T: Hall-Sensor-Signal zeigt Fensterlauf in falsche Richtung an	T: setze DTC_3, deaktiviere Fenster
	Schalter defekt	K: ” ”, W: Schalter nicht bewegbar	W: Kabel von Schalter zu ECU ohne Strom, T: -	T: keine, mechanischer Fehler
	Schaltersensor defekt	K: ” ”, W: Schalterbewegung nicht erkannt		
	ECU defekt	K: ” ”, W: Eingangssignale zu ECU liegen vor, keine zu Motor	W: Eingangssignal zu ECU liegt vor, Kabel von Motor zu ECU ohne Signaländerung, DTC_1 nicht gesetzt	T: -
	Fensterhebermotor defekt	K: ” ”, W: Kabel von Motor zu ECU ohne Strom	W: Kabel Hall-Sensor zu ECU ohne Strom und DTC_2 gesetzt, T: kein Hall-Sensor Signal nach Fensterbewegung vorliegend	T: setze DTC_2
	Variablenüberlauf in Einklemmschutzalgorithmus	K: + W: Fensterlauf stoppt abrupt, T: vorliegendes Einklemmen nicht erkannt, da ausgewerteter Parameter = 0	W: -, T: Überlaufsprüfung, höhere Variablenauflösung	T: stoppe Fensterlauf
	fehlende Rechenzeit für Einklemmschutzalgorithmus	K: + W: Fensterlauf stoppt abrupt, T: Algorithmus läuft nicht durch, neue Position liegt vor durch weitere Fensterbewegung $\rightarrow$ Einklemmen!	W: -, T: Semaphor	

Tabelle 6.1: Fallbeispiel Fensterheber: erweiterte FMEA. Quelle: eigene Darstellung

Fehler	Beschreibung
F_1	Kabel von Schaltersensor zu ECU defekt
F_2	Kabel von ECU zu Fensterhebermotor defekt
F_3	falsche Polarität Hall-Sensoren
F_4	Schalter defekt
F_5	Schaltersensor defekt
F_6	ECU defekt
F_7	Fensterhebermotor defekt
F_8	Variablenüberlauf in Einklemmschutzalgorithmus
F_9	fehlende Rechenzeit für Einklemmschutzalgorithmus

Tabelle 6.2: Fallbeispiel Fensterheber: Kodifizierung der Fehler.
Quelle: eigene Darstellung

Verlauf redundant sind. Diese Redundanzen entstehen durch die automatische Umwandlung der Daten der FMEA. Dadurch wird die Anzahl der Literale und Klauseln erhöht, was somit den Aufwand und Suchraum des SAT-Solvers erhöht und die Laufzeitperformanz des Solvers reduzieren kann.

Der Suchraum kann jedoch durch die vom SAT-Solver eingesetzten Heuristiken sowie die hierarchische Struktur des Rollenmodells eingegrenzt werden. Die Rollen erfassen das Verhalten des Systems mittels Beobachtungen auf verschiedenen Detailebenen, wodurch für ein bestimmtes Verhalten somit mehrere Beobachtungen existieren. Durch die Auswertung dieser Beobachtungen werden viele zu durchsuchende mögliche Pfade des Suchraums ausgeschlossen. Für die Eingrenzung des Suchraumes durch Heuristiken sei beispielhaft die Wahl des Verzweigungselements (VSIDS, siehe Abschnitt 4.7.1 sowie [Zha03, Kap. 2.3.2]) genannt.

Fehlermodell auf Werkstattebene

$$
\begin{array}{ll}
(\neg C2 \rightarrow CC_1 \vee CC_2) & \bigwedge \\
(CC_1 \rightarrow F_1 \vee \ldots \vee F_7) & \bigwedge \\
(CC_2 \rightarrow F_8 \vee \ldots \vee F_9) & \bigwedge \\
(\neg C2 \rightarrow F_1 \vee \ldots \vee F_9) & \bigwedge \\
(\neg G1 \rightarrow F_1 \vee F_4 \vee F_5) & \bigwedge \\
(\neg G2 \rightarrow F_2 \vee F_6) & \bigwedge \\
(\neg G3 \rightarrow F_3 \vee F_7) &
\end{array}
$$

Die Werkstatt analysiert und verifiziert die Kundenbeanstandung. Durch das im Tester gespeicherte Modell weiß der Reparateur, daß falls sich ein Fenster permanent nicht bewegt oder der Fensterlauf abrupt stoppt, zumindest einer der Fehler $F_1, ..., F_9$ aus Tabelle 6.2 vorhanden sein muß; ausgenommen die Möglichkeit, daß ein bisher unbekannter Fehler für das Verhalten des Fensters verantwortlich ist. Für den weiteren Verlauf wird jedoch angenommen, daß kein unbekannter Fehler vorliegt.

Der Werkstatt stehen mehrere Testfunktionen zur Verfügung, um eine off-board Diagnose durchführen zu können. Für den vorliegenden Fall sind dies die Funktionen D_{off_1}: *Prüfe Kabel vom Schalter zum Steuergerät auf Strom*, D_{off_2}: *Prüfe Kabel vom Steuergerät zum Fenster auf Strom*, D_{off_3}: *Prüfe Kabel von Hall-Sensoren zum Steuergerät auf*

Strom. Durch diese off-board Diagnosefunktionen ist die Werkstatt in der Lage, die Beobachtungen G_1, G_2, G_3 zu erfassen. Dadurch ergibt sich:

$$D_{off_1} : G_1 \mapsto \{0, 1\}$$
$$D_{off_2} : G_2 \mapsto \{0, 1\}$$
$$D_{off_3} : G_3 \mapsto \{0, 1\}$$

Dabei wird den Variablen G_1, G_2, G_3 der Wert 1 und somit Wahrheitswert wahr zugewiesen, falls die zugewiesene off-board Diagnosefunktion Strom messen konnte.

Fehler	**Reparaturmaßnahme(n)**	**Variable**
F_1	Ersetze Kabel von Schalter zu ECU	RM_1
F_2	Ersetze Kabel von ECU zu Fensterhebermotor	RM_2
F_3	Repolarisiere im Steuergerät den Eingang der Hall-Effekt Sensoren	$RM_{3.1}$
	Tausche ECU	$RM_{3.2}$
F_4	Tausche Schalter	RM_4
F_5		
F_6	erneutes Flashen ECU	$RM_{6.1}$
	Tausche ECU	$RM_{6.2}$
F_7	Tausche Fensterhebermotor	RM_7
F_8	-	-
F_9	-	-

Tabelle 6.3: Fallbeispiel Fensterheber: vorgeschlagene Reparaturmaßnahmen für Fehlerursachen

Tabelle 6.3 zeigt die vorgeschlagenen Reparaturmaßnahmen für die einzelnen Fehler.

Die Tabelle zeigt, daß für Fehler F_3 und Fehler F_6 jeweils verschiedene Reparaturmaßnahmen vorhanden sind. Die Sortierung der Maßnahmen erfolgt im Lebenszyklus durch die Gewichtungsfunktion $\omega(RM_i)$, die die Kosten der Reparatur und deren Erfolgsgrad als Gewichtungsfaktoren einbezieht (vgl. Abschnitt 4.8.3).

Die Fehler F_2 und F_6 sowie die Fehler F_3 und F_7 sind Fehler, die auf der Kunden- und Werkstattebene die gleichen Beobachtungen haben, aber verschiedene Reparaturmaßnahmen sowie verschiedene austauschbare Einheiten haben. Deshalb müssen diese Fehler, um kostspielige Wiederholreparaturen zu vermeiden, auf einer technisch detaillierteren Ebene durch zusätzliche Beobachtungen diskriminiert werden. Für diese zusätzliche Unterscheidung wird die Unterstützung der technischen Ebene in Form des Steuergeräts benötigt.

Für die Fehler F_8 und F_9 sind keine Reparaturmaßnahmen möglich. Es handelt sich hier um Einschränkungen der Kundenfunktion automatisches Schließen, die ausschließlich auftreten, um ein Versagen dieser Funktion zu vermeiden.

Die Fehler F_1, F_4, F_5 können weder durch die Werkstatt noch das Steuergerät unterschieden werden, weswegen die Werkstatt die verschiedenen Maßnahmen probieren muß. Auch hier bietet sich eine Gewichtung der Maßnahmen an.

Fehlermodell auf technischer Ebene

$$
\begin{array}{ll}
(T_1 \rightarrow DTC_1) & \bigwedge \\
(T_2 \rightarrow DTC_2) & \bigwedge \\
(T_3 \rightarrow DTC_3) & \bigwedge \\
(T_4 \rightarrow DTC_4) & \bigwedge \\
(T_5 \rightarrow DTC_5) & \bigwedge \\
(DTC_3 \rightarrow F_3) & \bigwedge \\
(DTC_4 \rightarrow F_8) & \bigwedge \\
(DTC_5 \rightarrow F_9) & \bigwedge \\
(F_3 \rightarrow CM_1) & \bigwedge \\
(F_8 \rightarrow CM_2) & \bigwedge \\
(F_9 \rightarrow CM_2) & \bigwedge
\end{array}
$$

Das Steuergerät wertet mittels der on-board Diagnosefunktionen D_{on_i} die Beobachtungen der Monitore T_i aus. Dabei gilt folgender Zusammenhang zwischen einer on-board Diagnosefunktion und den Variablen der Monitore:

$$
\begin{array}{l}
D_{on_1} : T_1 \mapsto \{0,1\} \\
D_{on_2} : T_2 \mapsto \{0,1\} \\
D_{on_3} : T_3 \mapsto \{0,1\} \\
D_{on_4} : T_4 \mapsto \{0,1\} \\
D_{on_5} : T_5 \mapsto \{0,1\}
\end{array}
$$

In dem vorliegenden Fallbeispiel werden die diskretisierten Hall-Effekt-Sensorsignale ausgewertet (Variable T11 aus Abbildung 6.3), die von den am Fenstermotor angebrachten Hall-Effekt-Sensoren an das Steuergerät gesandt werden.

Dabei handelt es um den Monitor T_1, der untersucht, ob sich *keine Änderung des diskretisierten Hall-Sensor Signals nach Fensterbewegung* ergab, Monitor T_2, der prüft, ob *kein diskretisiertes Hall-Sensor Signal nach Fensterbewegung* vorlag und Monitor T_3, der *Hall-Sensor Signal deutet auf eine Fensterbewegung in die falsche Richtung hin* beobachtet.

Monitor T_4 ist notwendig, da aufgrund der möglichen Verwendung eines leistungsschwachen Steuergeräts aufgrund Kostendrucks der Wertebereich der Eingangsparameter der Einklemmschutzfunktion beschränkt wird, um Rechenaufwand zu reduzieren. Dadurch kann vor allem bei der Bildung von Mittelwerten (vgl. Beschreibung des Fensterhebers) oder der Bestimmung der Drehzahl durch einen Überlauf ein Wert entstehen, der für die Funktion fälschlicherweise kein Einklemmen darstellt. Deshalb wird im Falle eines Überlaufs sofort die mit CM_2 in Verbindung stehende Gegenmaßnahme *Fensterlauf sofort stoppen* initiiert.

Die beschränkte Leistungsfähigkeit des Steuergeräts kann zusätzlich dazu führen, daß der Einklemmschutzalgorithmus nicht rechtzeitig mit der Berechnung fertig wird, bevor neue Hall-Sensor Signale vorliegen. Wird der Algorithmus nicht vollständig durchgeführt, kann dies zur Folge haben, daß obwohl ein Einklemmen vorliegt, das Einklemmen nicht ausgewertet werden bzw. entdeckt werden konnte und somit das Fenster trotz Einklemmens weiter schließt. Monitor T_5 überprüft, ob der Einklemmschutz exklusiv durchgeführt wird und nicht durch neue vorliegende Hall-Signale unterbrochen wird. Dazu wird anhand eines binären Semaphors (zuerst definiert in [Dij68]) sichergestellt, daß der Algorithmus exklusiv durchgeführt wird. Liegt nun

ein neues diskretisiertes Hall-Sensor Signal vor, obwohl der Algorithmus nicht fertig ist, wird der mit DTC_5 in Verbindung stehende Fehlerspeicher gesetzt und die mit CM_2 in Verbindung stehende Gegenmaßnahme *Fensterlauf sofort stoppen* gestartet.

Falls eine der Messungen T_1, T_2, T_3, T_4, T_5 den Wert 1 durch D_{on_i} zugewiesen bekommt, wird durch das Steuergerät der jeweilige Fehlerspeichereintrag gesetzt, der der Werkstatt bei der Fehlersuche helfen soll.

Neben den Fehlern F_8 und F_9 stellt der Fehler F_3 ein Beispiel für eine Fehlermitigation dar, bei dem die Diagnose im Steuergerät in die Funktion eingreift. Im Falle dieses Fehlers wird durch Analyse der Hall-Sensoren festgestellt, daß eine Bewegung des Fensters in die entgegengesetzte Richtung des Steuergerätkommandos stattfand. Dieser Fehler hat zur Folge, daß ein mögliches Einklemmen eines Gegenstandes nicht mehr erkannt werden kann. Da der Einklemmschutz sicherheitsrelevant ist, muß das Steuergerät die in Verbindung mit CM_1 stehende Gegenmaßnahme *permanentes Deaktivieren des Fensters* starten. Durch das Deaktivieren wird somit ein mögliches kritisches Versagen verhindert. Der Unterschied zu den anderen Fehlern liegt darin, daß der Fehler F_3 durch ein bewußtes Deaktivieren der Fensterbewegungsfunktionen entsteht. Dies stellt somit das Sollverhalten im Fehlerfall dar.

Das Einbeziehen der gesetzten DTC ermöglicht der Werkstatt zwischen den einzelnen Fehlern F_2, F_6 und F_3, F_7 zu unterscheiden und somit die Reparaturmaßnahmen aus Tabelle 6.3 durchführen zu können.

$$\begin{array}{ll} (\neg G2 \wedge DTC_1 \rightarrow F_2) & \bigwedge \\ (\neg G2 \wedge \neg DTC_1 \rightarrow F_6) & \bigwedge \\ (\neg G3 \wedge DTC_3 \rightarrow F_3) & \bigwedge \\ (\neg G3 \wedge DTC_2 \rightarrow F_7) & \end{array}$$

Da der Fehler F_6 einen Defekt des Steuergeräts selber bedeutet, sind keine relevanten DTC durch dieses Steuergerät möglich. Im Falle einer auf mehreren Steuergeräten verteilten Funktion wird ein solcher Fehler, falls er von anderen Steuergeräten entdeckt wird, in deren *Ereignisfehlerspeicher* festgehalten.

6.2 Fallstudie effizienter Diagnoseprozeß

6.2.1 Prozeßphasen

In diesem Abschnitt wird anhand eines speziell ausgewählten Fehlers der Fallstudie Fensterheber der vorgeschlagene Entwicklungsprozeß aus Kapitel 5 dargestellt. Bei dem Fehler handelt es sich um Fehler F_3, *falsche Polarität der Hall-Sensoren*. Der Fehler wurde ausgewählt, da er zum einen relativ leicht verständlich ist, andererseits aber alle relevanten Aspekte des Prozesses durch ihn abgedeckt werden.

Gesamtfahrzeuganforderungsanalyse

In der ersten Phase, der **Gesamtfahrzeuganforderungsanalyse**, werden die Features des Fahrzeugs durch Use Cases bestimmt. Abbildung 6.4 zeigt einen Ausschnitt hierfür.

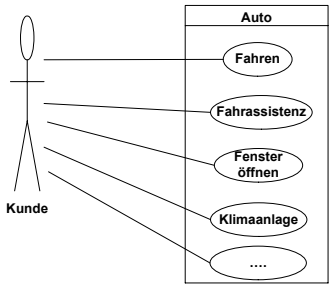


Abbildung 6.4: Fallbeispiel Fensterheber: Prozeßphase Gesamtfahrzeuganforderungsanalyse. Quelle: eigene Darstellung

Für die Diagnose werden die generellen Anforderungen festgelegt, die schon in Tabelle 2.2 dargestellt wurden, wie beispielsweise:

REQ-GFZ-ANF$_1$: Fehler müssen entdeckbar, speicherbar und zu austauschbaren bzw. reparierbaren Einheiten (AE) zuweisbar sein

REQ-GFZ-ANF$_2$: Diagnosedokumentation muß vollständig sein gemäß definiertem Datenmodell aus Abschnitt 4.5.

REQ-GFZ-ANF$_3$: Reparaturmaßnahmen müssen aus den gespeicherten Einträgen ableitbar sein.

Systemarchitekturanforderungsanalyse

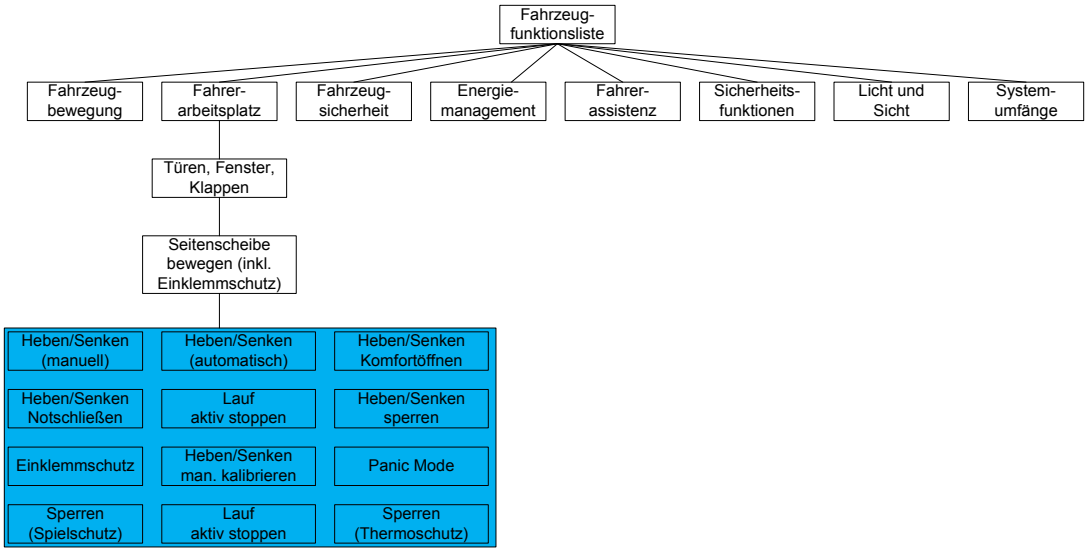


Abbildung 6.5: Fallbeispiel Fensterheber: Prozeßphase Systemarchitekturanforderungsanalyse. Quelle: eigene Darstellung

In der Phase Systemarchitekturanforderungsanalyse werden anhand der Use Cases der vorigen Phase die Kundenfunktionen des Fahrzeugs identifiziert und diese zerlegt bis hinunter zur *atomaren Kundenfunktion*. Die atomare Kundenfunktion stellt die kleinste für den Kunden erlebbare Funktion dar. Wurde die Kundenfunktion schon

entwickelt, so ist zu prüfen, welche Umfänge übernahmefähig sind.

Abbildung 6.5 zeigt die in Verbindung mit dem Fensterheber stehenden atomaren Kundenfunktionen. Für den weiteren Verlauf des Abschnitts wird dabei besonders auf die Funktion *automatisches Fensterschließen* eingegangen. Diese Funktion ist sicherheitsrelevant, da beim Hochfahren des Fensters Objekte eingeklemmt werden können.

Von Seitens Diagnose ist in dieser Phase die wichtigste Vorgabe **REQ-SA-ANF**$_1$, daß jeder für den Kunden sicht- oder erlebbare Fehler oder Einschränkung einer Kundenfunktion mit einem Fehlerspeichereintrag gesichert werden muß.

Systemarchitekturkonzept

In der nächsten Phase, der Systemarchitekturkonzeptphase, werden die Interaktionen und die daraus ableitbaren Anforderungen der atomaren Kundenfunktionen formal beschrieben, im Falle dieser Dissertation mittels MSC. Durch die Beschreibung der Interaktionen der Kundenfunktionen ergibt sich das Funktionsnetzwerk des Fahrzeugs.

Abbildung 4.5 stellt die Interaktionen der Funktion automatisches Fensterschließen dar, die durch ein längeres Drücken des Fahrzeugschlüssels ausgelöst wird. Zur besseren Veranschaulichung wird ein auf die wesentlichen Elemente reduzierter MSC der Funktion in Abbildung 6.6 dargestellt.

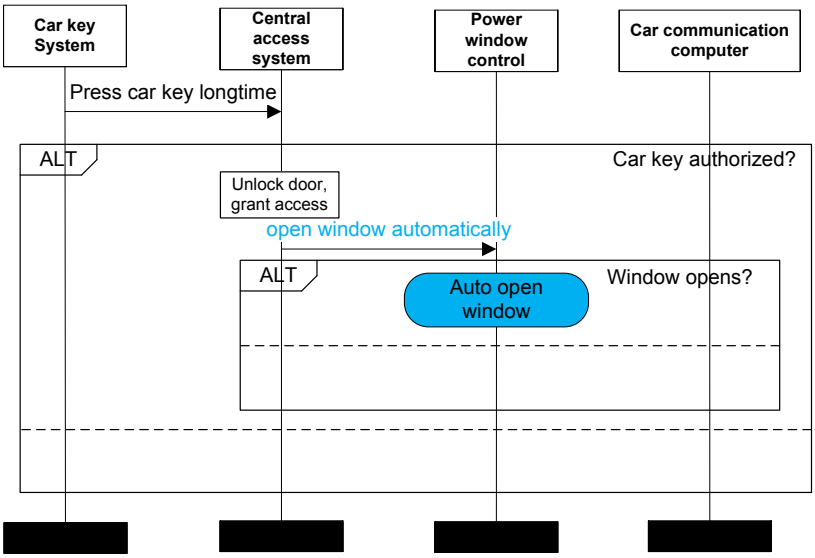


Abbildung 6.6: Fallbeispiel Fensterheber: Prozeßphase Systemarchitekturkonzept. Quelle: eigene Darstellung

Auf den MSC wird anschließend die erweiterte FMEA angewandt, wie Abbildung 6.7 anhand beispielhafter, in rot gehaltener Fehlerinformationen ausschnittweise zeigt. Auf dieser Ebene ist es bei neuen Entwicklungen meist nur möglich, den Effekt eines Fehlers auf den Kunden anzugeben. Die erfaßten Daten werden aber in den späteren Phasen mit detaillierteren Informationen angereichert. Konzentriert wird sich im weiteren Verlauf auf den Fehler, daß ein Einklemmen nicht erkannt werden kann

aufgrund der falschen Polarität der Hall-Sensoren. Ein Auszug aus der durchgeführten FMEA für diese Phase findet sich in der Zeile *SA-Konzept* der Tabelle 6.7.

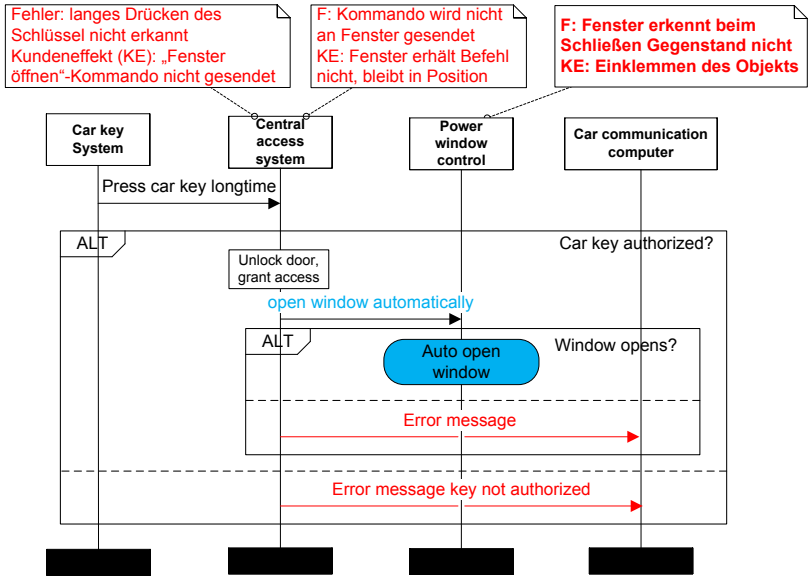


Abbildung 6.7: Fallbeispiel Fensterheber: Prozeßphase Systemarchitekturkonzept mit erweiterter FMEA. Quelle: eigene Darstellung

Systemarchitekturfunktionsspezifikation

Die atomaren Kundenfunktionen werden in der Phase Systemarchitekturfunktionsspezifikation in Funktionsbausteine zerlegt und die Anforderungen dieser Bausteine über Use Cases erarbeitet. Zudem findet hier der Übergang von der Kundenebene in die Werkstattebene statt. Die Daten der durchgeführten FMEA für diese Phase finden sich auszugsweise in Tabelle 6.7 in der Zeile *Systemarchitekturfunktionsspezifikation/-design*.

Für die Diagnose ergeben sich folgende Anforderungen, die später in der Phase Systemarchitekturfunktionsakzeptanztest validiert werden müssen:

REQ-SA-FKT$_1$: alle mit dem Einklemmschutz in Verbindung stehenden Funktionen müssen ASIL A erfüllen.

REQ-SA-FKT$_2$: falls der Einklemmschutz länger als eine kurze Zeitspanne nicht funktioniert, muß die automatische Fensterbewegung deaktiviert werden.

REQ-SA-FKT$_3$: eine permanente Fehlfunktion des Einklemmschutzes muß dem Fahrer berichtet werden, beispielsweise durch eine Anzeige im Kombi.

Systemarchitekturfunktionsdesign

In der Phase Systemarchitekturfunktionsdesign werden die Interaktionen der Funktionsbausteine beschrieben sowie das Komponentenmodell (vgl. Abbildung 6.1) erstellt. Die Abbildungen 6.9 zeigen die modellierten funktionalen Interaktionen der Funktion automatisches Schließen in dieser Phase.

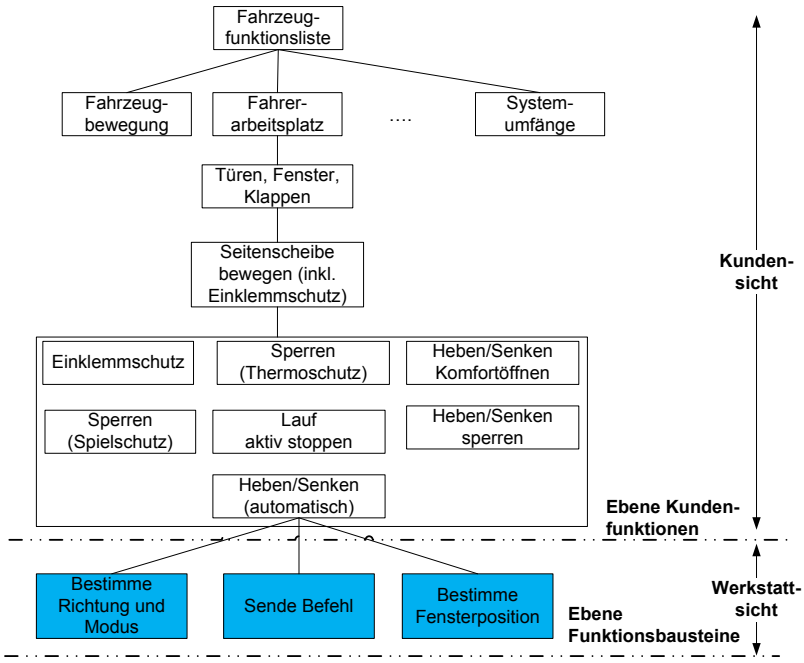
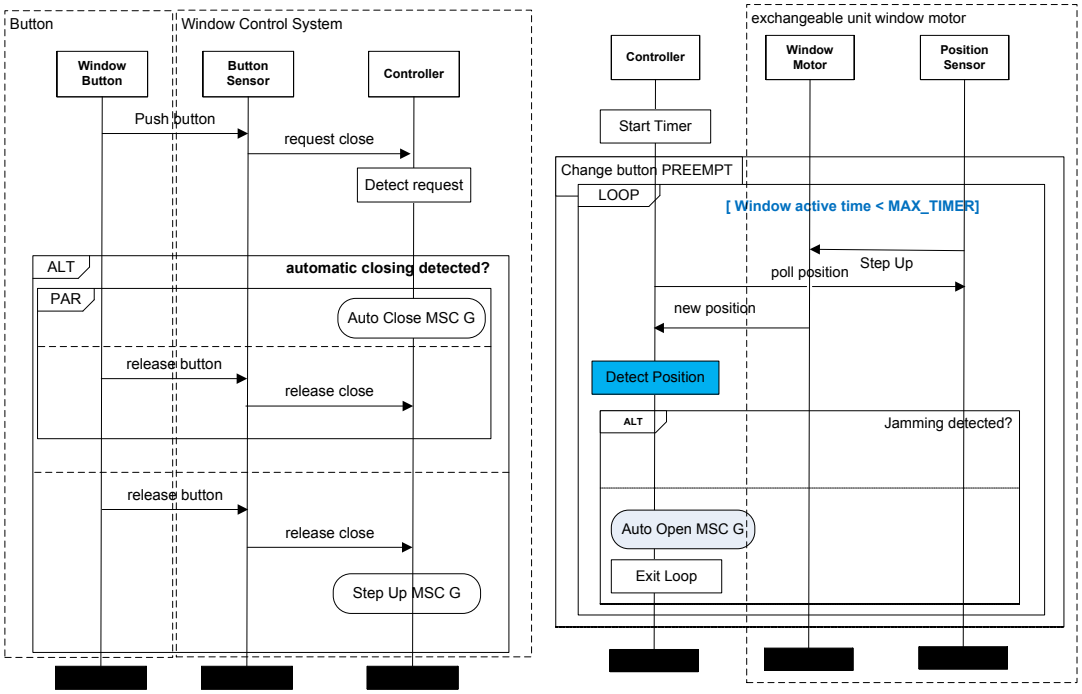


Abbildung 6.8: Fallbeispiel Fensterheber: Prozeßphase Systemarchitekturfunktionsspezifikation. Quelle: eigene Darstellung



(a) Interaktionen vom Schalter bis Steuergerät (b) MSC automatisches Fensterschließen

Abbildung 6.9: Fallbeispiel Fensterheber: Prozeßphase Systemarchitekturfunktionsdesign. Quelle: eigene Darstellung

Die gestrichelten Kästen rahmen die austauschbaren Einheiten ein. Das rechte Bild in Abbildung 6.9 stellt den MSC für die Funktionalität Fenster schließen dar, auf die im linken Bild verwiesen wird.

Auf die Interaktionen wird die erweiterte FMEA durchgeführt, um die möglichen Fehler zu analysieren. Ein Auszug aus den analysierten Fehlern dieser Phase findet sich in Tabelle 6.7.

Die Schleifenbedingung *(Position < CONST_UPPER_BLOCK_POS && Timer < MAX_TIMER)* stellt ein Beispiel für eine Fehlerdetektionsmaßnahme dar. Durch die Überprüfung dieser Funktion wird vermieden, daß der Fensterhebermotor durchbrennt. Für den weiteren Verlauf des Fallbeispiels ist jedoch der blaue Kasten *Detect Position* relevant.

Aus den bisherigen Anforderungen und Wissen läßt sich somit schon eine grobe FMEA für die Funktion *automatisches Schließen* darstellen, wie Tabelle 6.4 zeigt.

System: Fensterheber				
Funktion: Automatisches Schließen Fenster				
Fehler	**Effekt**	**Ursache**	**Entdeckung**	**Maßnahme**
Einklemmen nicht erkannt	Einklemmen eines Objekts (sicherheits-relevant!)	noch unklar	Implementierung Einklemmschutz	Deaktivere Fensterlauf (REQ-SA-FKT_2)
				Nachricht an Kombi (REQ-SA-FKT_3)
				Zuweisung Fehler an tauschbare Einheit (REQ-SA-ANF_1)

Tabelle 6.4: Grobe FMEA in Phase Systemarchitekturfunktionsdesign.
Quelle: eigene Darstellung

Software-Anforderungsanalyse

In dieser Phase werden die Funktionsbausteine in die Software-Bausteine zerlegt (vgl. Abbildung 6.10) und deren Anforderungen bestimmt. Die technische Ebene beginnt mit dieser Phase.

Auf diese Zerlegung wird die erweiterte FMEA durchgeführt. Zudem wird aus Sicht der Diagnose geprüft, ob die schon bestimmten austauschbaren Einheiten weiter verfeinert werden können (**REQ-SW-ANF**$_1$). Dies ist vor allem für das Qualitätsmanagement des OEM und seiner Lieferanten von großem Interesse.

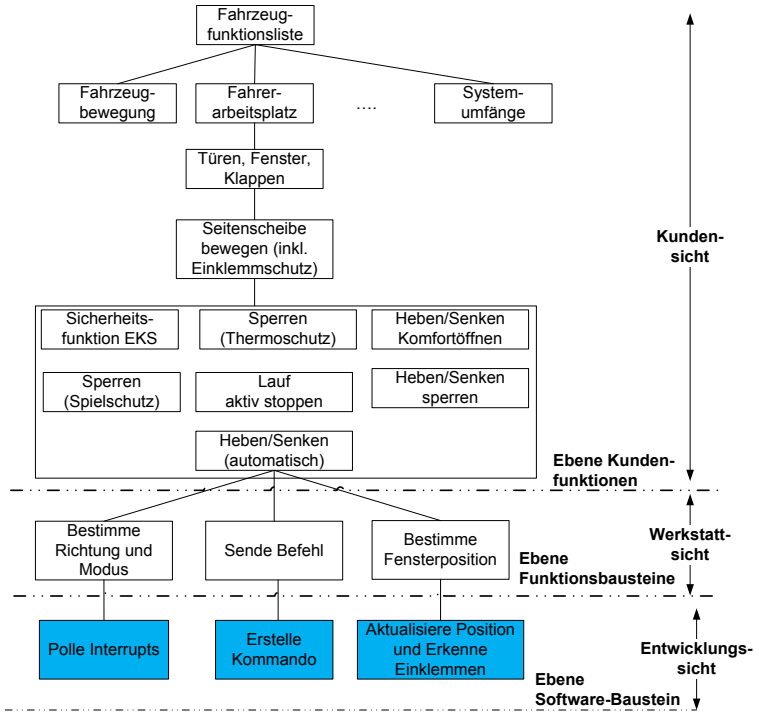


Abbildung 6.10: Fallbeispiel Fensterheber: Prozeßphase Software-Anforderungsanalyse. Quelle: eigene Darstellung

Software Architektur Design

In der Phase Software-Architekturdesign werden die Interaktionen der Software-Bausteine beschrieben.

Abbildung 6.11 zeigt die Interaktionen bis hin zur Bestimmung des automatischen Fensterschließens. Auf die Interaktionen wird die FMEA durchgeführt, um mögliche Fehler auf dieser Ebene zu erkennen.

In dieser Phase ergibt sich die zusätzliche Anforderungen **REQ-SW-ARCH**$_1$, daß sichergestellt werden muß, daß sicherheitsrelevante Funktionen regelmäßig auf korrekte Ausführung geprüft werden.

Software Unit Design

Die Software-Bausteine können nun in die Software Units zerlegt werden. Dadurch ergibt sich das gesamte Bild aus Abbildung 4.4. Das Zusammenspiel der Software Units kann beispielsweise mit State Machines (vgl. [Rum05, Kap. 7]) dargestellt werden, wie Abbildung 6.13 zeigt. Die Abbildung stellt den Ablauf des Einklemmschutzes dar. $T3, T4, T5$ stehen für die im Abschnitt 6.1.6 definierten logischen Variablen für die Beobachtungen im Steuergerät. Der Programmcode in Zustand *Update Position and Detect Jamming* wird in Listing 6.1 dargestellt.

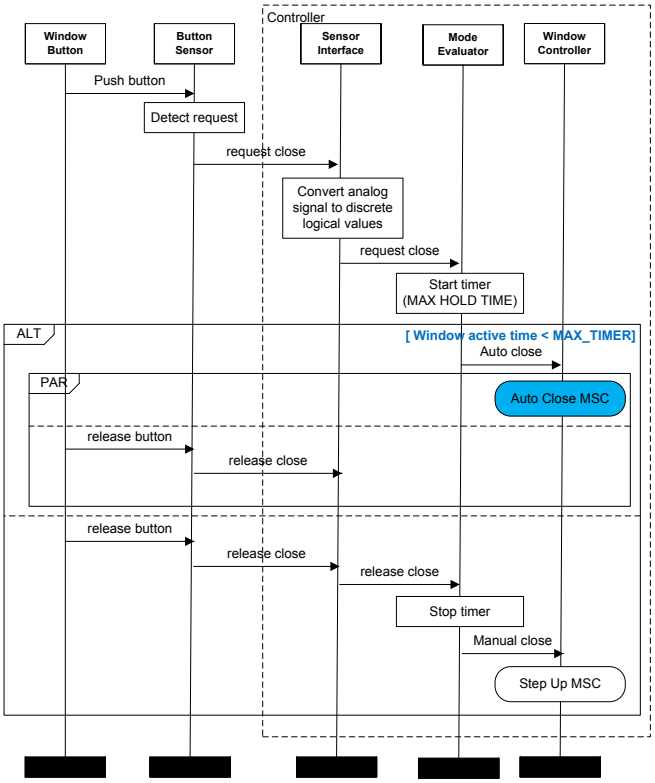


Abbildung 6.11: Fallbeispiel Fensterheber: Prozeßphase Software Architektur Design. Quelle: eigene Darstellung

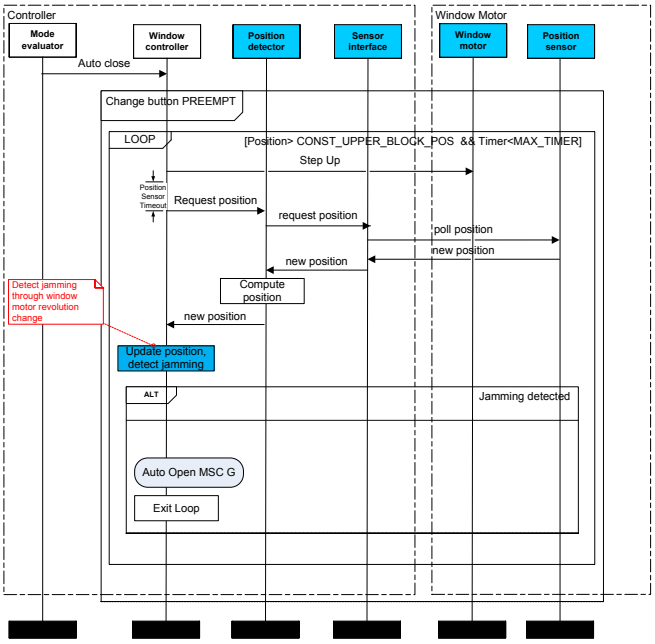


Abbildung 6.12: Fallbeispiel Fensterheber: Prozeßphase Software Architektur Design, Schritt2. Quelle: eigene Darstellung

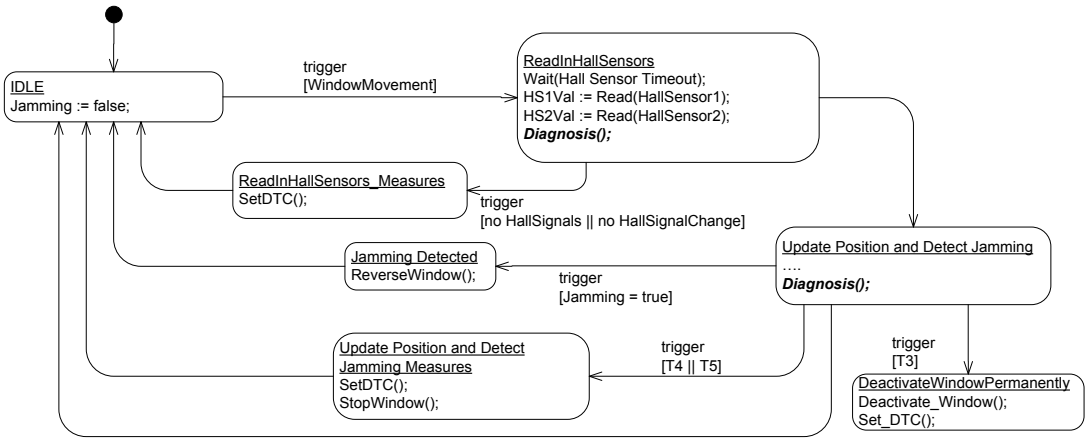


Abbildung 6.13: Fallbeispiel Fensterheber: Prozeßphase Software Unit Design. Quelle: eigene Darstellung

Programmierung und statische Analyse

Die Phase Programmierung und statische Analyse markiert den Beginn der Implementierung der Software und Diagnose. Listing 6.1 zeigt ausschnitthaft den Code für die Diagnose des Fallbeispiels und des Einklemmschutzes.

```
/* this function is run after each window movement */
void Update_Pos_Detect_Jamming(int newPos){
  bool T5, T3 = false; /* observers */
  if (Poll_Hall_Sensor_Interrupts()){ /* Hall Sensor signals are available */
    /* semaphore HallSem initialized on bootup with value 1, stored in NVRAM */
    int rtc = semaphore_getValue(HallSem, &currValue);
    if (rtc == 0){ /* semaphore could be obtained */
      if (currValue == 1){ /* HallSem.P() and its waiting queue cannot be used, time critical ! */
        HallSem.P(); /* protect */
        T3 = isGreater(WindowPos, NewPos); /* checks if old position > newer */
        if (T3){ /* function DeactivateWindowPermanently from state machine, inline here */
          STORE_DTC(DTC3); /* formula from diagnosis model: T3 > DTC3 */
          DEACTIVATE_WINDOW(); /* disable window as jammings cannot be detected */
          SEND_COMBI_REPORT(0x105); /* send fault to combi to inform driver*/
        }
        ...; /* further instructions for jamming detection */
        HallSem.V(); /* free semaphore */
      }
      else if (currValue == 0){
        T5 = true; /* jamming algorithm not completed, but new signals available! error! */
      }
    }
    else{ /* semaphore could not be obtained */
      T5 = true;
    }
    if (T5){ /* function Update Position and Detect Jamming Measures, inline here */
      SET_DTC(DTC5);
      STOP_WINDOW(); /* mitigator */
    }
```

```
  }
  ...; /* further instructions for jamming detection */
}
```

Listing 6.1: Entdecken des Fehlers „Hall Sensoren zeigen Bewegung in falsche Richtung an“

Das Code-Beispiel zeigt die Abbildung der Diagnoseformel $T3 \rightarrow DTC_3$ sowie die Initialisierung der Mitigatoren *Deaktiviere Fenster* sowie *Stoppe Fenster*, die dann wiederum zu Kundenbeanstandungen führen. Weiterhin wird das ordnungsgemäße Durchlaufen des Einklemmschutzes ohne Unterbrechung geprüft.

Software Unit Test

Mit der Software Unit Testphase beginnt der rechte Ast des V-Modells, in dem die erstellte Software gegen ihre Spezifikation geprüft wird. Anschließend werden die validierten Bestandteile sukzessive integriert und das integrierte Objekt wiederum auf einer höheren Ebene validiert, bis schließlich das ganze Fahrzeug mit seinen Komponenten validiert wurde.

In dieser Phase werden die Software Units validiert. Dies beinhaltet auch die Fehlerspeichereinträge auf Ebene Software Unit. Die Diagnose wird nur dann akzeptiert, falls die Funktionsfähigkeit der Setzbedingungen der Fehlerspeichereinträge auf dieser Ebene validiert werden kann. Da der Fehlerspeichereintrag des Fallbeispiels auf dieser Ebene spezifiziert wurde, muß er hier validiert werden. Tabelle 6.5 zeigt das Datenmodell des Fehlerspeichers.

Die Spalte **Setting conditions** zeigt die Setzbedingungen des Fehlerspeichers, die die Fehlererkennung auf der technischen Ebene des Steuergeräts ausmachen. Durch diese Setzbedingungen läßt sich der Fehlerspeicher validieren. Zusätzlich muß geprüft werden, ob die Mitigatoren und die Reparaturmaßnahmen aus Tabelle 6.6 funktionieren.

DTC	DTC_3
DTC-Text	Fensterheber Fahrerseite: Hall-Sensoren zeigen falsche Drehrichtung an
Event-DTC	Nein
Setting conditions	Messung von Motorspannung nach Fensterbewegung ($G3$) sowie Bewegung Hall-Sensoren entdeckt. Aber berechnete Richtung ungleich erwarteter, da neue Fensterposition geringer als vorherige ($T3$:= (neuPos $<$ altPos))
Clamp condition	Power on
Voltage condition	Spannung $>$ 10V
Involved components	Hall-Elemente, ECU
Mitigator/ Remedy	CM_1
	RM_1
	RM_2
CC report	Nachricht 0x105

Tabelle 6.5: DTC-Datenmodell für Fehlerspeicher aus Fallbeispiel

Remedy/ Mitigator	Trigger	Description	Sorting	Costs	Measure	Steps
CM_1	on-board diagnosis	Hall-Sensoren falsch angeschlossen	-	-	präventive Diagnose	1) setze DTC 2) deaktiviere Fenster
RM_1	DTC_3 gesetzt und Kundenbeanstandung „Fenster bewegt sich nicht“ und Signal von Fenstermotor zu ECU nicht korrekt ($DTC_3 \wedge \neg G_3 \wedge CC_1$)		1	40 - 200 €	Prüfe Polarisierung	1) Prüfe Verpolung des ECU-Steckers zu FH-Motor, Verpolung beseitigen 2) DTC löschen, Fenster mehrfach betätigen, ECU anschließend neu auslesen 3) Falls Fehler/ DTC noch vorhanden, ECU tauschen.
RM_2			2	200 €	Tausche ECU	Tausche ECU

Tabelle 6.6: Maßnahmen für Beispiel-DTC

Der Wert in der Spalte **Sortierung** ergibt sich aus einer Gewichtungsfunktion, die sowohl die Kosten einer Maßnahme als auch die Erfolgsrate berücksichtigen. Die Erfolgsrate wird im Lebenszyklus automatisch aktualisiert.

Software-Integrationstest

In dieser Phase werden die Software-Bausteine und ihre Interaktionen gegen ihre Spezifikation validiert. Für die Diagnose ergibt sich für die auf dieser Ebene spezifizierten Fehlerspeichereinträge das gleiche Verfahren wie im obigen Abschnitt: die DTC-Setzbedingungen müssen validiert werden und die Wirksamkeit der Maßnahmen nachgewiesen werden.

Für sicherheitsrelevante Funktionen wird in der Phase Software Architektur festgelegt, daß die Funktion regelmäßig auf konkrete Ausführung geprüft wird (**REQ-SW-ARCH**$_1$). Durch das Aufrufen der Funktion *Update_Pos_Detect_Jamming* nach jeder Fensterbewegung wird diese Anforderung erfüllt.

Software-Akzeptanztest

Die Software-Bausteine und ihre Anforderungen werden gegen ihre Spezifikation validiert. Fehlerspeicher, die auf dieser Ebene spezifiziert wurden, müssen über ihre Setzbedingungen validiert werden.

Systemarchitekturfunktionsintegrationstest

In dieser Phase werden die Funktionsbausteine und ihre Interaktionen validiert. Für die Diagnose gilt die Validierung der DTC wie in den vorigen Phasen. Zusätzlich muß in dieser Phase geprüft werden, ob für jeden bisherigen Fehlerspeicher mindestens eine austauschbare Einheit existiert, die für die Werkstatt zugänglich ist. Weiterhin wird dokumentiert, ob diese Einheit verfeinerbar ist. Dies stellt sicher, daß jeder Fehlerspeichereintrag für die Werkstatt zu einem Teil zuweisbar ist. Durch diese beiden Überprüfungen ist dann die gesamte grundlegende Diagnose aller Steuergeräte validiert.

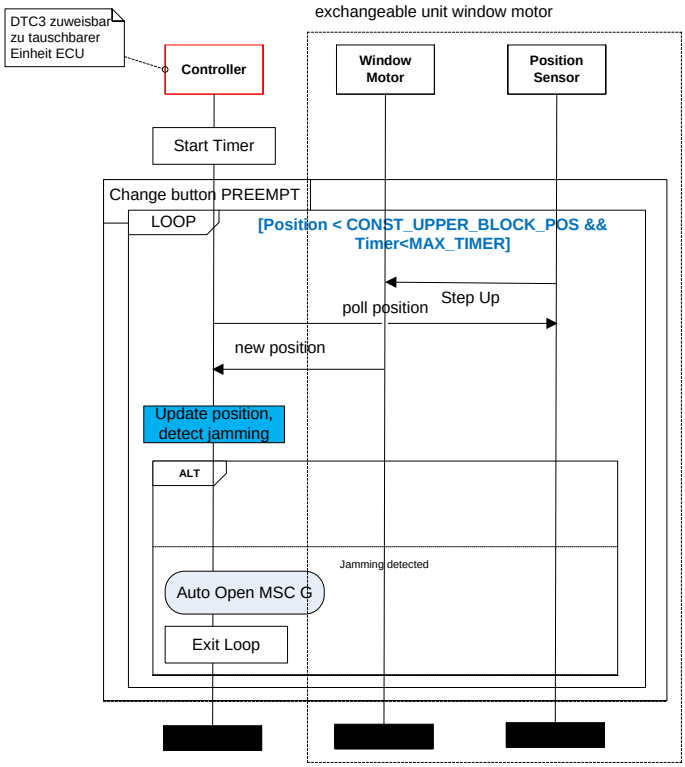


Abbildung 6.14: Fallbeispiel Fensterheber: Prozeßphase Systemarchitekturfunktionsintegrationstest. Quelle: eigene Darstellung

Abbildung 6.14 sowie die DTC-Spezifikation aus Tabelle 6.5 zeigen, daß der Fehlerspeicher DTC_3 sich eindeutig der austauschbaren Einheit Steuergerät zuweisen läßt. Somit ist der DTC spätestens ab dieser Ebene akzeptiert. Zusätzlich existiert mit der Repolarisierung der Hall-Sensoren-Eingänge eine kleinere austauschbare Einheit, wodurch die Anforderung **REQ-SW-ANF**$_1$ erfüllt ist.

Systemarchitekturfunktionsakzeptanztest

In dieser Phase werden die Anforderungen an die Funktionsbausteine validiert. In der vorigen Phase wurde sichergestellt, daß jeder bis zu dieser Ebene spezifizierte Fehlerspeichereintrag einer austauschbaren Einheit zugänglich ist. Die vollständige Akzeptanz und Validierung der komponenteninternen Diagnose erfolgt genau dann,

wenn jede bisher zugewiesene austauschbare Einheit zusätzlich für die Werkstatt zugänglich, d. h. reparierbar ist.

Zusätzlich werden in dieser Phase die fallbeispielspezifischen Anforderungen validiert. Wird die falsche Polarität der Hall-Sensoren erkannt, so wird durch die Gegenmaßnahme *deaktiviere Fenster* die Anforderung **REQ-SA-FKT**$_2$ erfüllt. Da zugleich eine Nachricht an das Kombi gesendet wird, wird zudem **REQ-SA-FKT**$_3$ befriedigt. Auf ASIL wird im Rahmen der Arbeit nicht näher eingegangen (**REQ-SA-FKT**$_1$).

Systemarchitekturintegrationstest

Durch Validierung der Interaktionen der Funktionsbausteine wird in dieser Phase das gesamte Feature-Netzwerk akzeptiert. Da in der vorigen Phase die komponenteninterne Diagnose abgeschlossen und akzeptiert wurde, kann nun mit der Validierung der komponentenübergreifenden Diagnose begonnen werden. Dies beginnt mit der Beschreibung und Implementierung der Busnachrichten, mit denen die interagierenden verteilten Funktionen diagnoserelevante Daten austauschen können. Dies kann erst in dieser Phase geschehen, da die Komponentenumfänge vorher nicht validiert und bestätigt waren.

Systemarchitekturakzeptanztest

In der Phase Systemarchitekturakzeptanztest werden die atomaren Kundenfunktionen gegen ihre spezifizierten Anforderungen validiert.

Weiterhin werden in dieser Phase die komponentenübergreifenden Fehlerspeicher validiert. Somit ist die komponentenübergreifende Diagnose akzeptiert.

Das Fallbeispiel betrifft einen Fehler der kundenrelevanten Funktion *automatisches Schließen des Fensters*. Da durch die Gegenmaßnahme *deaktiviere Fenster* diese Funktion abgeschaltet wird, muß diese Einschränkung mit einem Fehlerspeichereintrag gespeichert werden (**REQ-SA-ANF**$_1$), was durch Speicherung von DTC3 geschieht.

Gesamtfahrzeugakzeptanztest

In der letzten Phase wird sichergestellt, daß die Anforderungen der Fahrzeug-Features konform mit ihren Anforderungen sind.

Hierzu gehört auch die Diagnose. Es wird überprüft, ob die gesamte Diagnose den generellen Anforderungen an die Diagnose genügt, die in Tabelle 2.2 aufgelistet sind. Dazu gehört beispielsweise, daß die Dokumentation der Diagnose vollständig sein muß. Sind diese Anforderungen alle erfüllt, so ist die Diagnose vollständig akzeptiert.

Für das Fallbeispiel ergeben sich somit folgende Überprüfungen: die definierten Fehler (im Fallbeispiel nur einer) werden durch die Software-Funktion *Update_Pos_Detect_Jamming* entdeckt, mit einem Fehlerspeichereintrag (DTC_3) gesichert und der austauschbaren Einheit Steuergerät (bzw. Hall-Sensoreneingang im Steuergerät) zugewiesen. Somit ist Anforderung **REQ-GFZ-ANF**$_1$ erfüllt.
Da die Daten der vorgegebenen Datenmodelle für die erweiterte FMEA (vgl. Ab-

bildung 4.7) sowie eines Fehlerspeichers (vgl. Abbildung 4.8) vollständig für alle möglichen Fehler befüllt sind, ist weiterhin Anforderung **REQ-GFZ-ANF**$_2$ erfüllt. Die vollständige Befüllung schließt auch die Erfassung der Reparaturmaßnahmen ein (**REQ-GFZ-ANF**$_3$).

6.2.2 Erweiterte FMEA des Prozeßbeispiels

Tabelle 6.7 zeigt zusammenfassend die mit der erweiterten FMEA erstellte Analyse des gewählten Fehlers in den verschiedenen Entwicklungsphasen. Dabei nimmt die Detailtiefe in den einzelnen Phasen zu.

System: Fensterheber							
Funktion	**Fehler**	**Ursache**	**Effekt**	**Entdeckung**	**Vemeidung**		**Phase**
					on-board	**off-board**	
aut. Schließen Fensterheber	Einklemmen nicht erkannt	K: ?	K: Einklemmen eines Gegenstands	K: Objekt eingeklemmt	K: -	K: -	SA-Konzept
Bestimme Fensterposition	Fehler in Fkt.	W: fehlerhafte Eingänge	K: + W: Einklemmen	K: -, W: Hall-Sensor-Eingang (G3) messen	K: -, W: -	K: -, W: Repolarisiere Eingänge in ECU (RM_1), tausche ECU (RM_2)	SA-Funktionsspezifikation / SA-Funktions-Design
		W: keine Eingangssignale	K: + W: Einklemmen	K: -, W: kein Hall-Sensoreingang meßbar (G3)	K: -, W: -	K: -, W: tausche Fenstermotor (RM_3)	
Aktualisiere Fensterposition und entdecke Einklemmen	falsche Eingabe für Funktion Update Position & Detect Jamming	T: falsche Polarität der Hall-Sensoren	K: Einklemmen, W: Einklemmen, T: Einklemmen kann nicht erkannt werden, da keine Erkennung aktueller Position	K: -, W: G3 messen und prüfen, ob DTC_3 gesetzt, T: T_3:= (neue Position $\leq$ vorige Position)	K: -, W: -, T: DTC_3 setzen, Kombinachricht senden, Fenster abschalten	K: -, W: Repolarisiere Eingänge in ECU (RM_1), tausche ECU (RM_2)	SW-Anforderungsanalyse/ SW-Architekturdesign

Tabelle 6.7: Prozeßbeispiel Fensterheber: erweiterte FMEA. Quelle: eigene Darstellung, basierend auf Abb. 4.7

6.3 Fallstudie Wartung im Lebenszyklus anhand des ACC

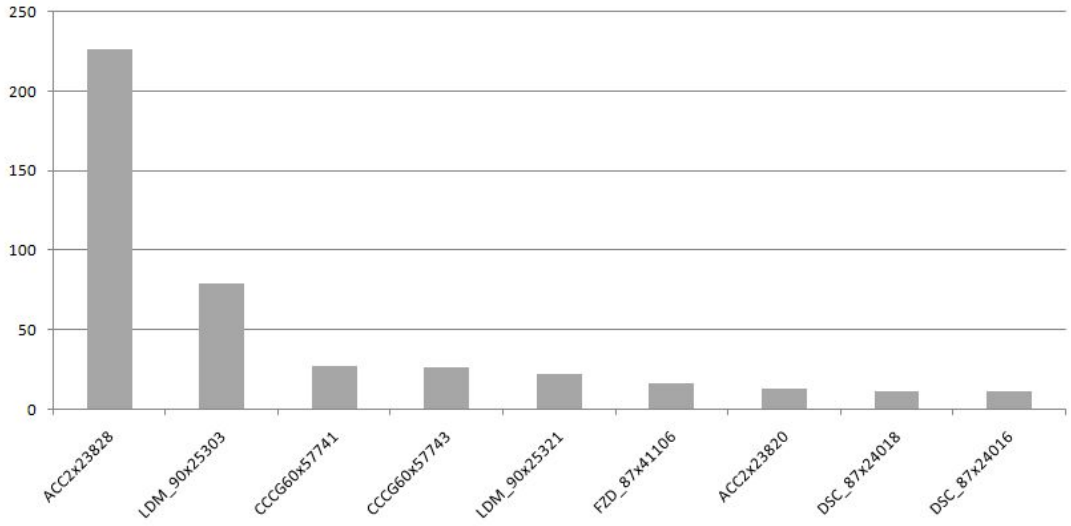


Abbildung 6.15: Gesetzte DTC im Falle der Wiederholreparatur „tausche ACC“. Quelle: [KKP+11, Abbildung 2]

Die Evaluation ist aus [KKP+11, Kap. V] entnommen. Abbildung 6.15 zeigt einen Ausschnitt der am häufigsten gesetzten DTC im Falle einer wiederholten, erfolglosen Anwendung der Reparaturmaßnahme *tausche ACC*, wodurch eine Wiederholreparatur entstand. Dabei sind die DTC beschriftet mit dem Namen des Steuergeräts, das den DTC setzte und der Nummer des DTC.

Die Abbildung zeigt, daß der *DTC 25303* mit dem Fehlertext „LDM heruntergefahren wegen ACC Sensor“, der durch das für die Längsdynamikkontrolle des Fahrzeugs verantwortliche Steuergerät *LDM_90* gesetzt wurde, mit dem ACC in Verbindung steht.

Durch Untersuchung der Verbindungen zwischen den einzelnen DTC der verschiedenen Steuergeräte kann, wie die Abbildung und das Beispiel zeigen, die Propagation von Fehlern untersucht werden. Da für jeden DTC seine Setzzeit verfügbar ist, läßt sich die zeitliche Kette genau bestimmen.

Die Erstellung dieser Kette ist vor allem bei der Analyse verteilter Funktionen nützlich, da dort wie erwähnt oft nicht alle Zusammenhänge der Funktionen und Auswirkungen einer Funktion auf den Gesamtverbund bekannt sind. Zudem sind diese Zusammenhänge schwer in Erfahrung zu bringen, falls die Anteile einer verteilten Funktion von verschiedenen Zulieferern erstellt werden.

Kapitel 7

Evaluation

In diesem Abschnitt wird der vorgestellte Ansatz der Dissertation evaluiert. Als Grundlage der Evaluation dienen dabei die in Kapitel 3 bestimmten Potentiale.

Abschnitt 7.1 bietet eine Übersicht dieser Potentiale und stellt jedem Potential einen Beitrag der Dissertation gegenüber.

Die Auswirkungen des Beitrags der Arbeit hinsichtlich Kosten und/oder Qualität der Diagnose werden in Abschnitt 7.2 detailliert.

Übersicht

7.1 Übersicht

Tabelle 7.1 zeigt eine Übersicht der in Kapitel 3 vorgestellten Potentiale der Diagnose sowie die Auswirkungen der Beiträge dieser Arbeit auf die Faktoren Kosten und Qualität.

Potential aus Kap. 3	Beitrag der Arbeit	Nutzen	Kosten	Qualität
Verschiedene Diagnosedatenquellen (3.2)	Datenmodell reduziert Datenvielfalt	Erhöhung Wartbarkeit und Wiederverwendbarkeit	↓	↑
Einbindung des Zulieferers (3.3)	Datenmodell mit Schichten bietet Zulieferersicht	Einbindung Zuliefer ab den frühen Entwicklungsphasen	↓	↑
Spezifikation der Diagnose (3.4)	formale Spezifikation Diagnose auf drei Abstraktionsebenen	Diagnose anpaßbar an technisches Verständnis, bspw. Werkstatt		↑
		Validierung Diagnose möglich		↑
Effizienz der Diagnoseinferenz (3.5)	Transformation Inferenz in Erfüllbarkeitsproblem für SAT-Solver	Steigerung der Geschwindigkeit der Fehlerbestimmung	↓	
		Größerer Zustandsraum durchsuchbar		↑
Diagnose verteilter Funktionen (3.6)	Einführung eines Diagnoseansatzes für verteilte Funktionen	Beherrschung der Komplexität verteilter Funktionen		↑
Kein gesamthaft geeigneter Ansatz für die Diagnose aller Komponenten (3.7)	Definition eines modellbasierten Ansatzes für on- und off-board Diagnose	Ansatz einsetzbar für on- **und** off-board Diagnose aller Komponenten	↓	↑
Manuelle Erstellung und eingeschränkte Wiederverwendung Diagnose (3.8)	Automatische Erstellung aus hierarchischer Datenstruktur	Reduktion Erstellungsaufwand	↓	
		Erhöhung Qualität		↑
Wartung Diagnosedaten verbesserbar (3.9)	Wartung Diagnose durch automatisierte statistische Analyse	Entdeckung fehlerhafter Diagnosefunktionen	↓	
		Automatisierte Wartung Diagnose	↓	↑

Tabelle 7.1: Übersicht Beitrag der Arbeit zu den Top-Potentialen Diagnose. Quelle: eigene Darstellung.

7.2 Evaluation der einzelnen Potentiale

Verschiedene Datenquellen

Aus dem vorgestellten hierarchischen, rollenbasierten Datenmodell der Diagnose lassen sich die Diagnosefunktionen für die on- und off-board Diagnose automatisiert generieren und auf der Zielplattform speichern, wie in Abschnitt 4.6 gezeigt wurde. Zudem erlaubt die geschichtete Datenstruktur die Wiederverwendung von Diagnosefunktionen und -daten auf den Ebenen der Rollen Kunde und Werkstatt sowie mit Einschränkungen auch auf der technischen Ebene.

Die Wiederverwendung führt zum einen zu einer Reduktion des Erstellungsaufwands für die Diagnose (vgl. [HKK04]), zum anderen reduziert die Software-Wiederverwendung die Entwicklungszeit und somit die Entwicklungskosten und erhöht zudem sowohl die Produktivität als auch die Qualität der Software (vgl. [ER03, Kap. 4.3.5]). Weitere Arbeiten, die den Nutzen der Wiederverwendbarkeit zeigen, finden sich in [Bal98, Kap. 3.6]. Eine konsequente Wiederverwendung reduziert zusätzlich die Gefahr, daß Diagnosefunktionen für bekannte Fehler des Fahrzeugs nicht umgesetzt werden.

Durch das definierte Datenmodell wird die Informationsmenge eingegrenzt und auf einen notwendigen, geringeren Umfang beschränkt. Dadurch wird die Wartbarkeit der Daten erhöht, was vor allem aufgrund des langen Betreuungszeitraums ökonomisch wichtig ist. Der Wartungs- und Betreuungsaufwand wird zusätzlich durch das in Abschnitt 4.8 vorgestellte multivariate Statistikverfahren reduziert, da die Daten stets automatisiert gewartet und auf mögliche Fehler untersucht werden.

Einbindung Zulieferer

Durch den in Abschnitt 5 definierten Prozeß sowie das definierte Datenmodell wird vorgegeben, zu welchem Zeitpunkt in der Entwicklung welche Daten vom OEM und Zulieferer vorliegen müssen und welche Aktivitäten von beiden durchzuführen sind.

Somit wird eine Zulieferersicht erstellt (vgl. [BGK+09]), die es den einzelnen Entwicklungspartnern ermöglicht, ihre Teilumfänge am Gesamtsystem zu prüfen, beginnend in den frühen Phasen der Entwicklung und vor allem in den Integrationsphasen auf dem rechten Ast des V-Modells. Dies ist besonders bei verteilten Systemen vorteilhaft.

Diese Sicht steigert die Qualität der Diagnose, da die einzelnen Teilumfänge und deren Zusammenspiel frühzeitig abgesichert werden und reduziert die Entwicklungskosten und Fehlerkosten nach Produktion. Dies basiert auf der als Boehms erstes Gesetz bezeichneten Aussage, daß Fehler am häufigsten in den Anforderungs- und Design-Phasen geschehen und um so teurer werden, je später sie beseitigt werden [BMU75]. Dieser Sachverhalt wird auch oft als Zehnerregel bezeichnet.

Spezifikation der Diagnose

Abschnitt 4.4 zeigte wie im vorgestellten Ansatz der Arbeit die Diagnose formal auf drei Abstraktionsebenen definiert wird. Die formale Spezifikation erleichtert sowohl

die automatisierte Generierung der Diagnosefunktionen als auch die Validierung der Diagnose. Dies ist um so höher anzusehen, da gegenwärtig sehr häufig informelle Regeln als Basis für das Diagnosewissen dienen.

Formale Methoden haben gegenüber informellen Methoden den Vorteil, daß sie Design-Fehler signifikant reduzieren oder früh eliminieren (Bauer-Zemanek-Hypothese, vgl. [Zem68, BW82] aus [ER03]), wodurch die Qualität der Diagnose gesteigert wird.

Die formale Spezifikation innerhalb des Schichtenmodells erlaubt zudem, den Umfang der Diagnose an das technische Verständnis der handelnden Rolle anzupassen. Dies ist wichtig, um beispielsweise den Kunden für die Prüfung eines beanstandeten Mangels des Fahrzeugs einzubinden oder unbekannte Fehler (ggf. per Ausschlußverfahren der bekannten Fehler) nachvollziehen zu können.

Effizienz der Diagnoseinferenz

In der Dissertation wird die Diagnose in ein aussagenlogisches Erfüllbarkeitsproblem umgewandelt, das durch einen SAT-Solver gelöst wird (vgl. Abschnitt 4.7).

Da der SAT-Solver für die Lösung maximal 2^n-Suchschritte benötigt mit n der Anzahl der Atomen der Formel, handelt es sich somit um ein $\mathcal{NP}$-komplexes Problem. Aufgrund des speziellen DIMACS-Eingabeformats sowie des Einsatzes von Heuristiken erfolgt die Lösungsbestimmung des SAT-Solvers jedoch schneller als in anderen Verfahren (vgl. [GPFW96]). Der SAT-Solver ist aufgrund Datenformat und Heuristiken in der Lage, die Lösung im Mittel meist schneller zu bestimmen als das häufig eingesetzte Breitensucheverfahren [Knu97] der modellbasierten (vgl. Abschnitt 2.4.3) oder fallbasierten Diagnose (vgl. Abschnitt 2.4.1), da nicht jeder Suchpfad durchlaufen werden muß. Zusätzlich wird die Anzahl der für den SAT-Solver zu durchsuchenden Suchpfade durch die Auswertung der vorhandenen Beobachtungen des vorgestellten Rollenmodells und seinen Ebenen vor Suchbeginn reduziert.

Der SAT-Solver ermöglicht somit eine Verkürzung der Inferenz jedes durchgeführten Diagnosevorgangs und ist besonders im Gewährleistungszeitraum für den OEM interessant. Kosten von Reparaturen eines Fahrzeugs im Gewährleistungszeitraum müssen vom Hersteller übernommen werden, was zu enormen Kosten führt (vgl. Punkte N2, N3 und N4 aus Abbildung 1.2). Eine schnellere Inferenz reduziert somit die Dauer jedes Reparaturvorgangs und somit die Gesamtkosten einer Reparatur. Dennoch sind diesem Faktor durch die Notwendigkeit des Dialogs zwischen Tester und Reparateur bei vielen Fehlern Grenzen gesetzt.

Wichtiger ist jedoch, daß der SAT-Solver durch die erwähnten Faktoren Eingabeformat und Verkürzung des Suchraums in der Lage ist, viel größere Zustandsräume zu durchsuchen und nicht dem sogenannten *state explosion*-Problem unterliegt (vgl. bspw. [GGYA03]). Diese Tatsache erlaubt einen effizienten Einsatz für die Diagnose umfangreicher, verteilter Systeme, da der limitierende Faktor des bei verteilten Systemen exponentiell wachsenden Zustandsraums aufgehoben wird. Zudem können umfangreichere Auswertungen innerhalb des gleichen Zeitrasters als bei Verfahren mit schlechterem Laufzeitverhalten durchgeführt werden. Da, wie in der Einleitung beschrieben, der Umfang verteilter, Software-intensiver Systeme im Fahrzeug weiterhin wachsen wird, ermöglicht der Einsatz eines SAT-Solver somit eine für die Zukunft gewappnete Diagnose automobiler Systeme.

Der begrenzte Suchraum der bisherigen Verfahren zeigt sich in der Werkstatt bei der Reparatur verteilter Systeme durch den Tester. Ein Diagnoseverfahren kann nur dann immer einen vorhandenen Fehler bestimmen, falls es in der Lage ist, den kompletten Zustandsraum zu durchsuchen. Andernfalls wird an den Reparateur zurückgemeldet, daß kein Fehler gefunden wurde, da das Verfahren nicht in der Lage war, den Fehler zu finden. Somit kann der Fehler nicht durch den Tester behoben werden und muß durch den Reparateur mittels „trial-and-error"-Reparaturen behoben werden.

Diagnose verteilter Systeme

Der vorgestellte Ansatz der Dissertation richtet einen besonderen Fokus auf die Diagnose verteilter Systeme, die über besondere Eigenschaften verfügen und somit die Diagnose erschweren. Hier sei besonders die Explosion des zu überwachenden Zustandsraums genannt. Die zusätzlichen Vorteile des Ansatzes werden in diesem Abschnitt evaluiert.

Kein gesamthafter Ansatz für die Diagnose aller Komponenten

In Abschnitt 3.7 wurde aufgezeigt, daß in der Domäne automobile Diagnose gegenwärtig verschiedene Diagnoseansätze eingesetzt werden, die aber jedoch alle Optimierungspotentiale besitzen.

Die einzelnen Ansätze wurden verglichen und dem Ansatz der Dissertation aus Abschnitt 4 gegenübergestellt. Dabei schnitt der rollenbasierte Ansatz der Dissertation gesamthaft am besten ab. Beispielhaft für die Vorteile sind die Reduktion signifikanter Kostenfaktoren wie beispielsweise der Erstellungs- und Wartungskosten, die gesteigerte Qualität der Diagnosefunktionen durch die Wiederverwendung und formale Spezifikation sowie die gezeigte Anwendbarkeit für sowohl on- (Abschnitt 4.7.3)- als auch off-board (Abschnitt 4.7.4) Diagnose.

Manuelle Erstellung und eingeschränkte Wiederverwendung

Die gegenwärtige Diagnose im Automobil verfügt über einen Umfang von ca. 3 Millionen Code-Zeilen (vgl. Abschnitt 4.8). Dieser Umfang zeigt, daß sowohl die Wiederverwendung von Code als auch eine nicht-manuelle Erstellung von Diagnose-Code große Potentiale bergen.

Das vorgestellte Datenmodell in Abschnitt 4.5 sowie die in Abschnitt 4.6 dargestellte automatisierte Umwandlung in speicherbarem C-Code reduziert den manuellen Programmieraufwand. Zudem kann durch automatisierte Generierung die Fehleranzahl reduziert werden.

Das Rollenmodell ermöglicht somit die schon weiter oben beschriebene Erhöhung der Wiederverwendbarkeit von Diagnosedaten und -funktionen, da die Diagnosefunktionen und -daten auf den höheren Ebenen des Rollenmodells relativ gleichbleibend sind. Die Wiederverwendung ergibt somit den Nutzen der gesteigerten Qualität der Diagnose durch die Wiederverwendung und somit einer Kostenreduktion (vgl. [LG84, Jon94, HKK04] sowie [Bal98, Teil IV, Kap. 3]).

Wartung Diagnose verbesserbar

Der Nutzen der Wiederverwendbarkeit und Wartung der Diagnose wurde schon anfangs des Kapitels skizziert, soll aber hier zusätzlich kurz aus einem anderen Blickwinkel beleuchtet werden.

Der in Abschnitt 4.8 vorgestellte statistische Ansatz ermöglicht das automatisierte Entdecken von optimierbaren und fehlerhaften Diagnosefunktionen, die nicht wirksame bzw. fehlerhafte Maßnahmen vorschlagen oder falsche Symptome in ihre Suche einbeziehen. Dies führt zu erhöhten Kosten durch Wiederholreparaturen sowie zu Kundenunzufriedenheit.

Der vorgestellte statistische Ansatz entdeckt automatisch fehlerhafte oder nicht ausreichend wirksame Maßnahmen einer Diagnosefunktion und löscht diese dann. Für eine fehlerhafte Reparaturmaßnahme RM_i ergibt sich dann beispielsweise der finanzielle Nutzen aus dem Produkt der Anzahl der Anwendungen von RM_i mit den Kosten der Maßnahme RM_i.

Kapitel 8

Zusammenfassung und Ausblick

Dieses letzte Kapitel faßt die Ergebnisse der Arbeit zusammen und wirft einen Blick auf mögliche anknüpfende Arbeiten und weiterführende zukünftige Forschungsthemen.

Übersicht

8.1 Zusammenfassung

In der Dissertation wurde eine umfassende, modellbasierte Diagnosemethodik zusammen mit einem Prozeß vorgestellt, die beide den Zielkonflikt der Diagnose zwischen Qualität und Kosten durch Steigerung der Effizienz der Diagnose bewältigen.

Die Effizienz der Diagnose wurde erhöht, indem identifizierte Potentiale der Diagnose durch den in einem Prozeß eingebetteten Ansatz der Arbeit erschlossen wurden. Diese Potentiale wurden in einer umfassenden Analyse des Gesamtlebenszyklus der Diagnose analysiert und in der Arbeit näher detailliert.

Der vorgestellte Ansatz der Arbeit ist für die on- *und* off-board Diagnose aller in der Domäne Automobil eingesetzten Komponenten anwendbar. Die Funktionen und das Datenmodell der Diagnose werden durch den Einsatz eines multivariaten Statistikverfahrens über den gesamten Lebenszyklus automatisiert gewartet.

Das Diagnosemodell des vorgestellten Ansatzes basiert auf aussagenlogischen Formeln, die das Verhalten des Systems auf drei Abstraktionsebenen bzw. Rollen formal spezifizieren. Die Rollen stellen dabei mit Kunde, Werkstatt und Entwicklung die Handlungsrollen jedes Diagnosevorfalles dar und sind notwendig, um alle möglichen Fehler der Domäne abzudecken. Zudem erleichtern die Rollen die Wiederverwendung der Diagnose. Weiterhin lassen sich durch die Verwendung aussagenlogischer Formeln die geläufigsten Diagnoseansätze einbinden.

Das Diagnosemodell wird zur Laufzeit mit den ausgewerteten Beobachtungen erweitert, wodurch sich ein Erfüllbarkeitsproblem ergibt, das durch den Einsatz eines SAT-Solvers gelöst wird. Der SAT-Solver basiert auf einem speziellen Datenformat und ist in der Lage, durch Heuristiken sowie das Rollenmodell den Lösungssuchraum deutlich einzugrenzen, wodurch für das Laufzeitverhalten der Inferenz sich ein deutlich besseres Verhalten als das der bisher vorwiegend eingesetzten Breitensuche ergibt. Durch diese Verbesserung der Inferenz ist der vorgestellte Ansatz in der Lage, den exponentiellen Zustandsraum verteilter Systeme zu überwachen, auch im Falle eines zukünftig weiterhin steigenden Umfangs der verteilten Funktionen.

Zusätzlich wurde in der Arbeit die zunehmende Arbeitsteilung in der Automobilentwicklung zu Lasten der Zulieferer berücksichtigt. So werden die Teilumfänge der zunehmend verteilten Systeme des Fahrzeugs immer mehr von verschiedenen Zulieferern erstellt. Für die Diagnose solcher Systeme ist umfassendes, detailliertes Wissen notwendig, was aber mit den Bestrebungen der Zulieferer, wettbewerbsrelevantes Wissen zu schützen, kollidiert. Um dieses Problem zu bewältigen und so die Diagnose verteilter Systeme zu verbessern, wurde in der Dissertation ein Datenmodell der Diagnose definiert, das durch Anwendung einer systematischen Qualitätsmethodik befüllt wird, um alle systematischen Fehler erfassen zu können. Als Qualitätsmethodik wurde die FMEA gewählt, die um identifizierte Potentiale erweitert wurde.

Durch die Definition eines neuen Entwicklungsprozesses mit der erweiterten FMEA und dem Datenmodell als Schnittstelle zwischen Zulieferer und OEM wurde eine Zuliefersicht ermöglicht, an der der Zulieferer die Diagnose seiner Teilumfänge am und im Zusammenspiel mit dem Gesamtsystem, beginnend in den frühen Entwicklungsphasen, prüfen und validieren kann. Um zudem die kontinuierliche Weiterentwicklung der Komponenten zu berücksichtigen, erstreckt sich der Prozeß über den

gesamten Lebenszyklus.

Methodik und Prozeß der Arbeit wurden anhand eines Fallbeispiels im Detail vorgestellt und anschließend evaluiert. Dabei wurde gezeigt, daß durch den vorgestellten Ansatz mitsamt Prozeß die identifizierten Potentiale gehoben werden und somit die Effizienz der Diagnose gesteigert werden kann.

8.2 Ausblick

Einsatz des Real Options-Verfahrens für die Maßnahmenbewertung

> „If you can look into the seeds of time, and say which grain will grow and which will not, speak then to me“
>
> *(William Shakespare - Macbeth)*

In Abschnitt 4.8.3 wurde eine Gewichtungsfunktion für die Reparaturmaßnahmen vorgestellt. Diese Funktion wird eingesetzt, um die Reparaturmaßnahmen im Lebenszyklus anhand ihrer Kosten und Wahrscheinlichkeit zu gewichten und ermöglicht so, die kostenoptimale Maßnahme zu wählen. Die Kosten bleiben wie erwähnt meist gleich, wodurch vorwiegend die Werte für die Wahrscheinlichkeit Änderungen unterworfen sind. Die Gewichtungsfunktion berücksichtigt aber nur Reparaturmaßnahmen beginnend mit der Produktion des Fahrzeugs.

Interessant für die Entwickler ist jedoch die Einbeziehung von Entdeckungs- und Fehlervermeidungsmaßnahmen zum Zeitpunkt der Entwicklung. Es wurde in Abschnitt 2.1.2 dargelegt, daß dem Einsatz solcher Maßnahmen Kostengrenzen gesetzt sind.

Die Idee ist nun, bei der Erstellung der erweiterten FMEA die Entdeckungs- und Vermeidungsmaßnahmen eines möglichen Fehlers mit den Maßnahmen zur Reparatur beim Auftreten des Fehlers sowohl hinsichtlich Kosten als auch Auftretenswahrscheinlichkeit sowie anderer Faktoren zu vergleichen. So kann beispielsweise das Verbauen eines Sensors oder die Hinzunahme mehrerer Meßpunkte im Steuergerät, um einen Fehler F_i zu vermeiden, mit den Kosten aller möglicher Reparaturen für F_i verglichen werden.

Eine solche Bewertung ist bei einer statischen Analyse meist nachteilig für die Vermeidungs- und Entdeckungsmaßnahmen (bzw. der Fehlervermeidung im Allgemeinen) aufgrund der Stückkosten/-zahl und Entwicklungskosten, vor allem da die Auftretenswahrscheinlichkeit initial nur geschätzt werden kann und die Fehlerkosten noch nicht vorliegen können. Im Lebenszyklus des Fahrzeugs und vor allem im Falle eines vermehrten Auftretens des betreffenden Fehlers sollte jedoch diese Entscheidung stets geprüft werden. Ein weiterer Grund hierfür ist die stetige Weiterentwicklung der Komponenten sowie die Übernahme von Komponenten in andere Fahrzeuge und Derivate.

Um also stets eine optimale Maßnahme identifizieren zu können, ist ein Ansatz notwendig, der sowohl den Kostenvergleich zwischen den alternativen Maßnahmen als auch die Unsicherheit über die Auftretensanzahl über einen längeren Zeitraum beginnend mit der Entwicklungsphase berücksichtigt. Es handelt sich also hierbei

um die Fragestellung, ob und wann ein OEM oder Zulieferer Geld in Form einer Maßnahme investiert, um nachher Kosten zu vermeiden oder einen höheren Return on Invest (ROI) erzielt.

Eine Analogie hierzu findet sich bei der Bewertung von Optionsgeschäften. Eine Option gewährt das Recht, einen Vermögenswert innerhalb eines definierten Zeitraums unter bestimmten Bedingungen zu kaufen, zu verkaufen oder die Option verfallen zu lassen [BS73, S. 637]. Entscheidend für den Umgang mit der Option ist die Differenz zwischen tatsächlichem Wert des Gegenstands und des Preises der Option. Ein fundamentaler Beitrag zur Berechnung des tatsächlichen Wertes einer Option stellt das Black-Scholes-Modell dar [BS73]. Dieses Modell wurde von [CRR79] zu einem diskretisierten Modell vereinfacht. Beispiele für Anwendungen des Ansatzes in der automobilen Domäne finden sich in [Axe00, Axe06].

Maßnahme	**Entwicklungskosten**	**Lebenszykluskosten**
Sensor verbauen	-1000 €	(-1,50 €) · Stückzahl
kein Sensor, RM_1	0 €	(-100 €) · Anzahl Reparaturen

Tabelle 8.1: Vereinfachtes Beispiel Maßnahmenbewertung. Quelle: eigene Darstellung

Tabelle 8.1 zeigt ein vereinfachtes Beispiel für den Vergleich zweier Maßnahmen für einen Fehler F_i, die durch die erweiterte FMEA bestimmt wurden. Der Fehler kann von einem verbauten Sensor entdeckt werden, der aber pro verbauter Komponente 1,50 € kostet sowie einmalig 1000 € Entwicklungskosten. Die andere Maßnahme besteht darin, den Fehler nicht zu vermeiden, sondern mit der Reparaturmaßnahme RM_1 zu reparieren, die aber 100 € pro Anwendung kostet. Die Entscheidung für eine der beiden Maßnahmen ist nun abhängig von der Stückzahl der Komponente, der Anzahl der Reparaturen sowie Erfolgsrate der Reparatur und der Fehlererkennung. Da alle diese Werte sich im Verlauf des Lebenszyklus deutlich ändern können, kann eine initiale Bewertung diese Änderungen nicht reflektieren.

Wie das Beispiel zeigte, ist das Ziel der Anwendung des Real Options-Ansatzes für die Diagnose eine vergleichende, stets aktualisierte Bewertung der Maßnahmen für mögliche Fehler. Die Maßnahmen werden durch das in Abschnitt 4.8 vorgestellte statistische Wartungsverfahren aktualisiert.

Zudem ermöglicht der Ansatz die Beantwortung einer der wichtigsten Fragestellungen der Qualitätsarbeit: die Gegenüberstellung der reaktiven und präventiven Qualität.

Der Real Options-Ansatz kann in das in der Dissertation eingebaute Datenmodell der Diagnose relativ leicht eingebaut werden. Notwendig hierfür sind die Erstellung eines umfassenden, detaillierten Kostenmodells sowie die Implementierung des Berechnungsalgorithmus.

Skalierbare Diagnosegenauigkeit

„Though this be madness, yet
there's method in't"

(William Shakespeare - Hamlet)

Die (erweiterte) FMEA ermöglicht die Erfassung aller systematischen Fehler einer Komponente bzw. eines Systems und eine Belegung der erfaßten Fehler mit Maßnahmen für das Vermeiden, Beheben sowie Erkennen der erfaßten Fehler. Dadurch werden alle Fehler mit Erkennungsmaßnahmen versehen und somit ein Höchstwert an Beobachtungen definiert.

Durch eine der zentralen Anforderungen der Diagnose, daß jeder Fehler einer kundenerlebbaren Funktion mittels DTC festgehalten werden muß (vgl. Tabelle 2.2), ergibt sich ein Mindestmaß an notwendigen Beobachtungen.

In Abbildung 1.2 sowie im vorigen Abschnitt wurde dargestellt, daß auch die Entwicklung von Fehlererkennungsmaßnahmen Geld kostet. Da die Entwicklungskosten für die off-board Diagnose stückzahlunabhängig und somit relativ konstant sind, ergibt sich die Fragestellung, wie gut die Fehlererkennung im Steuergerät sein muß.

Je mehr Beobachtungspunkte im Steuergerät vorhanden sind, desto genauer und feiner kann die Diagnose eine Fehlerursache entdecken und verschiedene Fehler durch die zusätzlichen Beobachtungen *diskriminieren*, aber desto mehr Ressourcen und Kosten sind notwendig. Diese Diskriminierung der genauen Fehlerursache ist für das Qualitätsmanagement des Lieferanten und OEM interessant, vor allem bei sich häufenden Fehlern und Reparaturen einer Komponente.

Dem gegenüber steht aber der schon erwähnte Fokus der Werkstatt auf tauschbare Einheiten aufgrund ihrer beschränkten Eingriffsmöglichkeiten. Für die Problembehebung in der Werkstatt macht es somit keinen großen Unterschied, was genau innerhalb der austauschbaren Einheit defekt ist, solange die fehlerhafte, tauschbare Einheit *detektierbar* ist.

Dieser Konflikt zwischen *Detektierbarkeit* und *Diskriminierbarkeit* wurde in mehreren Arbeiten schon untersucht, beispielsweise [DS03, Kap. 4 und 5], [Bau06, S.158ff], [TEO06], [CTA07] sowie [SP07]. In der Praxis ist es jedoch manchmal so, daß bei relativ simplen Komponenten weniger Diagnose wirtschaftlich gesehen besser ist.

Als Anknüpfungspunkt böte sich nun an, diese Fragestellung für nicht sicherheitsrelevante Komponenten umfassend unter einem wirtschaftlichen Aspekt zu betrachten. Die Idee ist, sowohl Anzahl und Umfang der Beobachtungen als auch die Granularität der austauschbaren Einheit kostenabhängig zu skalieren. Diese Fragestellung hängt somit direkt mit der dargestellten Kostenbewertung von Maßnahmen mittels des Real Options-Verfahrens zusammen.

Einbindung des Verfahrens in AUTOSAR

AUTOSAR wird von einem Konsortium spezifiziert, dem mehrere wichtige OEM (BMW, Daimler, Toyota, VW, ...) und Zulieferer (Bosch, ZF, ...) angehören, mit dem Ziel, eine neue offene Referenzarchitektur für automobile Steuergeräte bereitzustellen. Aufgrund der Unterstützung durch dieses Konsortium wird AUTOSAR in den

nächsten Jahren weitgehend in den Steuergeräten im Automobilbereich eingesetzt werden.

AUTOSAR ist eine geschichtete Architektur, die eine Reduktion der Komplexität der Entwicklung und Wiederverwendung ermöglichen soll. Abbildung 8.1 zeigt den Aufbau von AUTOSAR. Für eine detaillierte Erklärung der einzelnen Schichten sei auf [AUT09] verwiesen.

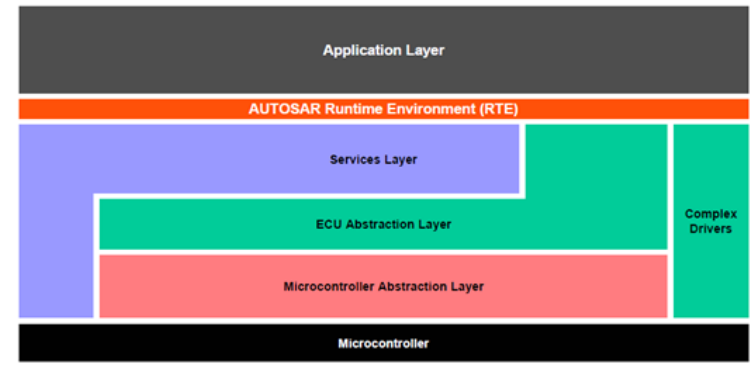


Abbildung 8.1: Schichtenmodell AUTOSAR. Quelle: aus [AUT09]

Vorgeschlagen wird die Einbindung der Software-Umfänge der Inferenz des SAT-Solvers in das Modul *Diagnosis Event Manager* (DEM) sowie die Einbindung der Fehlervermeidung in den *Functional Inhibition Manager* (FIM). Beide Module sind Teil des *Service Layer*, der die grundlegende Standardsoftware zur Verfügung stellt, die für alle Komponenten einsetzbar ist. Die Programmlogik der Fehlererkennung wird Teil des *application layer*, der die Programmlogik für jede Komponente enthält und somit je nach Komponente verschieden sein kann.

Im application layer wird festgelegt, was die Beobachtungen überwachen sowie der Zusammenhang zwischen Beobachtungen und möglichen Fehlern. Das Urteilen, ob und welcher Fehler vorliegt, gegeben bestimmte, gesammelte Beobachtungen, wird von den Inferenzfunktionen bzw. SAT-Solver im Service Layer basierend auf den ausgewerteten Beobachtungen übernommen.

Generische Software-FMEA

In Abschnitt 4.5.2 wurde vorgeschlagen, die erfaßten Fehler zu klassifizieren. Eine Klassifizierung aller Fehler, vor allem von Software-Fehlern, ergäbe die Möglichkeit, eine generische Software-FMEA zu erstellen, die um generische Entdeckungsmaßnahmen und Vermeidungsmaßnahmen erweitert wird. Beispielhaft hierfür ist die Definition einer generischen Plausibilisierungsprüfung. Das Thema generische Software-FMEA wurde schon in Abschnitt 3.9 angeschnitten.

Dies böte die Bereitstellung von Entdeckungs- und Vermeidungs-Pattern (vgl. [GHJV95, KGM+04, BKPS07]), die komponentenübergreifend eingesetzt werden können. Die Einbindung der Pattern in einen geschichteten Software-Architekturansatz wie AUTOSAR würde die Wiederverwendungsmöglichkeit zusätzlich erhöhen.

Dienstbasierte Architektur für die Diagnose

Die *dienstbasierte Architektur* bzw. *service-oriented architecture* (SOA) stellt ein Architekturparadigma dar, das zuerst 1996 in [SN96a, SN96b] vorgestellt wurde. Für SOA existieren mehrere Definitionen, worauf auch in [BKM07, S. 3] hingewiesen wird, so bspw. ein „paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains“ [MLM+06, Kap. 2.1].

In [BKM07] wird ein formales Modell für Dienste dargestellt. Beispiele für die Anwendung des Ansatzes für die Domäne Automobil finden sich in [KGM+04, EHK+07, EKM+07, EFF+08]. Obwohl [EHK+07, EKM+07, EFF+08] die SOA für die automobile Diagnose anpassen, fehlt gegenwärtig ein Ansatz, der die ganze Domäne der automobilen Diagnose abdeckt.

Dieser Ansatz muß sowohl die on-board Diagnose im Steuergerät, als auch die off-board Diagnose des Testers erfassen. Im Folgenden wird kurz skizziert, welche Vorteile ein solcher Ansatz böte.

Die SOA besteht aus mehreren gekapselten, lose gekoppelten Diensten, die vorwiegend asynchron über eine als Service-Infrastruktur bezeichnete Schnittstelle kommunizieren und so ihre Dienste anbieten (vgl. [SN96a, SN96b] sowie [Krc10, S.274]). Ziel dieses Aufbaus ist zum einen die Reduktion der Komplexität der ganzen Architektur und zum anderen die Erhöhung der Wiederverwendbarkeit der einzelnen Dienste. Die Komplexität des gesamten Systems wird reduziert, da gekapselte Dienste ausschließlich über eine Schnittstelle kommunizieren.

Die Fähigkeit der SOA, Elemente wiederzuverwenden, könnte so einen Beitrag zur Erstellung eines produktlinienübergreifenden Diagnosebaukastens leisten, was sich in einer deutlichen Reduzierung der Diagnoseerstellungskosten bemerkbar macht. Dieser Vorteil kann zudem im Zusammenspiel mit der vorgestellten, generischen Software-FMEA gesteigert werden.

Da das Gesamtsystem aus gekapselten, lose gekoppelten Bausteinen bzw. Diensten besteht, die ausschließlich über eine spezifizierte Schnittstelle kommunizieren, bietet diese Infrastruktur den Vorteil einer erhöhten Portabilität der einzelnen Dienste, da die logische Architektur von der Hardware entkoppelt wird. Somit sind die Dienste der Software-Diagnose von der Hardware-Plattform unabhängig und so wiederverwendbar.

Ein noch größerer Vorteil dieser Architektur stellt aber das relativ einfach mögliche Hinzufügen oder Ersetzen von einzelnen Diensten auch zur Laufzeit dar. So könnte der Tester in den Werkstätten mit umfangreichen Diagnosediensten befüllt werden, die dann zur Laufzeit im Falle einer Reparatur dynamisch eingesetzt werden können im Zusammenspiel mit dem Fahrzeuggesamtsystem. Bei den Diensten könnte es sich beispielsweise um Diagnosefunktionen zur Bearbeitung sporadischer oder seltener Fehler handeln, die aufgrund des begrenzten Speichers nicht auf das Steuergerät gespeichert werden. Weiterhin ließen sich so Fehler aufgrund seltener Systemausstattungen leichter nachvollziehen und diagnostizieren, da der Tester deren Funktionalitäten zur Laufzeit simulieren kann. Schließlich ermöglicht dies auch einem Zulieferer, die Diagnose seiner Systemumfänge am Gesamtsystem zu prüfen oder für eine Fehlersuche Fehler zu simulieren.

Literaturverzeichnis

[ABRW09] ANGERMANN, A., BEUSCHEL, M., RAU, M. und WOHLFARTH, U.: *Matlab - Simulink - Stateflow. Grundlagen, Toolboxen, Beispiele*. Oldenbourg Verlag, sechste Auflage, 2009. (Zitiert auf Seite 86)

[Air10] *Airbus Corporate Information: Orders & deliveries*. Airbus SAS – An EADS Company, April 2010. Verfügbar unter Link, zugegriffen am 01.09.2010. (Zitiert auf Seite 16)

[AIS93] AGRAWAL, R., IMIELIŃSKI, T. und SWAMI, A.: *Mining association rules between sets of items in large databases*. In: *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, SIGMOD '93, Seiten 207–216, New York, NY, USA, 1993. Association for Computing Machinery (ACM). (Zitiert auf Seiten 137 und 139)

[AKMP05] AHLUWALIA, J., KRÜGER, I. H., MEISINGER, M. und PHILLIPS, W.: *Model-Based Run-Time Monitoring of End-to-End Deadlines*. In: *Proceedings of the fifth ACM International Conference on Embedded Software (EMSOFT '05)*, Jersey City, NJ, USA, September 2005. Association for Computing Machinery (ACM). (Zitiert auf Seiten 94, 116 und 117)

[ALR01] AVIŽIENIS, A., LAPRIE, J.-C. und RANDELL, B.: *Fundamental Concepts of Dependability*. Technischer Bericht 010028, University of California Los Angeles, 2001. (Zitiert auf Seite 27)

[ALRL04] AVIŽIENIS, A., LAPRIE, J. C., RANDELL, B. und LANDWEHR, C.: *Basic Concepts and Taxonomy of Dependable and Secure Computing*. IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, 2004. (Zitiert auf Seiten 15, 27, 29, 30, 31, 32, 57, 98, 106 und 111)

[AP94] AAMODT, A. und PLAZA, E.: *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches*. Artificial Intelligence (AI) Communications, 7(1):39–59, März 1994. (Zitiert auf Seiten 45 und 80)

[AS94] AGRAWAL, R. und SRIKANT, R.: *Fast Algorithms for Mining Association Rules*. In: *Proceedings of the 20th International Conference on Very Large Data Bases*, Seiten 487–499, 1994. (Zitiert auf Seiten 137 und 139)

[ASA04] *ASAM MCD-2 D: Data Model for ECU Diagnostics*. Association for Standardisation of Automation and Measuring Systems (ASAM), 2004. Standardisiert

2004. (Zitiert auf Seite 89)

[Aud10] *Audi Geschäftsbericht 2009*. Audi AG, Februar 2010. Veröffentlicht am 10.02.2010. Verfügbar unter Link, abgerufen am 20.12.2010. (Zitiert auf Seiten 5 und 6)

[Aut08] AUTOMOTIVE INDUSTRY ACTION GROUP (AIAG) (Herausgeber): *Potential Failure Mode and Effects Analysis (FMEA)*. AIAG, vierte Auflage, Juni 2008. (Zitiert auf Seiten 99, 100, 102 und 103)

[AUT09] *Automotive Open System Architecture (AUTOSAR) Release 4.0*. AUTOSAR Development Cooperation, November 2009. Verfügbar unter Link, abgerufen am 20.12.2010. (Zitiert auf Seiten 141 und 190)

[Avi67] AVIŽIENIS, A.: *Design of Fault-Tolerant Computers*. In: THOMPSON, D. C. (Herausgeber): *Proceedings of the American Federation of Information Processing Societies (AFIPS) Fall Joint Computer Conference*, Band 31, Seiten 733–743, Washington, D. C., 1967. (Zitiert auf Seiten 27, 57 und 111)

[AW91] ALTHOFF, K.-D. und WESS, S.: *Case-Based Knowledge Acquisition, Learning and Problem Solving in Diagnostic Real World Tasks*. In: *Proceedings of the European Knowledge Acquisition Workshop (EKAW '91)*, Seiten 48–67, 1991. (Zitiert auf Seiten 45 und 80)

[Axe00] AXELSSON, J.: *Cost Models for Electronic Architecture Trade Studies*. IEEE International Conference on Engineering of Complex Computer Systems, Seiten 229–239, 2000. (Zitiert auf Seite 188)

[Axe06] AXELSSON, J.: *Cost Models with Explicit Uncertainties for Electronic Architecture Trade-off and Risk Analysis*. Technischer Bericht, Mälardalen University, Schweden, 2006. (Zitiert auf Seite 188)

[Bal98] BALZERT, H.: *Lehrbuch der Software-Technik. Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Spektrum Akademischer Verlag, 1998. (Zitiert auf Seiten 85, 86, 87, 181 und 183)

[Bau06] BAUER, A.: *Model-based runtime analysis of distributed reactive systems*. Doktorarbeit, Technische Universität München, 2006. (Zitiert auf Seiten 49, 50, 51, 54, 55, 82, 83, 114, 115 und 189)

[BBSS97] BROY, M., BREITLING, M., SCHÄTZ, B. und SPIES, K.: *Summary of Case Studies in Focus - Part II*. Technischer Bericht TUM-I9740, Technische Universität München, 1997. (Zitiert auf Seite 142)

[BCG87] BERRY, G., COURONNE, P. und GONTHIER, G.: *Synchronous programming of reactive systems: an introduction to ESTEREL*. Technischer Bericht 647, Institut National de Recherche en Informatique et Automatique (INRIA) Sophia-Antipolis, 1987. (Zitiert auf Seite 57)

[BCJS92] BELL, D., COX, L., JACKSON, S. und SCHAEFER, P.: *Using causal reasoning for automated failure modes and effects analysis (FMEA)*. In: *Annual Reliability and Maintainability Symposium*, Seiten 343–353, 1992. (Zitiert auf Seite 104)

[Bec05] BECKER, H.: *Auf Crashkurs. Automobilindustrie im globalen Verdrängungswettbewerb*. Springer Verlag, 2005. (Zitiert auf Seiten 7, 16, 17 und 68)

[Ber89] BERRY, G.: *Real Time Programming: Special Purpose or General Purpose Languages*. Technischer Bericht 1065, Institut National de Recherche en Informatique et Automatique (INRIA) Sophia-Antipolis, August 1989. (Zitiert auf Seiten 14 und 57)

[BF94] BIRSCH, D. und FIELDER, J. (Herausgeber): *The Ford Pinto Case: Study in Applied Ethics, Business and Technology*. State University of New York, 1994. (Zitiert auf Seite 99)

[BFG+08] BROY, M., FEILKAS, M., GRÜNBAUER, J., GRULER, A., HARHURIN, A., HARTMANN, J., PENZENSTADLER, B., SCHÄTZ, B. und WILD, D.: *Umfassendes Architekturmodell für das Engineering eingebetteter Softwareintensiver Systeme. Modellierungstheorien und Architekturebenen*. Technischer Bericht TUM-I0816, Technische Universität München, 2008. (Zitiert auf Seiten 93 und 141)

[BFOS84] BREIMAN, L., FRIEDMAN, J. H., OLSHEN, R. A. und STONE, C. J.: *Classification and regression trees*. Chapman and Hall, 1984. (Zitiert auf Seite 137)

[BGK+09] BROY, M., GLEIRSCHER, M., KLUGE, P., KRENZER, W., MERENDA, S. und WILD, D.: *Automotive Architecture Framework: Towards a Holistic and Standardised System Architecture Description*. Technischer Bericht TUM-I0915, Technische Universität München, Juli 2009. (Zitiert auf Seiten 69, 93, 116, 141 und 181)

[BH96] BREESE, J. M. und HECKERMAN, D.: *Decision-theoretic troubleshooting: A framework for repair and experiment*. In: *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, Seiten 124 – 132, Portland, Oregon, August 1996. (Zitiert auf Seite 45)

[BHS99] BROY, M., HUBER, F. und SCHÄTZ, B.: *AutoFocus – Ein Werkzeugprototyp zur Entwicklung eingebetteter Systeme*. Informatik Forschung und Entwicklung, 14:121–134, 1999. (Zitiert auf Seiten 86 und 142)

[Bir99] BIROLINI, A.: *Reliability Engineering - Theory and Practice*. Springer Verlag, dritte Auflage, 1999. (Zitiert auf Seiten 27, 28 und 29)

[BK02] BREIING, A. J. und KUNZ, A. M.: *Critical consideration and improvement of the FMEA*. In: *Fourth International Symposium on Tools and Methods of Competitive Engineering (TMCE '02)*, Seiten 519 – 530, Wuhan, China, April 2002. (Zitiert auf Seite 102)

[BKM07] BROY, M., KRÜGER, I. H. und MEISINGER, M.: *A formal model of services*. ACM Transmissions on Software Engineering Methodology (TOSEM), 16, Februar 2007. (Zitiert auf Seite 191)

[BKPS07] BROY, M., KRÜGER, I. H., PRETSCHNER, A. und SALZMANN, C.: *Engineering automotive software*. Proceedings of the Institute of Electrical and Electronics Engineers (IEEE), 95(2):356–373, 2007. (Zitiert auf Seiten 3 und 190)

[BM88] BEDNARZ, S. und MARRIOTT, D.: *Efficient analysis for FMEA*. In: *Proceedings of the IEEE Annual Reliability and Maintainability Symposium*, Seiten 416–421, 1988. (Zitiert auf Seite 102)

[BMU75] BOEHM, B. W., MCCLEAN, R. K. und URFRIG, D. B.: *Some experience with automated aids to the design of large-scale reliable software*. ACM Special Interest Group on Programming Languages (SIGPLAN) Notices - International Conference on Reliable Software, 10:105–113, April 1975. (Zitiert auf Seite 181)

[BMW06] *BMW Produktentstehungsprozeß (PEP)*. BMW Group, März 2006. Interner Standard der BMW Group. (Zitiert auf Seiten 40, 63, 145, 149, 214, 215, 218 und 229)

[BMW07] *BMW Group Standard Embedded Software Entwicklung*. BMW Group, April 2007. Interner Standard der BMW Group. (Zitiert auf Seiten 63, 145, 151, 214, 218 und 229)

[BMW08] *Lastenheft Diagnose: allgemeiner Teil*. BMW Group, April 2008. Internes Lastenheft der BMW Group. (Zitiert auf Seiten 37, 38, 39, 40, 41, 146, 218 und 229)

[BMW10] *Jahresabschluß der BMW AG 2009*. BMW Group, Februar 2010. Veröffentlicht am 26.02.2010. Verfügbar unter Link, abgerufen am 20.12.2010. (Zitiert auf Seiten 5 und 6)

[Bos91] *CAN Specification Version 2.0*. Robert Bosch GmbH, 1991. (Zitiert auf Seiten 3 und 21)

[BPS+08] BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M., MALER, E. und YERGEAU, F. (Herausgeber): *Extensible Markup Language (XML) 1.0 - W3C Recommendation*. World Wide Web Consortium (W3C), fünfte Auflage, November 2008. Verfügbar unter Link, abgerufen am 23.12.2010. (Zitiert auf Seite 123)

[BR96] BEN-DAYA, M. und RAOUF, A.: *A revised failure mode and effects analysis model*. International Journal of Quality & Reliability Management, 13(1):43–47, 1996. (Zitiert auf Seite 102)

[Bre01] BREITLING, M.: *Formale Fehlermodellierung für verteilte reaktive Systeme*. Doktorarbeit, Technische Universität München, 2001. (Zitiert auf Seiten 28, 30, 31 und 59)

[Bro02] BROOKSBY, R. AND BARNES, N.: *The Memory Pool System - Thirty person-years of memory management development goes Open Source*. 2002. Verfügbar unter Link, abgerufen am 08.08.2011. (Zitiert auf Seite 132)

[Bro03] BROY, M.: *Automotive Software Engineering*. In: *Proceedings of the 25th International Conference on Software Engineering (ICSE 2003)*, Seiten 719–720. Institute of Electrical and Electronics Engineers (IEEE) Computer Society, 2003. (Zitiert auf Seite 4)

[Bro06] BROY, M.: *Challenges in Automotive Software Engineering*. In: *International Conference on Software Engineering (ICSE 2006)*, Seiten 33–42, Shanghai, China, Mai 2006. Association for Computing Machinery (ACM). (Zitiert auf Seiten 4 und 93)

[BS73] BLACK, F. und SCHOLES, M.: *The Pricing of Options and Corporate Liabilities*. The Journal of Political Economy, 81(3):637–654, 1973.

(Zitiert auf Seite 188)

[BS84] BUCHANAN, B. G. und SHORTLIFFE, E. H. (Herausgeber): *Rule-Based Expert Systems - The MYCIN Experiments of the Stanford Heuristic Programming Project*, Band 1 der Reihe *The Addison-Wesley Series in Artificial Intelligence*. Addison-Wesley, 1984. (Zitiert auf Seiten 41 und 42)

[BS91] BOUSSINOT, F. und SIMONE, R. DE: *The ESTEREL language*. Proceedings of the Institute of Electrical and Electronics Engineers (IEEE), 79(9):1293–1304, 1991. (Zitiert auf Seite 57)

[BS01] BROY, M. und STØLEN, K.: *Specification and Development of Interactive Systems - FOCUS on Streams, Interfaces, and Refinement*. Monographs in Computer Science. Springer Verlag, 2001. (Zitiert auf Seiten 117, 118 und 142)

[BS02] BRAUN, P. und SLOTOSCH, O.: *Development of a Car Seat: A Case Study using AutoFocus, DOORS, and the Validas Validator*. In: *OMER - Object-oriented Modeling of Embedded Real-Time Systems, LNI P-5*. Springer Verlag, 2002. (Zitiert auf Seite 142)

[BSML08] BLUMENSTOCK, A., SCHWEIGGERT, F., MÜLLER, M. und LANQUILLON, C.: *Rule cubes for causal investigations*. Knowledge and Informations Systems, 18(1):109–132, 2008. (Zitiert auf Seite 142)

[Bun09] BUNDESMINISTERIUM DER JUSTIZ (Herausgeber): *Handelsgesetzbuch (HGB)*. Bundesministerium der Justiz, 2009. Zuletzt geändert durch Art. 6a G vom 31.7.2009 I 2512. Verfügbar unter Link, abgerufen am 08.08.2008. (Zitiert auf Seite 6)

[Bun10] BUNDESMINISTERIUM DER JUSTIZ (Herausgeber): *Bürgerliches Gesetzbuch (BGB)*. Bundesministerium der Justiz, 2010. Bekanntmachung vom 2. Januar 2002 (BGBl. I S. 42, 2909; 2003 I S. 738), letzte Änderung durch Artikel 1 des Gesetzes vom 24. Juli 2010 (BGBl. I S. 977). (Zitiert auf Seiten 6, 18 und 35)

[BW82] BAUER, F. L. und WÖSSNER, H.: *Algorithmic Language and Program Development*. Texts and monographs in Computer Science. Springer Verlag, 1982. (Zitiert auf Seiten 77 und 182)

[BW01] BOWLES, J.B. und WAN, C.: *Software failure modes and effects analysis for a small embedded control system*. In: *Proceedings of the Annual Reliability and Maintainability Symposium*, Seiten 1–6. Institute of Electrical and Electronics Engineers (IEEE), 2001. (Zitiert auf Seiten 88 und 99)

[BZ09] BUDDHAKULSOMSIRI, J. und ZAKARIAN, A.: *Sequential pattern mining algorithm for automotive warranty data*. Computers & Industrial Engineering, 57(1):137–147, 2009. (Zitiert auf Seite 142)

[Cal94] CALIFORNIA AIR RESOURCES BOARD (CARB): *Malfunction and Diagnostic System Requirements – 1994 and Subsequent Model-Year Passenger Cars, Light-Duty Trucks, and Medium-Duty Vehicles and Engines (OBD II)*. In: *California Code of Regulations (CCR), Title 13, Section 1968.1*, 1994. (Zitiert auf Seiten 4, 34 und 123)

[Cal10] CALIFORNIA AIR RESOURCES BOARD (CARB): *The California Low-*

Emission Vehicle Regulations for Passenger Cars, Light-Duty Trucks and Medium-Duty Vehicles. In: *California Code of Regulations*, April 2010. Verfügbar unter Link, abgerufen am 07.08.2010. (Zitiert auf Seite 3)

[Cas08] CASSANDRAS, C. AND LAFORTUNE, S.: *Introduction to Discrete Event Systems*. Springer Verlag, zweite Auflage, 2008. (Zitiert auf Seite 46)

[CD99] CLARK, J. und DEROSE, S. (Herausgeber): *XML Path Language (XPath) Version 1.0 - Recommendation*. World Wide Web Consortium (W3C), November 1999. Verfügbar unter Link, abgerufen am 23.12.2010. (Zitiert auf Seite 123)

[CDK05] COULOURIS, G., DOLLIMORE, J. und KINDBERG, T.: *Distributed Systems - Concepts and Design*. Addison-Wesley, vierte Auflage, 2005. (Zitiert auf Seiten 13 und 32)

[CES71] COFFMAN, E. G., ELPHICK, M. J. und SHOSHANI, A.: *System Deadlocks*. Computing Surveys, 3:67–78, 1971. (Zitiert auf Seite 14)

[Coo71] COOK, S. A.: *The complexity of theorem-proving procedures*. In: *Proceedings of the third annual ACM symposium on theory of computing (STOC '71)*, Seiten 151–158, New York, NY, USA, 1971. Association for Computing Machinery (ACM). (Zitiert auf Seiten 74 und 129)

[CRR79] COX, J., ROSS, S. und RUBINSTEIN, M.: *Option pricing: A simplified approach*. Journal of Financial Economics, 7(3):229 – 263, 1979. (Zitiert auf Seite 188)

[CTA07] CASSEZ, F., TRIPAKIS, S. und ALTISEN, K.: *Sensor Minimization Problems with Static and Dynamic Observers for Fault Diagnosis*. In: *Proceedings of the seventh International Conference on Application of Concurrency to System Design (ACSD 2007)*, 2007. (Zitiert auf Seite 189)

[CTS02] CUNIS, R., TECHNOW, R. und SEIZMAIR, M.: *Überblick über Diagnosemethoden und ihre Eignung für die Fahrzeugdiagnose*. Technischer Bericht, ServiceXpert GmbH, 2002. Interne Studie der BMW Group. (Zitiert auf Seite 77)

[Dai00] DAIS, S.: *Elektronik im Kraftfahrzeug - ein Beitrag zur Aufrechterhaltung der Mobilität*. In: *VDI Berichte*, Band 1547, Seiten 3 – 15. Verein Deutscher Ingenieure (VDI), 2000. (Zitiert auf Seite 3)

[Dai10] *Daimler Geschäftsbericht 2009*. Daimler AG, März 2010. Veröffentlicht am 03.03.2010. Verfügbar unter Link, abgerufen am 20.12.2010. (Zitiert auf Seiten 5 und 6)

[Deg00] DEGELE, N.: *Informiertes Wissen - eine Wissenssoziologie der computerisierten Gesellschaft*. Campus Verlag, 2000. (Zitiert auf Seiten 88 und 134)

[Dij68] DIJKSTRA, E. W.: *Cooperating sequential processes*. In: GENUYS, F. (Herausgeber): *Programming Languages: NATO Advanced Study Institute*, Seiten 43–112. Academic Press, 1968. (Zitiert auf Seite 162)

[DIM93] *Satisfiability and Maximum Satisfiability. Descriptions, Readings, Problems*. Center for Discrete Mathematics and Theoretical Computer Science (DIMACS), Rutgers State University, Mai 1993. Verfügbar unter Link, abgerufen am

20.12.2010. (Zitiert auf Seiten 124 und 125)

[DIN70] *DIN 40042: 1970 - 6. Zuverlässigkeit elektrischer Geräte, Anlagen und System - Begriffe*. Deutsches Institut für Normung (DIN) e. V., Juni 1970. Norm wurde März 1986 zurückgezogen. (Zitiert auf Seite 57)

[DIN71] *DIN 72552-2: Klemmenbezeichnungen in Kraftfahrzeugen, Bedeutungen*. Deutsches Institut für Normung (DIN) e. V., März 1971. (Zitiert auf Seite 123)

[DIN80] *DIN 25448 - Ausfalleffektanalyse (Fehlermöglichkeits- und Einflußanalyse)*. Deutsches Institut für Normung (DIN) e. V., Mai 1980. (Zitiert auf Seiten 99, 100 und 119)

[DIN90] *DIN 40041 1990-12-00 . Zuverlässigkeit - Begriffe*. Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE, 1990. (Zitiert auf Seiten 57 und 58)

[DIN94] *DIN 19226-1: Regelungstechnik und Steuerungstechnik: Allgemeine Grundbegriffe*. Deutsches Institut für Normung (DIN) e. V., 1994. (Zitiert auf Seite 21)

[DIN06] *Analysetechniken für die Funktionsfähigkeit von Systemen – Verfahren für die Fehlzustandsart- und -auswirkungsanalyse (FMEA)*. Deutsches Institut für Normung (DIN) e. V., November 2006. Aktualisierung der Norm DIN 25448. (Zitiert auf Seiten 20, 67 und 99)

[DK04] DANNENBERG, J. und KLEINHANS, C.: *The Coming Age of Collaboration in the Automotive Industry*. Mercer Management Journal, 18:88 – 94, 2004. (Zitiert auf Seiten 3, 5 und 68)

[DLL62] DAVIS, M., LOGEMANN, G. und LOVELAND, D.: *A machine program for theorem-proving*. Communications of the ACM, 5(7):394–397, 1962. (Zitiert auf Seiten 55 und 129)

[DP60] DAVIS, M. und PUTNAM, H.: *A Computing Procedure for Quantification Theory*. Journal of the ACM (JACM), 7(3):201–215, 1960. (Zitiert auf Seiten 55, 129 und 130)

[DS03] DRESSLER, O. und STRUSS, P.: *A Toolbox Integrating Model-based Diagnosability Analysis and Automated Generation of Diagnostics*. In: *Proceedings of the 14th International Workshop on Principles of Diagnosis (DX '03)*, Seiten 99 – 104, Washington, D.C., USA, Juni 2003. (Zitiert auf Seite 189)

[EAD10] *EAST-ADL Domain Model Specification*. Advancing Traffic Efficiency and Safety through Software Technology (ATESST) Consortium, Juni 2010. Verfügbar unter Link, abgerufen am 20.12.2010. (Zitiert auf Seiten 20, 93, 95, 141 und 142)

[ECM91] *A reference model for frameworks of computer aided software engineering environments*. Technischer Bericht TR/55, European Computer Manufacturers Association (ECMA), 1991. (Zitiert auf Seite 85)

[Edl01] EDLER, A.: *Nutzung von Felddaten in der qualitätsgetriebenen Produktentwicklung und im Service*. Doktorarbeit, Technische Universität Berlin, 2001. (Zitiert auf Seiten 19, 36, 66, 87 und 134)

[EFF+08] ERMAGAN, V., FARCAS, C., FARCAS, E., KRÜGER, I. H. und ME-

NARINI, M.: *A Service-Oriented Approach to Failure Management*. In: *Proceedings of the Dagstuhl Workshop on Model-Based Development of Embedded Systems (MBEES '08)*, April 2008. (Zitiert auf Seite 191)

[EG95] EITER, T. und GOTTLOB, G.: *Identifying the minimal transversals of a hypergraph and related problems*. Society for Industrial and Applied Mathematics (SIAM) Journal on Computing, 24(6):1278–1304, 1995. (Zitiert auf Seite 82)

[EHK+07] ERMAGAN, V., HUANG, T.-J., KRÜGER, I. H., MEISINGER, M., MENARINI, M. und MOORTHY, P.: *Towards Tool Support for Service-Oriented Development of Embedded Automotive Systems*. In: *Proceedings of the Dagstuhl Workshop on Model-Based Development of Embedded Systems (MBEES '07)*, 2007. (Zitiert auf Seiten 90, 116, 126 und 191)

[EKL07] EHRLENSPIEL, K., KIEWERT, A. und LINDEMANN, U.: *Kostengünstig Entwickeln und Konstruieren. Kostenmanagement bei der integrierten Produktentwicklung*. Springer Verlag, sechste Auflage, 2007. (Zitiert auf Seite 85)

[EKM+07] ERMAGAN, V., KRÜGER, I. H., MENARINI, M., MIZUTANI, J. -I., OGUCHI, K. und WEIR, D.: *Towards Model-Based Failure Management for Automotive Software*. In: *Proceedings of the fourth International Workshop on Software Engineering for Automotive Systems (SEAS '07)*, Seite 8. IEEE Computer Society, Mai 2007. (Zitiert auf Seiten 90, 116, 126, 127, 128 und 191)

[ER03] ENDRES, A. und ROMBACH, D.: *A Handbook of Software and Systems Engineering*. The Fraunhofer Institut für Experimentelles Software Engineering (IESE) Series on Software Engineering. Pearson Addison-Wesley, 2003. (Zitiert auf Seiten 77, 88, 134, 181 und 182)

[ES04a] EIGNER, M. und STELZER, R.: *Product Lifecycle Management: Ein Leitfaden für Product Development und Life Cycle Management*. Springer Verlag, zweite Auflage, 2004. (Zitiert auf Seite 89)

[ES04b] EÉN, N. und SÖRENSSON, N.: *An Extensible SAT-Solver*. In: GIUNCHIGLIA, E. und TACCHELLA, A. (Herausgeber): *Theory and Applications of Satisfiability Testing*, Band 2919 der Reihe *Lecture Notes in Computer Science*, Seiten 333–336. Springer Verlag Berlin/ Heidelberg, 2004. (Zitiert auf Seiten 55, 130 und 131)

[ETA10] *Advanced Simulation and Control Engineering Tool (ASCET)*. Engineering Tools, Application and Services (ETAS), 2010. Verfügbar unter Link, abgerufen am 20.12.2010. (Zitiert auf Seite 86)

[Eur99] EUROPÄISCHES PARLAMENT UND DER RAT DER EUROPÄISCHEN UNION (Herausgeber): *Richtlinie 1999/44/EG des Europäischen Parlaments und des Rates vom 25. Mai 1999 zu bestimmten Aspekten des Verbrauchsgüterkaufs und der Garantien für Verbrauchsgüter*. Nummer L171 in *Amtsblatt der Europäischen Union*. Europäisches Parlament und der Rat der Europäischen Union, April 1999. Veröffentlicht am 25.05.1999 mit Wirkung zum 07.07.1999. (Zitiert auf Seite 34)

[Eur09] EUROPÄISCHES PARLAMENT UND DER RAT DER EUROPÄISCHEN UNION (Herausgeber): *Verordnung (EG) Nr. 443/2009 des Europäischen Parlaments und des Rates vom 23. April 2009 zur Festsetzung von Emissionsnormen für*

neue Personenkraftwagen im Rahmen des Gesamtkonzepts der Gemeinschaft zur Verringerung der CO2-Emissionen von Personenkraftwagen und leichten Nutzfahrzeugen. Nummer L 140/1 in *Amtsblatt der Europäischen Union*. Europäisches Parlament und der Rat der Europäischen Union, April 2009. Veröffentlicht am 23.04.2009 mit Wirkung zum 08.06.2009. (Zitiert auf Seite 3)

[FBL70] FEIGENBAUM, E. A., BUCHANAN, B. G. und LEDERBERG, J.: *On generality and problem solving: a case study using the DENDRAL program*. Technischer Bericht, Stanford University, Stanford, CA, USA, 1970. (Zitiert auf Seite 41)

[Fei77] FEIGENBAUM, E. A.: *The art of artificial intelligence: themes and case studies of knowledge engineering*. In: *Proceedings of the fifth international joint conference on Artificial intelligence*, Band 2, Seiten 1014–1029, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc. (Zitiert auf Seite 41)

[FME80] *Military Standard procedures for performing a Failure Mode, Effects and Criticality Analysis (MIL-STD-1629A)*. Department of Defense (DOD), November 1980. Ersetzt MIL-STD-1629 (SHIPS) vom 1. November 1974 sowie MIL-STD-2070 (AS) vom 12. Juni 1977. (Zitiert auf Seiten 20 und 98)

[FP95] FUCHS, M. und PHILIPPS, J.: *Formal Development of a Production Cell in Focus – A Case Study*. In: LEWERENTZ, C. und LINDNER, T. (Herausgeber): *Formal Development of reactive systems*, Band 891 der Reihe *Lecture Notes in Computer Science*, Seiten 185–197. Springer Verlag Berlin/ Heidelberg, 1995. (Zitiert auf Seite 142)

[Fra09] FRARACCI, A.: *Model-based failure-modes-and-effects analysis and its application to aircraft subsystems*. Dissertationen zur künstlichen Intelligenz Nr. 326, Technische Universität München, 2009. (Zitiert auf Seiten 81 und 90)

[GGYA03] GANAI, M., GUPTA, A., YANG, Z. und ASHAR, P.: *Efficient Distributed SAT and SAT-Based Distributed Bounded Model Checking*. In: *Correct Hardware Design and Verification Methods*, Band 2860 der Reihe *Lecture Notes in Computer Science*, Seiten 334–347. Springer Verlag Berlin/ Heidelberg, 2003. (Zitiert auf Seite 182)

[GHJV95] GAMMA, E., HELM, R., JOHNSON, R. und VLISSIDES, J.: *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995. (Zitiert auf Seiten 126 und 190)

[Gil93] GILCHRIST, W.: *Modelling failure modes and effects analysis*. International Journal of Quality & Reliability Management, 10(5), 1993. (Zitiert auf Seite 103)

[God00] GODDARD, P. L.: *Software FMEA Techniques*. In: *Proceedings of the Annual Reliability and Maintainability Symposium 2000*. Institute of Electrical and Electronics Engineers (IEEE), 2000. (Zitiert auf Seite 99)

[GPFW96] GU, J., PURDOM, W. P., FRANCO, J. und WAH, B. W.: *Algorithms for the Satisfiability (SAT) problem: a survey*. In: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Seiten 19 – 152. American Mathematical Society (AMS), 1996. (Zitiert auf Seiten 8, 74, 83, 129 und 182)

[Hal93] HALBWACHS, N.: *Synchronous Programming of Reactive Systems*. Kluwer

Academic Publishers, 1993. (Zitiert auf Seiten 13, 14, 15 und 57)

[Hal98] HALBWACHS, NICOLAS: *Synchronous Programming of Reactive Systems: A tutorial and commented bibliography*. In: HU, A. und VARDI, M. (Herausgeber): *Computer Aided Verification*, Band 1427 der Reihe *Lecture Notes in Computer Science*, Seiten 1–16. Springer Verlag Berlin/ Heidelberg, 1998. (Zitiert auf Seiten 14, 15 und 57)

[Hay85] HAYES-ROTH, F.: *Rule-based Systems*. Communications of the ACM, 28(9):921 – 932, September 1985. (Zitiert auf Seite 43)

[HBH+98] HORVITZ, E. J., BREESE, J., HECKERMAN, D., HOVEL, D. und ROMMELSE, K.: *The Lumière project: Bayesian user modeling for inferring the goals and needs of software users*. In: *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, Seiten 256–265, Madison, Wisconsin, Juli 1998. (Zitiert auf Seite 45)

[HCK92] HAMSCHER, W., CONSOLE, L. und KLEER, J. DE (Herausgeber): *Readings in Model-Based Diagnosis*. Morgan Kaufmann Publishers Inc., 1992. (Zitiert auf Seite 59)

[HDS95] HANLE, A., DROSDOWSKI, G. und SCHOLZE-STUBENRECHT, W. (Herausgeber): *Brockhaus Enzyklopädie*. F.A. Brockhaus Verlag Mannheim, 19. Auflage, 1995. (Zitiert auf Seite 26)

[Hec85] HECKERMAN, D.: *Probabilistic Interpretations for MYCIN's Certainty Factors*. In: *Proceedings of the First Conference on Uncertainty in Artificial Intelligence (UAI '85)*, Seiten 9–20, New York, NY, Juli 1985. (Zitiert auf Seite 43)

[HH02] HAAPANEN, P. und HELMINEN, A.: *Failure Mode and Effects Analysis of Software-Based Automation Systems*. Technischer Bericht STUK-YTO-TR 190, Technical Research Centre of Finland (VTT) und Radiation and Nuclear Safety Authority Finland (STUK), August 2002. (Zitiert auf Seite 99)

[HHNZ06] HOCH, D. J., HUHN, W., NÄHER, U. und ZIELKE, A. E.: *The Race to Master Automotive Embedded Systems Development*. In: *McKinsey's Global Embedded Systems Initiative*. McKinsey Publications, 2006. (Zitiert auf Seiten 3, 5, 6 und 7)

[Him78] HIMMELBLAU, D. M.: *Fault Detection and Diagnosis in Chemical and Petrochemical Processes*, Band 8 der Reihe *Chemical Engineering Monographs*. Elsevier Scientific Publishing Company, 1978. (Zitiert auf Seite 89)

[HKK04] HARDUNG, B., KÖLZOW, T. und KRÜGER, A.: *Reuse of software in distributed embedded automotive systems*. In: *Proceedings of the fourth ACM international conference on Embedded software*, EMSOFT '04, Seiten 203–210, Pisa, Italien, September 2004. Association for Computing Machinery (ACM). (Zitiert auf Seiten 5, 6, 181 und 183)

[HMDJ08] HUANG, Y., MCMURRAN, R., DHADYALLA, G. und JONES, R. P.: *Probability based vehicle fault diagnosis: Bayesian network method*. Journal of Intelligent Manufacturing, 19:301–311, 2008. (Zitiert auf Seite 45)

[Hoa04] HOARE, C. A. R.: *Communicating Sequential Processes*. Prentice-Hall International, 2004. Verfügbar unter Link, abgerufen am 20.12.2010.

(Zitiert auf Seite 118)

[HP85] HAREL, D. und PNUELI, A.: *On the development of Reactive Systems*. In: APT, K. R. (Herausgeber): *NATO Advanced Science Institutes (ASI) Series on Logics and Models of Concurrent Systems*, Band F13, Seiten 477–499. Springer Verlag, 1985. (Zitiert auf Seiten 13 und 57)

[HSF+04] HEINECKE, H., SCHNELLE, K.-P., FENNEL, H., BORTOLAZZI, J., LUNDH, L., LEFLOUR, J., MATÉ, J. - L., NISHIKAWA, K. und SCHARNHORST, T.: *AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures*. Technischer Bericht 2004-21-0042, Convergence Transportation Electronics Association, 2004. (Zitiert auf Seite 89)

[HSSS96] HUBER, F., SCHÄTZ, B., SCHMIDT, A. und SPIES, K.: *AutoFocus – A tool for distributed systems specification*. In: JONSSON, B. und PARROW, J. (Herausgeber): *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Band 1135 der Reihe *Lecture Notes in Computer Science*, Seiten 467–470. Springer Verlag Berlin/ Heidelberg, 1996. (Zitiert auf Seiten 86 und 142)

[Huf94] HUFF, D.: *How to Lie with Statistics*. W. W. Norton & Company, 1994. (Zitiert auf Seite 137)

[IC93] IOANNOU, P. und CHIEN, C.: *Autonomous Intelligent Cruise Control*. IEEE Transactions on Vehicular Technology, 42(4):657–672, 1993. (Zitiert auf Seite 13)

[IEC98] *IEC 61508 - Functional Safety of Electrical/ Electronic/ Programmable Electronic Safety-Related Systems*. International Electrotechnical Comission (IEC), 1998. (Zitiert auf Seite 17)

[Int08] INTERNATIONAL ACCOUNTING STANDARDS BOARD (IASB) (Herausgeber): *International Financial Reporting Standards (IFRS) 37: Rückstellungen, Eventualschulden und Eventualforderungen*. IASB, 2008. Verfügbar unter Link oder Link, abgerufen am 20.12.2010. (Zitiert auf Seite 6)

[Ise94] ISERMANN, R. (Herausgeber): *Überwachung und Fehlerdiagnose. Moderne Methoden und ihre Anwendungen bei technischen Systemen*. VDI-Verlag, 1994. (Zitiert auf Seiten 52, 53, 54 und 59)

[Ise04] ISERMANN, R.: *Model-Based Fault Detection and Diagnosis - Status and Applications*. In: *Proceedings of the 16th International Federation of Automatic Control (IFAC) Symposium on Automatic Control in Aerospace*, Seiten 71–85, 2004. (Zitiert auf Seiten 52, 54 und 59)

[ISO90] *IEEE 610.12-1990: IEEE Standard Glossary of Software Engineering Technology*. International Organization for Standardization (ISO), 1990. (Zitiert auf Seite 12)

[ISO94a] *ISO 10303-1:1994 Industrial automation systems and integration – Product data representation and exchange - Part 1: Overview and Fundamental Principles*. International Organization for Standardization (ISO), 1994. International Standard ISO TC184/SC4. (Zitiert auf Seite 89)

[ISO94b] *ISO 11898-1:2003 Road vehicles – Controller area network (CAN)*. International

Organization for Standardization (ISO), 1994. (Zitiert auf Seite 21)

[ISO99a] *ISO 14230: Road vehicles – Diagnostic systems – Keyword Protocol 2000*. International Organization for Standardization (ISO), 1999. (Zitiert auf Seite 38)

[ISO99b] *ISO/IEC 9899:1999 Programming languages – C*. International Organization for Standardization (ISO), Dezember 1999. (Zitiert auf Seite 128)

[ISO00] *ISO 3888-2: Passenger cars - Test track for a severe lane-change manoeuvre - Part 2: Obstacle avoidance*. International Organization for Standardization (ISO), 2000. (Zitiert auf Seite 3)

[ISO04] *ISO 15765: Road vehicles – Diagnostics on Controller Area Networks (CAN)*. International Organization for Standardization (ISO), 2004. (Zitiert auf Seite 38)

[ISO05a] *IS0 9000:2005. Qualitätsmanagementsysteme - Grundlagen und Begriffe*. International Organization for Standardization (ISO), Dezember 2005. (Zitiert auf Seiten 5, 57, 58, 62 und 99)

[ISO05b] *ISO 15031-6 Road vehicles – Communication between vehicle and external equipment for emissions-related diagnostics - Part 6: Diagnostic trouble code definitions*. International Organization for Standardization (ISO), Dezember 2005. (Zitiert auf Seiten 33, 37, 121 und 122)

[ISO06] *ISO 14229-1:2006 Road vehicles – Unified diagnostic services (UDS)*. International Organization for Standardization (ISO), 2006. (Zitiert auf Seiten 38, 39, 89 und 146)

[ISO08a] *ISO 22901-1:2008 Road vehicles – Open diagnostic data exchange (ODX) – Part 1: Data model specification*. International Organization for Standardization (ISO), 2008. (Zitiert auf Seite 89)

[ISO08b] *ISO 9001:2008: Quality management systems – Requirements*. International Organization for Standardization (ISO), Dezember 2008. (Zitiert auf Seite 88)

[ISO09a] *ISO 26262: Road Vehicles - Functional Safety. Draft International Standard (DIS)*. International Organization for Standardization (ISO), Juli 2009. (Zitiert auf Seiten 14, 17, 31, 35, 41, 58, 69, 104, 105, 134, 153, 219, 229 und 231)

[ISO09b] *ISO/DIS 13209-1 Road vehicles – Open Test sequence eXchange format (OTX) – Part 1: General information and use cases*. International Organization for Standardization (ISO), 2009. (Zitiert auf Seite 90)

[JD88] JAIN, A. K. und DUBES, R. C.: *Algorithms for clustering data*. Prentice-Hall, Inc., 1988. (Zitiert auf Seite 137)

[JKK93] JOHNSON, N. L., KOTZ, S. und KEMP, A. W.: *Univariate discrete distributions*. Wiley Series in Probability and Mathematical Sciences. Wiley-Interscience Publication, zweite Auflage, 1993. (Zitiert auf Seite 136)

[Jon94] JONES, C.: *Economics of Software Reuse*. IEEE Computer, 27(7):106–107, Juli 1994. (Zitiert auf Seiten 86 und 183)

[JRT08] JÜRJENS, J., REISS, D. und TRACHTENHERZ, D.: *Model-Based Quality Assurance of Automotive Software*. In: CZARNECKI, K., OBER, I., BRUEL, J. - M., UHL, A. und VÖLTER, M. (Herausgeber): *Model Driven Engineering Languages and Systems*, Band 5301 der Reihe *Lecture Notes in*

Computer Science, Seiten 858 – 873. Springer Verlag Berlin/ Heidelberg, 2008. (Zitiert auf Seite 152)

[KB10] KOHL, J. und BAUER, A.: *Role-based Diagnosis for Distributed Vehicle Functions*. In: *Proceedings of the 2010 International Workshop on Principles of Diagnosis (DX '10)*, Portland, OR, Oktober 2010. (Zitiert auf Seiten 9, 55, 79, 93, 94, 106, 108, 114, 115, 153, 154 und 156)

[KFI99] KMENTA, S., FITCH, P. und ISHII, K.: *Advanced FMEA of Complex Processes*. In: *Proceedings of the American Society Of Mechanical Engineers (ASME) Design Engineering Technical Conferences*. ASME, September 1999. (Zitiert auf Seiten 99, 102 und 104)

[KGM+04] KRÜGER, I. H., GUPTA, D., MATHEW, R., MOORTHY, P., PHILLIPS, W., RITTMANN, S. und AHLUWALIA, J.: *Towards a Process and Tool-Chain for Service-Oriented Automotive Software Engineering*. In: *Proceedings of the ICSE 2004 Workshop on Software Engineering for Automotive Systems (SEAS)*, 2004. (Zitiert auf Seiten 190 und 191)

[KI00] KMENTA, S. und ISHII, K.: *Scenario-based FMEA: a life cycle cost perspective*. In: *Proceedings of the American Society of Mechanical Engineers (ASME) Design Engineering Technical Conferences*, September 2000. (Zitiert auf Seiten 103, 119 und 120)

[KK03] KLEER, J. DE und KURIEN, J.: *Fundamentals of Model-Based Diagnosis*. Proceedings of the fifth IFAC symposium on Fault Detection, Supervision and Safety of technical Processes (Safeprocess), Seiten 25 – 36, 2003. (Zitiert auf Seiten 59 und 81)

[KKP+11] KOHL, J., KOTUCZ, A., PRENNINGER, J., DORNEICH, A. und MEINZER, S.: *Using multivariate split analysis for an improved maintenance of automotive diagnosis functions*. In: *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR '11)*, 2011. (Zitiert auf Seiten 9, 66, 87, 88, 134, 136, 137, 138, 139, 142, 150, 151 und 178)

[Kle86] KLEER, J. DE: *An assumption-based truth maintenance system*. Artificial Intelligence, 28:127–162, 1986. (Zitiert auf Seiten 51 und 59)

[Knu97] KNUTH, D. E.: *The Art Of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, dritte Auflage, 1997. (Zitiert auf Seiten 50, 74, 80, 82 und 182)

[Koh06] KOHL, J.: *Nutzung der Fahrzeugbetriebs- und Servicedaten als Sensor im Problemmanagementprozeß*. Diplomarbeit, Technische Universität München, 2006. (Zitiert auf Seiten 19, 36, 38, 44, 66, 87, 88, 134 und 140)

[Koo92] KOORDINIERUNGS- UND BERATUNGSSTELLE DER BUNDESREGIERUNG FÜR INFORMATIONSTECHNIK IN DER BUNDESVERWALTUNG (KBST) (Herausgeber): *Planung und Durchführung von IT-Vorhaben in der Bundesverwaltung - Vorgehensmodell (V-Modell)*, Band 27/1 der Reihe *Schriftenreihe der Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung (KBSt)*. Bundesministerium des Inneren, August 1992. (Zitiert auf Seiten 39, 64, 145 und 149)

[Krü00] KRÜGER, I. H.: *Distributed Systems Design with Message Se-*

quence Charts. Doktorarbeit, Technische Universität München, 2000. (Zitiert auf Seiten 76, 97, 116, 117, 118, 146 und 149)

[Krc10] KRCMAR, H.: *Informationsmanagement*. Springer Verlag, fünfte Auflage, 2010. (Zitiert auf Seite 191)

[Kud04] KUDER, M.: *Kundengruppen und Produktlebenszyklus. Dynamische Zielgruppenbildung am Beispiel der Automobilindustrie*. Doktorarbeit, Technische Universität Chemnitz, 2004. (Zitiert auf Seite 19)

[Kui94] KUIPERS, B.: *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. MIT Press, erste Auflage, 1994. (Zitiert auf Seiten 81 und 114)

[KW87] KLEER, J. DE und WILLIAMS, B. C.: *Diagnosing Multiple Faults*. Artificial Intelligence, 32(1):97 – 130, April 1987. (Zitiert auf Seiten 48, 49, 50, 51, 59 und 83)

[LA90] LEE, P. A. und ANDERSON, T.: *Fault Tolerance. Principles and Practice*. In: AVIŽIENIES, A., KOPETZ, H. und LAPRIE, J. C. (Herausgeber): *Dependable Computing and Fault-Tolerant Systems*, Band 3. Springer Verlag, zweite revidierte Auflage, 1990. (Zitiert auf Seiten 32 und 57)

[Lam78] LAMPORT, L.: *Time, Clocks, and the Ordering of Events in a Distributed System*. Communications of the ACM, 21(7):558 – 565, 1978. (Zitiert auf Seite 13)

[Lap85] LAPRIE, J. C.: *Dependable Computing and Fault Tolerance: Concepts and Terminology*. In: *Proceedings of the 15th IEEE International Symposium on Fault Tolerant Computing (FTCS-15)*, Seiten 2 – 11, Juni 1985. (Zitiert auf Seite 57)

[Lap92] LAPRIE, J.-C. (Herausgeber): *Dependability: Basic Concepts and Terminology in English, French, German, Italian and Japanese*, Band 5 der Reihe *Dependable Computing and Fault-Tolerant Systems*. Springer Verlag, 1992. (Zitiert auf Seiten 27, 28 und 57)

[Lev95] LEVESON, N.: *Safeware - System Safety and Computers. A Guide to Preventing Accidents and Losses caused by Technology*. Addison-Wesley, 1995. (Zitiert auf Seite 57)

[LG84] LANERGAN, R. G. und GRASSO, C.: *Software engineering with reusable designs and code*. IEEE Transactions on Software Engineering, 10(5):498–501, 1984. (Zitiert auf Seiten 86 und 183)

[LHM99] LIU, B., HSU, W. und MA, Y.: *Mining Association Rules with multiple mininum supports*. In: *Proceedings of the fifth ACM SIGKDD*, Seiten 337–341. Association for Computing Machinery (ACM), 1999. (Zitiert auf Seite 137)

[Liu00] LIU, J. W. S.: *Real-Time Systems*. Prentice Hall, 2000. (Zitiert auf Seite 14)

[Loè77] LOÈVE, M.: *Probability Theory I*. Springer Verlag, vierte Auflage, 1977. (Zitiert auf Seite 135)

[Lun10a] LUNZE, J.: *Regelungstechnik 1: Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen*. Springer Verlag, achte Auflage, 2010. (Zitiert auf Seite 21)

[Lun10b] LUNZE, J.: *Regelungstechnik 2: Mehrgrößensysteme, digitale Regelung*. Springer Verlag, sechste Auflage, 2010. (Zitiert auf Seite 22)

[LY03] LI, Q. und YAO, C.: *Real-Time Concepts for Embedded Systems*. CMP Books, 2003. (Zitiert auf Seite 132)

[McK91] MCKINNEY, B.: *FMECA, the right way*. In: *Proceedings of the 1991 IEEE Annual Reliability and Maintainability Symposium*, Seiten 253 – 259, 1991. (Zitiert auf Seite 102)

[Men93] MENGE, H. (Herausgeber): *Langenscheidt Taschenwörterbuch Altgriechisch*. Langenscheidt, zweite Auflage, 1993. (Zitiert auf Seite 26)

[MI03] MCKINSEY & COMPANY und INSTITUT FÜR PRODUKTIONSMANAGEMENT, TECHNOLOGIE UND WERKZEUGMASCHINEN (PTW) TU DARMSTADT (Herausgeber): *HAWK 2015 - Herausforderung automobile Wertschöpfungskette. Wissensbasierte Veränderung der automobilen Wertschöpfungskette*, Band 30 der Reihe *Materialen zur Automobilindustrie*. Verband der Automobilindustrie (VDA), 2003. (Zitiert auf Seiten 3, 5, 6, 7, 16, 17 und 68)

[MIO87] MUSA, J., IANNINO, A. und OKUMOTO, K.: *Software Reliability: Measurement, Predication, Application*. Series in Software Engineering and Technology. McGraw-Hill, 1987. (Zitiert auf Seiten 57 und 58)

[MKL+09] MÜLLER, T., KRIEGER, O., LANGE, K., BREUER, A. und FORM, T.: *Neuronale Netzmengen für die Fehlerdiagnose in komplexen Fahrzeugsystemen*. In: BÄKER, B.A. AND UNGER, A. (Herausgeber): *Diagnose in mechatronischen Fahrzeugsystemen*, Band 1, Seiten 168 – 179. Expert Verlag, Juli 2009. (Zitiert auf Seiten 59 und 142)

[MLM+06] MACKENZIE, C. M., LASKEY, K., MCCABE, F., BROWN, P. F. und METZ, R. (HERAUSGEBER): *Reference Model for Service Oriented Architecture. Committee Specification 1*. Technischer Bericht, Organization for the Advancement of Structured Information Standards (OASIS), August 2006. Verfügbar unter: Link, abgerufen am 26.03.2011. (Zitiert auf Seite 191)

[MM75] MAGNUSON, W. G. und MOSS, J. E.: *Magnuson-Moss Warranty-Federal Trade Commission Improvement Act*. In: *United States Code (U.S.C) Title 15 - Commerce and Trade §§ 2301 -2312 (Supp. V 1975), 93-637, 88 Stat. 2183*. United States Government Printing Office, 1975. Verfügbar unter Link, abgerufen am 07.08.2010. (Zitiert auf Seite 5)

[MMZ+01] MOSKEWICZ, M. W., MADIGAN, C. F., ZHAO, Y., ZHANG, L. und MALIK, S.: *Chaff: engineering an efficient SAT solver*. In: *Proceedings of the 38th annual Design Automation Conference (DAC '01)*, Seiten 530–535, New York, NY, 2001. Association for Computing Machinery (ACM). (Zitiert auf Seiten 8, 55 und 130)

[Moo65] MOORE, G. E.: *Cramming more components onto integrated circuits*. Electronics, 38(8):114–117, April 1965. (Zitiert auf Seiten 2 und 12)

[Mot04] MOTOR INDUSTRY SOFTWARE RELIABILITY ASSOCIATION (MISRA): *Guidelines for the Use of the C Language in Critical Systems*, 2004. (Zitiert auf Seiten 83, 132, 147 und 223)

[MPEE10] MEINZER, S., PRENNINGER, J., EBERL, M. und EREN, T.: *Durch Predictive Analytics von Diagnosedaten zu fundierten Qualitätsmanagemententscheidungen und höherer Kundenzufriedenheit*. In: HEYER, G., LUY, J.F.

und JAHN, A. (Herausgeber): *Text- und Data Mining für die Qualitätsanalyse in der Automobilindustrie, Leipziger Beiträge zur Informatik Vol XXV*, Seiten 17 – 27, 2010. (Zitiert auf Seiten 36, 134, 137 und 140)

[MSC04] *Recommendation Z.120 – Message Sequence Chart (MSC)*. ITU-T Telecommunication Standardization Sector of International Telecommunication Union (ITU), April 2004. Verfügbar unter Link, abgerufen am 19.09.2010. (Zitiert auf Seiten 72, 76, 97, 116, 146, 149 und 220)

[MWS05] MIKAELIAN, T., WILLIAMS, B. C. und SACHENBACHER, M.: *Diagnosing Complex Systems with Software-Extended Behavior using Constraint Optimization*. In: *Proceedings of the 16th International Workshop on Principles of Diagnosis (DX '05)*, Monterey, CA, 2005. (Zitiert auf Seite 81)

[NAS66] *Procedure for Failure Mode, Effects and Criticality Analysis (FMECA)*. National Aeronautics and Space Administration (NASA), August 1966. (Zitiert auf Seite 98)

[Nat99] NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA): *Mars Climate Orbiter Mishap Investigation Board Phase I Report*. Technischer Bericht, National Aeronautics and Space Administration (NASA), November 1999. Verfügbar unter Link, abgerufen am 13.11.2010. (Zitiert auf Seite 31)

[Nat09] NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA): *Jet Propulsion Laboratory (JPL) Institutional Coding Standard for the C Programming Language*. Technischer Bericht, National Aeronautics and Space Administration (NASA), März 2009. Verfügbar unter Link, abgerufen am 07.08.2011. (Zitiert auf Seite 132)

[Neg06] NEGELE, H.: *Systems Engineering Challenges and Solutions from an Automotive Perspective. Keynote Presentation*. In: *International Council on Systems Engineering (INCOSE) Symposium 2006*, Orlando, FL, Juni 2006. (Zitiert auf Seite 2)

[NHB05] NOLTE, T., HANSSON, H. und BELLO, L. L.: *Automotive Communications - Past, Current and Future*. In: *10th IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2005)*, Band 1, 2005. (Zitiert auf Seiten 13, 21 und 154)

[Nik74] NIKULIN, M.S.: *χ^2-Test for Continuous Distributions with Shift and Scale Parameters*. Theory of Probability and its Applications, 18(3):559–568, 1974. (Zitiert auf Seite 137)

[NS63] NEWELL, A. und SIMON, H. A.: *GPS, a program that simulates human thought*. In: FEIGENBAUM, E. A. und FELDMAN, J. (Herausgeber): *Computers and thought*, Seiten 279–293. MIT Press, 1963. Erschienen 1995 im Sammelband Computers and thought. (Zitiert auf Seite 41)

[NS04] NYBERG, M. und STUTTE, T.: *Model based diagnosis of the air path of an automotive Diesel engine*. Control Engineering Practice, 12(5):513–525, 2004. (Zitiert auf Seite 54)

[OHL91] ORMSBY, A., HUNT, J. und LEE, M.: *Towards an automated FMEA assistant*. Artificial Intelligence in Engineering, Seiten 739–752, 1991.

(Zitiert auf Seite 103)

[Pat01] PATÉ-CORNELL, E. AND DILLON, R.: *Probabilistic risk analysis for the NASA space shuttle: a brief history and current work*. Reliability Engineering & System Safety, 74(3):345 – 352, 2001. (Zitiert auf Seite 98)

[Pau81] PAU, L. F.: *Failure Diagnosis and Perfomance Monitoring*. In: MENDEL, J. (Herausgeber): *Control and Systems Theory*, Band 11. Marcel Dekker, Inc., 1981. (Zitiert auf Seite 89)

[PBC+02] PICARDI, C., BRAY, R., CASCIO, F., CONSOLE, L., DAGUE, P., DRESSLER, O., MILLET, D., REHFUS, B., STRUSS, P. und VALLÉE, C.: *IDD: Integrating Diagnosis in the Design of automotive systems*. In: *Proceedings of the Fifteenth European Conference on Artificial Intelligence (ECAI '02)*, Seiten 628 – 632, Lyon, Frankreich, Juli 2002. (Zitiert auf Seiten 90 und 112)

[PBKS07] PRETSCHNER, A., BROY, M., KRÜGER, I. H. und STAUNER, T.: *Software Engineering for Automotive Systems: A Roadmap*. In: *Future of Software Engineering (FOSE '07)*, Seiten 55 – 71. IEEE Computer Society, Mai 2007. (Zitiert auf Seiten 4, 78, 83 und 93)

[Pea86] PEARL, J.: *Fusion, propagation and structuring in belief networks*. Artificial intelligence, 29(3):241–288, 1986. (Zitiert auf Seiten 44 und 80)

[Pha03] PHAM, H. (Herausgeber): *Handbook of Reliability Engineering*. Springer Verlag, 2003. (Zitiert auf Seiten 27 und 57)

[Pnu77] PNUELI, A.: *The temporal logic of programs*. In: *Annual IEEE Symposium on Foundations of Computer Science*, Band 0, Seiten 46–57, Los Alamitos, CA, USA, Juli 1977. IEEE Computer Society. (Zitiert auf Seiten 55 und 115)

[Pnu86] PNUELI, A.: *Applications of Temporal Logic to the Specification and Verification of Reactive Systeme: A Survey of Current Trends*. In: *Current trends in concurrency. Overviews and tutorials*. Springer Verlag, 1986. (Zitiert auf Seiten 13, 57 und 115)

[PPWS95] PRICE, C.J., PUGH, D.R., WILSON, M. S. und SNOOKE, N.: *The flame system: automating electrical failure mode and effects analysis (FMEA)*. In: *Proceedings of the Annual Reliability and Maintainability Symposium*, Seiten 90–95. Institute of Electrical and Electronics Engineers (IEEE), Januar 1995. (Zitiert auf Seite 81)

[PSST00] PRESTL, W., SAUER, T., STEINLE, J. und TSCHERNOSTER, O.: *The BMW Active Cruise Control (ACC)*. Society of Automotive Engineers (SAE) Transactions, 109(7):119–125, 2000. (Zitiert auf Seiten 3, 13, 24 und 25)

[PT02] PRICE, C.J. und TAYLOR, N.S.: *Automated multiple failure FMEA*. Reliability Engineering & System Safety, 76(1):1–10, 2002. (Zitiert auf Seite 81)

[Ram06] RAMSDEN, E.: *Hall-Effect Sensors—Theory and Application*. Elsevier, zweite Auflage, 2006. (Zitiert auf Seiten 154 und 157)

[Rau11] RAUBOLD, U.: *Lebenszyklusmanagement in der Automobilindustrie*. Doktorarbeit, Technische Universität Cottbus, 2011. (Zitiert auf Seite 18)

[Rei79] REIFER, D. J.: *Software failure modes and effects analysis*. IEEE Transactions

on Reliability, 28:247–249, 1979. (Zitiert auf Seiten 88 und 99)

[Rei87] REITER, R.: *A Theory of Diagnosis from First Principles*. Artificial Intelligence, 32(1):57 – 95, 1987. (Zitiert auf Seiten 48, 49, 50, 51, 59, 81 und 83)

[Rei06] REIF, K. (Herausgeber): *Automobilelektronik. Eine Einführung für Ingenieure*. Vieweg Verlag, erste Auflage, April 2006. (Zitiert auf Seiten 20, 123, 154 und 157)

[Rei11] REIF, K. (Herausgeber): *Bosch Autoelektrik und Autoelektronik. Bordnetze, Sensoren und elektronische Systeme*. Vieweg + Teubner Verlag, sechste überarbeitete und erweiterte Auflage Auflage, 2011. (Zitiert auf Seiten 20 und 21)

[Ren07] RENNER, I.: *Methodische Unterstützung funktionsorientierter Baukastenentwicklung am Beispiel Automobil*. Doktorarbeit, Technische Universität München, 2007. (Zitiert auf Seite 85)

[Rit08] RITTMANN, S.: *A methodology for modeling usage behavior of multifunctional systems*. Doktorarbeit, Technische Universität München, 2008. (Zitiert auf Seite 85)

[RN10] RUSSEL, S. und NORVIG, P.: *Artificial Intelligence. A modern approach*. Pearson Education Inc., dritte Auflage, 2010. (Zitiert auf Seiten 42, 43, 44, 45, 79, 80 und 124)

[Rob07a] ROBERT BOSCH GMBH (Herausgeber): *Autoelektrik / Autoelektronik*. Vieweg + Teubner, fünfte Auflage, 2007. (Zitiert auf Seiten 20 und 21)

[Rob07b] ROBERT BOSCH GMBH (Herausgeber): *Kraftfahrtechnisches Taschenbuch*. Vieweg Verlag, 26. Auflage, 2007. (Zitiert auf Seiten 2, 3 und 20)

[Ros06] ROSS, S.: *A first course in probability*. Pearson International Edition, siebte Auflage, 2006. (Zitiert auf Seite 135)

[Rum05] RUMBAUGH, J. AND JACOBSON, I. AND BOOCH, G.: *The Unified Modeling Language Reference Manual*. Addison-Wesley Professional, zweite Auflage, 2005. (Zitiert auf Seiten 45, 146, 147, 149 und 169)

[Saf92] SAFRA, J. E. (Herausgeber): *The new Encyclopaedia Britannica - Macropaedia*. Encyclopaedia Britannica Inc., 15. Auflage, 1992. (Zitiert auf Seite 26)

[Sak08] SAKS, D.: *Know when to use dynamic allocation*. Electronic Engineering Times Asia, August 2008. Verfügbar unter Link, abgerufen am 07.08.2011. (Zitiert auf Seite 132)

[SBPM08] STEINBERG, D., BUDINSKY, F., PATERNOSTRO, M. und MERKS, E.: *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, zweite Auflage, 2008. (Zitiert auf Seite 128)

[Sch00] SCHÖNING, U.: *Logik für Informatiker*. Spektrum Akademischer Verlag, fünfte Auflage, 2000. (Zitiert auf Seiten 107 und 108)

[Sch02] SCHULZ, A. P.: *Systemtechnische Gestaltung der Informationsarchitektur im Entwicklungsprozeß*. Doktorarbeit, Technische Universität München, 2002. (Zitiert auf Seite 19)

[Sch08] SCHWITZER, W.: *Spezifikation und Implementierung eines modellbasierten Deployment-Konzepts für AutoFocus3*. Diplomarbeit, Technische Universität

München, 2008. (Zitiert auf Seiten 127 und 128)

[Sch11] SCHEICKL, O.: *Timing Constraints in distributed development of automotive real time systems*. Doktorarbeit, Technische Universität München, 2011. (Zitiert auf Seiten 2 und 26)

[Sim54] SIMON, H. A.: *Spurious correlation: A causal interpretation*. Journal of the American Statistical Association (JSTOR), 49(267):467–479, 1954. (Zitiert auf Seite 137)

[Sim62] SIMON, H. A.: *The architecture of complexity*. In: *Proceedings of the American Philosophical Society*, Band 106, Seiten 467–482, Dezember 1962. (Zitiert auf Seite 94)

[SK05] SCHORN, M. und KAPUST, A.: *Design Review Based on Failure Mode (DRBFM): die Toyota-Methode*. VDI Z-Integrierte Produktion/ Springer Verlag, 147(7-8):67–69, 2005. (Zitiert auf Seiten 99 und 103)

[SKMP10] SANKAVARAM, C., KODALI, A., MARTINEZ AYALA, D. F. und PATTIPATI, K.: *Event-driven Data Mining Techniques for Automotive Fault Diagnosis*. In: *Proceedings of the 2010 International Workshop on Principles of Diagnosis (DX '10)*, Portland, OR, Oktober 2010. (Zitiert auf Seite 142)

[SN96a] SCHULTE, R. W. und NATIS, Y. V.: *Service Oriented Architectures, Part 1*. Technischer Bericht Research Note SPA-401-068, Gartner Inc., April 1996. (Zitiert auf Seite 191)

[SN96b] SCHULTE, R. W. und NATIS, Y. V.: *Service Oriented Architectures, Part 2*. Technischer Bericht Research Note SPA-401-069, Gartner Inc., April 1996. (Zitiert auf Seite 191)

[SP07] SCHUMANN, A. und PENCOLÉ, Y.: *Scalable Diagnosability Checking of Event-Driven Systems*. In: *International Joint Conference on Artificial Intelligence (IJCAI '07)*, Seiten 575–580, 2007. (Zitiert auf Seite 189)

[SR03] SPENCER, C. M. und RHEE, S. J.: *Cost based failure modes and effects analysis (FMEA) for systems of accelerator magnets*. In: *Proceedings of the Particle Accelerator Conference (PAC 2003)*, Band 4, Seiten 2177–2179. Institute of Electrical and Electronics Engineers (IEEE), 2003. (Zitiert auf Seite 103)

[SRB+02] STRUSS, P., REHFUS, B., BRIGNOLO, R., CASCIO, F., CONSOLE, L., DAGUE, P., DUBOIS, P., DRESSLER, O. und MILLET, D.: *Model-based Tools for the integration of Design and Diagnosis into a common process - a project report*. In: *Working Papers of the 13th International Workshop on the Principles of Diagnosis (DX '02)*, Semmering, Austria, 2002. (Zitiert auf Seite 112)

[SS96] SILVA, J. P. M. und SAKALLAH, K. A.: *GRASP – a new search algorithm for satisfiability*. In: *Proceedings of the 1996 IEEE/ ACM international conference on Computer-aided design (ICCAD '96)*, Seiten 220–227. IEEE Computer Society Press, 1996. (Zitiert auf Seite 130)

[SSL+94] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., SINNAMOHIDEEN, K. und TENEKETZIS, D.: *Diagnosability of discrete event systems*. In: COHEN, G. und QUADRAT, J.-P. (Herausgeber): *Eleventh International Conference on Analysis and Optimization of Systems Discrete Event Systems*,

Band 199 der Reihe *Lecture Notes in Control and Information Sciences*, Seiten 73–79. Springer Verlag, 1994. (Zitiert auf Seiten 46, 47, 82 und 95)

[SSL+96] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., SINNAMOHIDEEN, K. und TENEKETZIS, D.: *Failure diagnosis using discrete-event models*. IEEE Transactions on Control Systems Technology, 4(2):105–124, 1996. (Zitiert auf Seiten 47 und 95)

[Sta88] STANKOVIC, J. A.: *Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems*. Computer, 21:10–19, Oktober 1988. (Zitiert auf Seite 14)

[Str06] STRUSS, P.: *A model-based methodology for the integration of diagnosis and fault analysis during the entire life cycle*. In: *Proceedings of SAFEPROCESS 2006*. Elsevier, 2006. (Zitiert auf Seite 152)

[Stu04] STUMVOLL, H.: *Return on Quality (ROQ): Wirtschaftlichkeit von Produktqualität aus Unternehmenssicht. Entwicklung einer kundenloyalitätsbasierten Bewertungsmethode am Beispiel eines Automobilherstellers*. Doktorarbeit, Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen, 2004. (Zitiert auf Seite 5)

[SWLL06] SON, J., WILSON, I., LEE, W. und LEE, S.: *Model Based Embedded System Development for In-Vehicle Network Systems*. Technischer Bericht 2006-01-0862, Society of Automotive Engineers (SAE) International, 2006. (Zitiert auf Seite 152)

[SZ10] SCHÄUFFELE, J. und ZURAWKA, T.: *Automotive Software Engineering. Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen*. Vieweg + Teubner Verlag, vierte Auflage, 2010. (Zitiert auf Seiten 2, 16, 18, 20, 22, 24, 30, 38, 53, 63 und 95)

[Ten00] TENNENHOUSE, D.: *Proactive computing*. Communications of the ACM, 43(5):43–50, 2000. (Zitiert auf Seite 12)

[TEO06] TRAVÉ-MASSUYES, L., ESCOBET, T. und OLIVE, X.: *Diagnosability analysis based on component-supported analytical redundancy relations*. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 36(6):1146–1160, 2006. (Zitiert auf Seite 189)

[TF06] TABACHNICK, B. und FIDELL, L.: *Using Multivariate Statistics*. Allyn and Bacon, fünfte Auflage, 2006. (Zitiert auf Seite 136)

[TH02] THIEL, S. und HEIN, A.: *Modeling and Using Product Line Variability in Automotive Systems*. IEEE Software, 19:66–72, 2002. (Zitiert auf Seiten 85 und 89)

[TK08] THEODORIDIS, S. und KOUTROUMBAS, K.: *Pattern Recognition*. Academic Press, vierte Auflage, 2008. (Zitiert auf Seite 137)

[Toy09] *Toyota Motor Corporation Annual Report*, 2009. Verfügbar unter Link, abgerufen am 20.12.2010. (Zitiert auf Seite 6)

[Tri09] TRIPAKIS, S.: *A Combined on-line/ off-line Framework for Black-Box Fault Diagnosis*. In: *Ninth International Workshop on Runtime Verification (RV 2009)*, Seiten 152 – 169, Grenoble, Frankreich, Juni 2009. Springer Verlag.

(Zitiert auf Seiten 54, 55, 83, 114, 115 und 129)

[TS06] TANENBAUM, A. und STEEN, M. VAN: *Distributed Systems. Principles and Paradigms*. Prentice Hall International, zweite revidierte Auflage, 2006. (Zitiert auf Seite 13)

[VDA96] *Sicherung der Qualität vor Serieneinsatz. Band 4 Teil 2 - System-FMEA)*. Verband der Automobilindustrie (VDA), 1996. (Zitiert auf Seiten 71, 99, 100, 102 und 104)

[VGSS09] VOGET, S., GOLM, M., SANCHEZ, B. und STAPPERT. F.: *Application of the AUTOSAR standard*. In: NAVET, N. und SIMONOT-LION, F. (Herausgeber): *Automotive Embedded Systems Handbook*, 2009. (Zitiert auf Seite 141)

[VLG10] VO, G.N., LAI, R. und GARG, M.: *Building Automotive Software Component within the AUTOSAR Environment - A Case Study*. In: *Ninth International Conference on Quality Software (QSIC '09)*, Seiten 191–200, Jeju Island, Korea, August 2010. Institute of Electrical and Electronics Engineers (IEEE). (Zitiert auf Seite 152)

[VS08] VOLPATO, G. und STOCHETTI, A.: *Managing product life cycle in the auto industry: evaluating carmakers effectiveness*. International Journal of Automotive Technology and Management, 8(1):22 – 41, 2008. (Zitiert auf Seite 19)

[VW94] VARDI, M. Y. und WOLPER, P.: *Reasoning about infinite computations*. Information and Computation, 115:1–37, November 1994. (Zitiert auf Seiten 74 und 83)

[WDS09] WINNER, H., DANNER, B. und STEINLE, J.: *Adaptive Cruise Control*. In: WINNER, H., HAKULI, S. und WOLF, G. (Herausgeber): *Handbuch Fahrerassistenzsysteme - Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Vieweg + Teubner, erste Auflage, 2009. (Zitiert auf Seiten 3, 24 und 25)

[WFH+06] WILD, D., FLEISCHMANN, A., HARTMANN, J., PFALLER, C., RAPPL, M. und RITTMANN, S.: *An Architecture-Centric Approach towards the Construction of Dependable Automotive Software*. In: *Proceedings of the Society of Automotive Engineers (SAE) 2006 World Congress*, 2006. (Zitiert auf Seiten 85, 93 und 141)

[Wis99] WISSENSCHAFTLICHER RAT DER DUDENREDAKTION (Herausgeber): *Duden - Das große Wörterbuch der deutschen Sprache*. Duden Verlag, dritte Auflage, 1999. (Zitiert auf Seite 7)

[WR11] WALLENTOWITZ, H. und REIF, K. (Herausgeber): *Handbuch Kraftfahrzeugelektronik. Grundlagen - Komponenten - Systeme - Anwendungen*. Vieweg + Teubner Verlag, zweite Auflage, 2011. (Zitiert auf Seiten 20, 21 und 154)

[WWP04] WOLF, M., WEIMERSKIRCH, A. und PAAR, C.: *Security in Automotive Bus Systems*. In: *Proceedings of the Workshop on Embedded Security in Cars (escar '04)*, 2004. (Zitiert auf Seiten 13 und 21)

[Zei00] ZEIGLER, B. P. AND PRAEHOFER, H. AND KIM, T. G.: *Theory of Modeling and Simulation. Integrating Discrete Event and Continuous Com-*

plex Dynamic Systems. Elsevier Academic Press, zweite Auflage, 2000. (Zitiert auf Seite 46)

[Zem68] ZEMANEK, H.: *Abstrakte Objekte*. Elektronische Rechenanlagen, 10(5):208 – 217, 1968. (Zitiert auf Seiten 77 und 182)

[Zha03] ZHANG, L.: *Searching for truth: Techniques for satisfiability of boolean formulas*. Doktorarbeit, Princeton University, Juni 2003. Verfügbar unter Link, abgerufen am 20.12.2010. (Zitiert auf Seiten 8, 74, 83, 130 und 160)

[ZS08] ZIMMERMANN, W. und SCHMIDGALL, R.: *Bussysteme in der Fahrzeugtechnik: Protokolle und Standards*. Vieweg + Teubner, dritte Auflage, 2008. (Zitiert auf Seiten 13, 21, 24, 90 und 154)

Anhang A

Entwicklungsprozesse im Detail

A.1 Bisheriger Entwicklungsprozeß

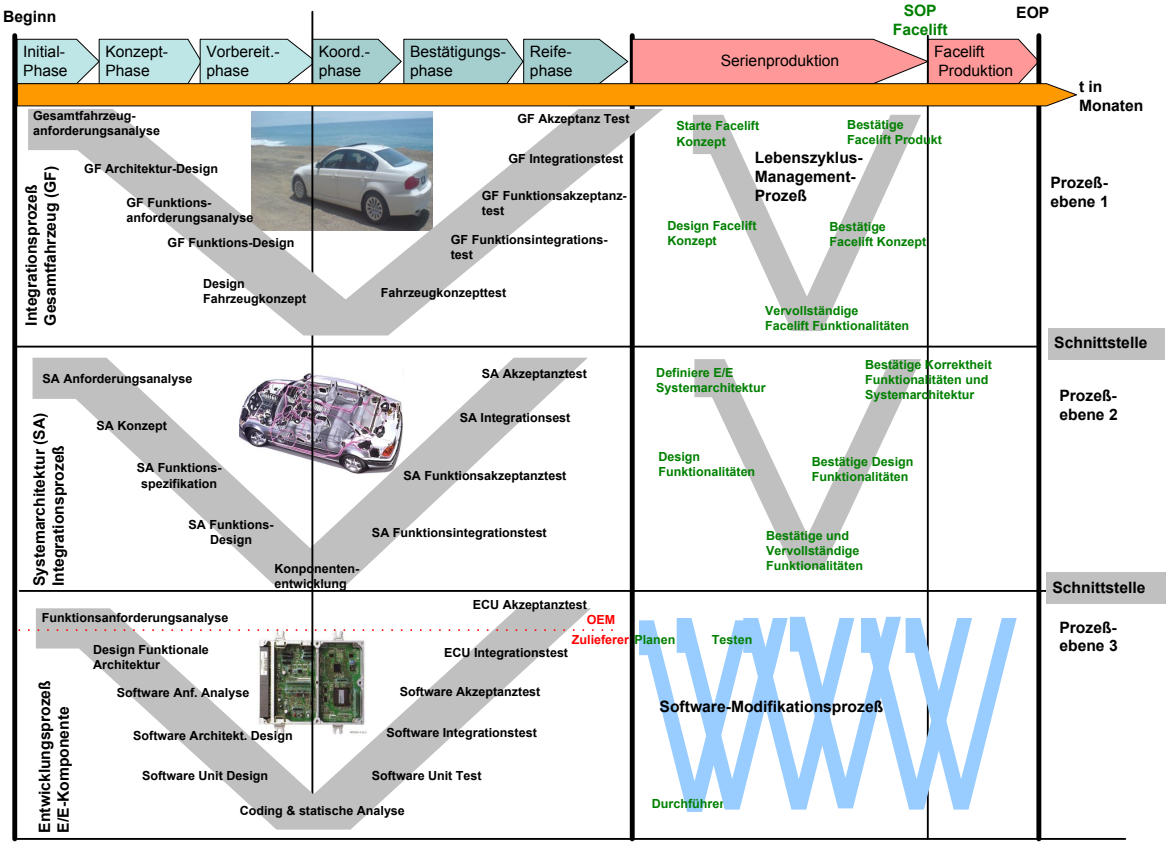


Abbildung A.1: Übersicht über den Standardentwicklungsprozeß für Automobile über den gesamten Lebenszyklus. Quelle: basierend auf [BMW06, BMW07]

A.2 Prozeßschritte des Entwicklungs-Workflows

In diesem Abschnitt werden die einzelnen Prozeßschritte aus Abbildung 3.1 aufgeführt. Die Schritte sind entnommen aus [BMW06].

1. Wähle Fahrzeugteile aus Baukastendatenbank.
2. Schlage Gesamtfahrzeugarchitektur vor.
3. Schlage Gesamtfahrzeugarchitektur vor.
4. Schlage vor, welche Komponenten für das Fahrzeug wiederverwendet oder neu entwickelt werden sollen.
5. Berichte bekannte kritische Themen für die vorgeschlagenen Komponenten.
6. Berichte bekannte kritische Themen für die vorgeschlagene Systemarchitektur.
7. Kritische Themen auf Komponenten- und Systemarchitekturebene sind dokumentiert.
8. Schlage Fahrzeugsystemarchitektur mit Komponenten vor.
9. Definiere Fahrzeugkonzeptszenarien und -ausstattungen. Bestätigung, daß Vorschlag Systemarchitektur mit Komponenten akzeptiert ist.
10. Definiere Fahrzeugkonzeptszenarien und -ausstattungen. Bestätigung, daß Komponentenvorschlag akzeptiert ist.
11. Bericht mit Innovationsbewertungen sowie Spezifikationen der gewählten Komponenten verfügbar.
12. Absicherungsplan funktionale Sicherheit wird für neu definierte Innovationskomponenten erstellt oder bei Übernahmekomponenten in die Spezifikation integriert. Lessons learned in Spezifikationen eingebaut.
13. Bereitstellung Erstbewertung Gefahren und Risiken der gewählten Systemarchitektur sowie Qualitätsbewertung Systemarchitektur mit Lessons learned.
14. Erstidentifizierung und Dokumentierung kritischer Konzeptanteile.
15. Definition grundlegendes E/E-Konzept mit vorläufiger Risikobewertung.
16. Grundlegendes Gesamtfahrzeugkonzept mitsamt Systemarchitektur bestätigt. Dies beinhaltet das funktionale Design des Fahrzeugs.
17. Bordnetz, Betriebssystem sowie funktionales Design als Grundgerüst für die Komponentenentwicklung fertiggestellt. Vorgeschlagene Komponenten bestätigt.
18. Qualitätsanforderungen in Spezifikationen eingearbeitet.
19. Funktionen auf Komponentenebene definiert. Spezifikationen abgeschlossen.
20. Gefahren- und Risikenanalyse für bestätigte Systemarchitektur abgeschlossen.
21. Gefahren- und Risiken der Funktionen bewertet.
22. E/E-Spezifikation mit allen Kunden- und Systemfunktionen auf System- und Komponentenebene abgeschlossen.

23. Komponentenübergreifende Funktionen (auf Systemebene) bestätigt.
24. Funktionen auf Komponentenebene bestätigt.
25. Bereitstellung der technischen Spezifikationen, die relevant für komponentenübergreifende Interaktionen sind. Bereitstellung Implementierungsplanung.
26. Qualitätsanforderungen auf Komponenten- und Systemarchitekturebene sind in Komponentenspezifikation eingebaut.
27. Qualitätsanforderungen auf der Ebene komponentenübergreifende Funktionen sind in Spezifikation eingebaut. Sicherheitsanforderungen auf Systemebene definiert.
28. Konflikte und Gefahren für das Konzept sind bewertet.
29. Vorlage des Funktionsdesign und Implementierungsplan zur Genehmigung durch Gesamtfahrzeugebene.
30. Bestätigung Design und Implementierungsplan. Implementierung kann beginnen.
31. Bestätigung technische Spezifikation und Interaktionen der Komponenten. Systemarchitektur wird eingefroren, d.h. keine funktionale Änderungen mehr an der Systemarchitektur möglich.
32. Erstellung detailliertes Design für die Implementierungsphase.
33. Programmierung und statische Programmanalyse abgeschlossen. Übergang zu Testphasen.
34. Software-Einheiten gegen Design-Spezifikation validiert. Bereit für Integration Software-Einheiten in Software-Komponenten.
35. Komponenten mit implementierten Funktionen fertig.
36. Validierungsplan für Komponenten definiert.
37. Validierungsplan für Systemarchitektur definiert.
38. Dokumentation vorhandener Probleme nach der Implementierung.
39. Bericht, daß Basisfunktionen implementiert sind.
40. Gesamtfahrzeug bestätigt durch entwickelte Komponenten. Systemarchitektur bestätigt.
41. Komponentenkonzept bestätigt.
42. Alle Funktionen auf Komponentenebene implementiert.
43. Testfälle funktionale Sicherheit auf Komponentenebene definiert.
44. Testfälle funktionale Sicherheit auf Systemarchitekturebene definiert. System-FMEA vervollständigt.
45. Potentiell gefährliche Abweichungen vom Fahrzeugkonzept dokumentiert und Gesamtfahrzeugkonzept bestätigt durch Diagnoseentwickler.
46. Bericht, daß alle E/E-Funktionen entwickelt sind.
47. Alle Fahrzeugfunktionen bestätigt auf Gesamtebene.

48. Funktionen bestätigt, können in Steuergeräte integriert und getestet werden.
49. Software in Komponenten integriert und validiert.
50. Werkstatt/Service-Anforderungen der Komponenten sind definiert.
51. Anforderungen an die Werkstätten/After Sales Service sind definiert.
52. Potentiell gefährliche/risikoreiche Abweichungen von den Gesamtfahrzeugfunktionen sind dokumentiert.
53. Alle E/E-Funktionen implementiert und gegen Anforderungen validiert.
54. Gesamtfahrzeugfunktionen validiert, Fahrzeugprodukt bestätigt.
55. Funktionen auf technischer Ebene bestätigt.
56. Alle Steuergerätefunktionen validiert. Produktionsfreigabe Steuergeräte.
57. Risikobewertung der Komponenten abgeschlossen.
58. Risikobewertung der Systemarchitektur abgeschlossen.
59. Verfügbarkeit der Werkstattmaßnahmen zum Produktionsanlauf bestätigt. Potentiell gefährliche Abweichungen vom Gesamtfahrzeugprodukt dokumentiert.
60. Alle Funktionen im Fahrzeug integriert und ohne Fehler.
61. Gesamtfahrzeugkonfiguration validiert. Produktionsanlauf bestätigt.
62. Sicherheitsuntersuchung abgeschlossen. FMEA-Maßnahmen implementiert.
63. Systemarchitektur validiert und abgeschlossen.
64. Abweichungen vom Gesamtfahrzeugkonzept dokumentiert.
65. Diagnose vervollständigt und für die Werkstätten bereitgestellt.
66. Gesamtfahrzeug konform zu Anforderungen. Prozeßsicherheit Fahrzeug bestätigt.
67. Potentiell gefährliche Abweichungen von Prozeßsicherheit Gesamtfahrzeug dokumentiert.

A.3 Entwicklungsprozeß für die effiziente Diagnose verteilter Steuergerätefunktionen

In diesem Abschnitt werden die Schritte des vorgeschlagenen Diagnoseentwicklungsprozesses für verteilte Funktionen aus Abschnitt 5.1 einzeln aufgelistet. Die einzelnen Schritte des Prozesses basieren dabei auf [BMW06] und [BMW07], wurden aber geändert und erweitert für eine Einbindung der Methodik der Dissertation sowie des Gesamtziels, die Effizienz der Diagnose zu steigern. Anhand des Prozeßvorschlags wurde zudem die Diagnose für das Fallbeispiel Fensterheber entwickelt (vgl. Abschnitt 6.2).

Gesamtfahrzeuganforderungsanalyse

Input

- Allgemeines Lastenheft Diagnose (bspw. [BMW08])
- Datenbank der Baukastenkomponenten
- Projektplan

Aktivitäten

- Use Case Analyse, welche Features das Auto haben soll
- Erstelle Sicherheitsziele des Gesamtfahrzeugs
- Erstelle erste, generelle Diagnoseanforderungen:
  - Fehler müssen entdeckbar, speicherbar und zu austauschbaren bzw. reparierbaren Einheiten (AE) zuweisbar sein
  - Reparaturmaßnahmen müssen aus gespeicherten Einträgen ableitbar sein
  - Dokumentation Diagnose muß vollständig sein gemäß definiertem Datenmodell aus Abschnitt 4.5
  - Einbau verbindlicher Diagnoseziele hinsichtlich:
    - Fehler pro Fahrzeug
    - Diagnose- bzw. Inferenzdauer

Output

- Feature Liste Fahrzeug
- Use Cases der Features der Feature List
- Erste Anforderungen an Diagnose
- Liste mit Sicherheitszielen Fahrzeug

Systemarchitekturanforderungsanalyse

Input

- Feature Liste Fahrzeug

- Erste Anforderungen an Diagnose
- Atomare Kundenfunktionen von ähnlicher/n Komponente(n) oder Vorgängersystem(en), gewartet mit Lebenszyklusdaten
- Diagnosemodell und -dokumentation für atomare Kundenfunktionen ähnlicher oder Vorgängersysteme, gewartet mit Lebenszyklusdaten

Aktivitäten

- Bestimme Kundenfunktionen
- Zerlege Kundenfunktionen bis auf Ebene atomarer Kundenfunktion (kleinste für den Kunden erlebbare Funktion)
- Identifiziere Anforderungen aller atomarer Kundenfunktionen
- Stelle sicher, daß die Systemarchitektur erweiterbar ist für verschiedene Sonderausstattungen und späteres Facelift/ Modellüberarbeitung
- Definiere bzw. aktualisiere weitere Diagnoseanforderungen:
  - Berücksichtige für sicherheitsrelevante Systeme, daß die Kundenfunktionen relevanten Standards genügen müssen (vgl. ISO 26262 [ISO09a])
  - Fehler von kundensichtbaren oder -erlebbaren Funktionen oder deren Einschränkungen müssen mittels DTC gespeichert werden
  - DTC-Setzbedingungen müssen reproduzierbar und somit verifizierbar sein
  - Jeder DTC-Eintrag muß vollständige Dokumentation haben gemäß DTC-Modell (Abschnitt 4.5.3)

Output

- Liste atomarer Kundenfunktionen
- Use Cases atomarer Kundenfunktionen
- Anforderungen atomarer Kundenfunktionen
- Diagnoseanforderungen
- Diagnosemodell auf Ebene atomarer Kundenfunktionen

Systemarchitekturkonzept

Input

- Liste atomarer Kundenfunktionen
- Use Cases atomarer Kundenfunktionen
- Anforderungen atomarer Kundenfunktionen
- Diagnoseanforderungen
- Diagnosemodell auf Ebene atomarer Kundenfunktionen

Aktivitäten

- Beschreibe atomare Kundenfunktionen und ihre Interaktionen formal, bspw.

mit MSC [MSC04] (vgl. Abschnitt 4.2).
→ Ergibt das für die Systemarchitektur notwendige Feature Function Network

- Führe erweiterte FMEA (Abschnitt 4.3) auf atomare Kundenfunktionen und ihre Interaktionen aus bzw. aktualisiere das Diagnosemodell des Vorgängers/ ähnlicher Funktionen und weise erkannte mögliche Fehler den Komponenten zu
  - Berücksichtige für sicherheitsrelevante Systeme, daß Ereignis-Fehler (vgl. Abschnitt 4.5.3) von den Kommunikationspartnern gesetzt werden müssen
  - Bestimme Kundenauswirkungen (und für Werkstatt sichtbare Effekte, falls möglich) der Fehler und füge diese Informationen dem Diagnosemodell hinzu

Output

- Feature Function Network
- Anforderungen Feature Function Network
- Diagnosespezifikation auf Ebene atomarer Kundenfunktionen bzw. Update vorhandenes Diagnosedatenmodell

Systemarchitekturfunktionsspezifikation

Input

- Feature Function Network
- Anforderungen Feature Function Network
- Diagnosespezifikation auf Ebene atomarer Kundenfunktionen bzw. Update vorhandenes Diagnosedatenmodell

Aktivitäten

- Erstelle Use Cases zur Identifikation der Funktionsbausteine
- Zerlege atomare Kundenfunktionen in Funktionsbausteine
- Identifiziere Anforderungen Funktionsbausteine
- Erstelle erweiterte FMEA auf Ebene Funktionsbausteine bzw. aktualisiere vorhandenes Diagnosemodell
- Spezifiziere DTC für Funktionsbausteine so, sodaß eine austauschbare Einheit auf der Ebene Funktionsbausteine identifizierbar ist
- Berücksichtige für sicherheitsrelevante Systeme, daß Funktionsbausteine relevanten Standards genügen müssen, bspw. ISO 26262

Output

- Liste Funktionsbausteine
- Anforderungen Funktionsbausteine
- Definition DTC auf Ebene Funktionsbausteine

- Diagnosemodell auf Funktionsbausteine

Systemarchitekturfunktionsdesign

Input

- Liste Funktionsbausteine
- Anforderungen Funktionsbausteine
- DTC auf Ebene Funktionsbausteine
- erweiterte FMEA/Diagnosemodell auf Ebene Funktionsbausteine

Aktivitäten

- Beschreibe Funktionsbausteine und ihre Interaktionen (mittels MSC)
- Erstelle Komponentenmodell und weise Bausteine zu
- Führe erweiterte FMEA auf Funktionsbausteinen und ihren Interaktionen durch bzw. aktualisiere vorhandenes Diagnosemodell von ähnlichen Komponenten bzw. Vorgängersystemen
- Bestimme auf Ebene Werkstatt Auswirkungen der Fehler sowie passende(n) Befund(e)
- Beginne mit Implementierung genereller Diagnose-Standardfunktionen (Lese Fehlerspeicher, lösche Fehlerspeicher,...)
- Erweitere Diagnosemodell durch die Zuweisung der kleinsten austauschbaren Einheit auf die Ebene Funktionsbaustein. Diese Zuweisung kann aufgrund von Lebenszyklusdaten später verfeinert werden oder auf dieser Ebene bleiben
- Erstelle Anforderung, daß diese austauschbare Einheit für die Werkstatt zugänglich sein muß
- Berücksichtige für sicherheitsrelevante Systeme, daß Ereignis-DTC von kommunizierenden Steuergeräten gesetzt werden müssen

Output

- Funktionsbausteinenetzwerk
- Diagnose-Standard-Jobs
- Diagnosespezifikation auf Ebene Funktionsbausteine

Software-Anforderungsanalyse

Input

- Funktionsbausteinenetzwerk
- Diagnose-Standard-Jobs
- Diagnosespezifikation auf Ebene Funktionsbausteine

Aktivitäten

- Zerlege Funktionsbausteine in Software-Bausteine
- Identifiziere Anforderungen an Software-Bausteine
- Prüfe Verfeinerung der vorhandenen DTC auf Ebene Software-Bausteine aus Qualitätsmanagementgründen. Diese Verfeinerung kann aufgrund von Lebenszyklusdaten erfolgen
- Berücksichtige für sicherheitsrelevante Systeme, daß Software-Bausteine relevanten Standards genügen müssen, bspw. ISO 26262

Output

- Liste von Software-Bausteinen
- Anforderungen an Software-Bausteine
- Mögliche Verfeinerung DTC auf Ebene Software-Bausteine
- Diagnosespezifikation auf Ebene Software-Bausteine

Software Architekturdesign

Input

- Liste von Software-Bausteinen
- Anforderungen an Software-Bausteine
- Mögliche Verfeinerung DTC auf Ebene Software-Bausteine
- Diagnosespezifikation auf Ebene Software-Bausteine

Aktivitäten

- Beschreibe Software-Bausteine und ihre Interaktionen mit MSC
- Erstelle erweiterte FMEA auf Software-Bausteine und ihre Interaktionen oder verwende vorhandenes (mit Daten aus Lebenszyklus gewartetem) Diagnosemodell wieder
- Bestimme Auswirkungen der Fehler auf Ebene Entwicklung und weise Ursache auf Ebene Software-Baustein zu
- Erweitere Diagnosemodell durch Zuweisung kleinster austauschbarer Einheit auf Ebene Software-Baustein. Diese Zuweisung kann durch Lebenszyklusdaten später verfeinert werden oder auf dieser Ebene bleiben
- Berücksichtige für sicherheitsrelevante Systeme, daß Ereignisfehler gesetzt werden müssen. Weise diese Fehler interagierenden Software-Bausteinen zu
- Aktualisiere Diagnosespezifikation

Output

- Beschreibung Software-Bausteine und Interaktionen
- Anforderungen an Software-Bausteine
- Diagnosespezifikation auf Ebene Software-Baustein
- Mögliche Verfeinerung DTC auf Ebene Software-Baustein

Software Unit Design

Input

- Beschreibung Software-Bausteine und Interaktionen
- Anforderungen an Software-Bausteine
- Diagnosespezifikation auf Ebene Software-Baustein
- Mögliche Verfeinerung DTC auf Ebene Software-Baustein

Aktivitäten

- Zerlege Software-Bausteine in Software Units
- Beschreibe Software Unit-Design, bspw. mit State Machines
- Führe erweiterte FMEA auf Software Units durch oder verwende vorhandenes Diagnosemodell wieder
- Prüfe Verfeinerung DTC auf Ebene Software Units, basierend auf Lebenszyklusdaten, wiederum aus Qualitätsmanagementgründen
- Berücksichtige für sicherheitsrelevante Systeme, daß Software Units relevanten Standards genügen müssen, bspw. ISO 26262
- Passe Diagnosespezifikation an

Output

- Software Unit-Design
- Angepaßte Diagnosespezifikation auf Ebene Software Unit
- Mögliche Verfeinerung DTC auf Ebene Software Unit

Programmierung und statische Code-Analyse

Input

- Software Unit-Design
- Angepaßte Diagnosespezifikation auf Ebene Software Unit
- Mögliche Verfeinerung DTC auf Ebene Software Unit

Aktivitäten

- Beginne Implementierung Software
- Beginne Implementierung Diagnose
  - Erweitere Standard-Jobs so durch komponentenspezifische Jobs, sodaß kleinste austauschbare Einheit auf gewählter Ebene identifizierbar ist
  - Implementiere DTC für Systemfunktionen
  - Erweitere Diagnose-Job und DTC-Liste
- Erstelle Code Analysen (bspw. mit [Mot04])

Output

- Funktionaler Code
- Diagnose-Code

Software Unit Test

Input

- Funktionaler Code
- Diagnose-Code
- Software Unit-Design (aus SW Unit Design)
- Diagnosespezifikation auf Ebene SW Unit (aus SW Unit Design)

Aktivitäten

- Validiere erstellte Software Units gegen Design
- Bestätige DTC auf Ebene SW Unit
- Validiere Diagnosespezifikation auf Ebene Software Unit
- Passe Diagnosespezifikation an Tests und Ergebnisse an ($\rightarrow$ transitive Abhängigkeiten!)

Output

- Validierte Software auf Software Unit-Ebene
- Bestätigte DTC auf SW Unit-Ebene
- Validierte Diagnosespezifikation auf SW Unit-Ebene

Software-Integrationstest

Input

- Validierte Software auf SW Unit-Ebene
- Bestätigte DTC auf SW Unit-Ebene
- Validierte Diagnosespezifikation auf SW Unit-Ebene
- Beschreibung Interaktionen Software-Bausteine (SW Architektur Design)
- Mögliche Verfeinerung DTC auf Software-Bausteinebene (SW Arch. Design)

Aktivitäten

- Validiere Software-Bausteine und ihre Interaktionen gegen Spezifikation
- Validiere Spezifikationen auf Ebene Software-Baustein
- Passe Spezifikationen an Tests an ($\rightarrow$ transitive Abhängigkeiten!)

Output

- Validierte Software auf Ebene Software-Baustein
- Validierte Diagnose auf Ebene Software-Baustein

Software-Akzeptanztest

Input

- Validierte Interaktionen auf Software-Bausteinebene
- Validierte Diagnose auf Software-Bausteinebene
- Liste von Software-Bausteinen (SW-Anforderungsanalyse)
- Anforderungen an Software-Bausteine (SW-Anforderungsanalyse)
- Verfeinerung DTC auf Ebene Software-Baustein (SW-Anforderungsanalyse)

Aktivitäten

- Validiere Software-Bausteine und ihre Anforderungen gegen Spezifikation
- Validiere Diagnosespezifikation auf Ebene Software-Baustein und Interaktionen gegen Spezifikation
- Bestätige gewählte DTC auf Software-Bausteinebene
- Passe Spezifikationen an Testergebnisse an ($\rightarrow$ transitive Abhängigkeiten!)

Output

- Validierte Software auf Ebene Software-Baustein und Interaktionen
- Validierte Diagnosespezifikation auf Ebene Software-Baustein und Interaktionen

Systemarchitekturfunktionsintegrationstest

Input

- Validierte Software auf Ebene Software-Baustein und Interaktionen
- Akzeptierte DTC auf Systemfunktionsebene
- Validierte Diagnosespezifikation auf Ebene Software-Baustein und Interaktionen
- Funktionsbausteinenetzwerk und Interaktionen (SA Funktions-Design)
- Diagnose-Standard-Jobs (SA Funktions-Design)
- Erweiterung Diagnosespezifikation durch Zuordnung Funktionsbaustein austauschbare Einheit (SA Funktions-Design)

Aktivitäten

- Validiere Funktionsbausteine und ihre Interaktionen
- Beginne (iterative) vollständige Dokumentation Standard-Jobs und DTC
- Validiere Diagnosespezifikation auf dieser Ebene genau dann, wenn für jeden DTC mindestens eine für die Werkstatt zugängliche austauschbare Einheit existiert
- Dokumentiere, falls noch detaillierbarere tauschbare Einheiten möglich
- Spätestens hier muß Bestätigung DTC für Systemfunktionen erfolgen
- Validiere alle Diagnose-Standard-Jobs

→ grundlegende Diagnose aller Steuergeräte validiert

Output

- validiertes SA-Funktions-Design
- validierte grundlegende Diagnose aller Steuergeräte
- validierte Diagnosespezifikation auf Ebene Funktionsbaustein mit Interaktionen
- iterative Dokumentation auf Level Diagnose-Standard-Jobs und DTC

Systemarchitekturfunktionsakzeptanztest

Input

- validiertes SA Funktions-Design
- validierte grundlegende Diagnose
- validierte Diagnosespezifikation auf Ebene Funktionsbaustein mit Interaktionen
- iterative Dokumentation auf Ebene Diagnose-Standard-Jobs und DTC
- Liste Funktionsbausteine (aus Ebene SA-Funktionsspezifikation)
- Anforderungen Funktionsbausteine (aus Ebene SA-Funktionsspezifikation)
- Verfeinerung DTC auf Ebene Funktionsbausteine (aus Ebene SA-Funktionsspezifikation)

Aktivitäten

- Validiere Anforderungen an Funktionsbausteine
- Validiere Diagnosespezifikation auf Funktionsbausteinebene
- Validiere komponentenspezifische Diagnosejobs
  → alle Diagnosejobs validiert
- Validiere Systemfunktions-DTC genau dann wenn jede austauschbare Einheit zuweisbar und zugänglich
  → komponenteninterne Diagnose akzeptiert
- Beginne Diagnosejobs für komponentenübergreifende Kundenfunktionen
- Beginne Implementierung der DTC für Kundenfunktionen

Output

- Validierte Anforderungen Funktionsbausteine
- Validierte komponenteninterne Diagnose
- Validierte Diagnosespezifikation auf Ebene Funktionsbausteine
- Erste Diagnosejobs für komponentenübergreifende Kundenfunktionen

Systemarchitekturintegrationstest

Input

- Report, daß komponenteninterne Diagnose akzeptiert ist
- Validierte Diagnosespezifikation auf Ebene Funktionsbausteine
- Erste Diagnosejobs für komponentenübergreifende Kundenfunktionen
- Feature Function Network (SW Funktionsspezifikation)
- Anforderungen Feature Function Network (SW Funktionsspezifikation)

Aktivitäten

- Validiere Feature Function Network und Interaktionen
- Füge Beschreibung Diagnosetelegramme hinzu, sodaß die Komponenten kommunizieren können bezüglich Diagnose.

Output

- Validiertes Feature Function Network
- Diagnosespezifikation auf Ebene Funktionsbausteine mit Diagnosetelegramme

Systemarchitekturakzeptanztest

Input

- Validiertes Feature Function Network
- Diagnosespezifikation auf Ebene Funktionsbausteine mit Diagnosetelegramme
- Liste atomare Kundenfunktionen (SA-Anforderungsanalyse)
- Use Cases atomare Kundenfunktionen (SA-Anforderungsanalyse)
- Anforderungen atomare Kundenfunktionen (SA-Anforderungsanalyse)
- Diagnosespezifikation auf Ebene atomarer Kundenfunktionen (SAAnforderungsanalyse)

Aktivitäten

- Validiere System gegen Spezifikation atomare Kundenfunktionen
- Validiere komponentenübergreifende DTC
  $\rightarrow$ komponentenübergreifende Diagnose akzeptiert
- Validiere Diagnose gegen Spezifikation Diagnose auf Ebene atomarer Kundenfunktionen
- Update Diagnosedokumente

Output

- Validierte atomare Kundenfunktionen
- Validierte Diagnose für atomare Kundenfunktionen

Gesamtfahrzeugakzeptanztest/SOP

Input

- Validierte Diagnose für atomare Kundenfunktionen
- Validierte atomare Kundenfunktionen
- Feature Liste Fahrzeug (Gesamtfahrzeuganforderungsanalyse)
- Use Case Feature List (Gesamtfahrzeuganforderungsanalyse)
- Erste Anforderungen Diagnose (Gesamtfahrzeuganforderungsanalyse)

Aktivitäten

- Validiere Anforderungen Feature List
- Validiere komplette Diagnose genau dann wenn die generellen Diagnoseanforderungen aus Tabelle 2.2 erfüllt sind sowie zudem:
  - Diagnose bisher auf allen Ebenen akzeptiert
  - Abweichungen von Diagnosespezifikation(en) dokumentiert
  - Diagnoseziele erreicht oder übererfüllt

  $\rightarrow$ vollständige Diagnose akzeptiert
- Prüfe Einsetzbarkeit Diagnosedaten und Diagnosespezifikation für Facelift

Output

- Validierte vollständige Diagnose
- Fahrzeug akzeptiert und zur Produktion freigegeben
- Freigabe vollständige Diagnose für nachgeordnete Prozeßpartner Produktion und Werkstätten

A.4 Lebenszyklusprozeß für die effiziente Diagnose verteilter Steuergerätefunktionen

In Abschnitt 5.2.1 wurde ein Diagnoseprozeß über den Lebenszyklus, beginnend mit dem Produktionsanlauf des Fahrzeugs, vorgestellt. In diesem Abschnitt werden die einzelnen, auf [BMW06] und [BMW07] basierenden, Prozeßschritte im Detail dargelegt.

Facelift-Anforderungsanalyse

Input

- Baukastendatenbank
- Vorhandenes Gesamtfahrzeugkonzept mit Spezifikationen existierender Kundenfunktionen
- Existierende Diagnosespezifikation Gesamtfahrzeug
- Lastenheft allgemeine Anforderungen Diagnose (bspw. [BMW08])

Aktivitäten

- Wähle differenzierende atomare Kundenfunktionen anhand Use Cases/ aus Baukastendatenbank
  → Definiere Facelift-Konzept basierend auf Gesamtfahrzeugkonzept
- Bestimme kritische Themen aus vorhandenen Diagnosedaten für Facelift-Konzept
- Definiere weitere/ aktualisiere vorhandene Diagnoseanforderungen
  - Fehler von kundensichtbaren oder -erlebbaren Funktionen oder deren Einschränkungen müssen mit DTC gespeichert werden
  - DTC-Setzbedingungen müssen verifizierbar/ reproduzierbar sein
  - Jeder DTC-Eintrag muß komplette Dokumentation haben
- Berücksichtige für sicherheitsrelevante Systeme, daß Funktionsbausteine relevanten Standards genügen müssen, bspw. ISO 26262 [ISO09a]

Output

- Facelift Konzept mit neuen atomaren Kundenfunktionen
- Diagnosespezifikation auf Kundenfunktionsebene

Definition E/E-Systemarchitektur

Die Systemarchitektur des Fahrzeugs bleibt größtenteils die gleiche. Wie in der Phase Systemarchitekturanforderungsanalyse in Abschnitt A.3 beschrieben, wird beim Design der Systemarchitektur die Möglichkeit der Erweiterung des Fahrzeugs im Rahmen des Facelifts bzw. bestimmter Systemausstattungen vorgehalten.

Input

- Facelift Konzept auf Ebene atomarer Kundenfunktionen
- Spezifikationen vorhandener Kundenfunktionen
- Diagnosespezifikation mit kritischen Themen auf Kundenfunktionsebene

Aktivitäten

- Untersuche, welche Kundenfunktionen geändert werden müssen
- Für neue oder geänderte Kundenfunktionen:
  - Beschreibe Funktionen und deren Auswirkungen auf Gesamtsystem sowie auf vorhandene Kundenfunktionen
    → Update Systemarchitektur und Feature Network
  - Berücksichtige für sicherheitsrelevante Systeme, daß Ereignis-DTC gesetzt werden müssen von den Kommunikationspartnern
  - Berücksichtige für sicherheitsrelevante Systeme, daß Kundenfunktionen relevanten Standards genügen müssen, bspw. ISO 26262
  - Führe erweiterte FMEA auf neue/geänderte Kundenfunktionen und Interaktionen aus bzw. aktualisiere Diagnosemodell ähnlicher Funktionen/ Vorgängerfunktionen
  - Bestimme Auswirkungen der Fehler auf den Kunden
  - Weise erkannte Probleme Komponenten zu
  - Erste Überlegung austauschbare Einheit für neue Kundenfunktionen
- Für nicht geänderte Kundenfunktionen:
  - Untersuche, ob durch neue Funktionen Änderungsbedarf besteht; falls ja, beschreibe die Änderungen
  - Berücksichtige für sicherheitsrelevante Systeme, daß Ereignis-DTC gesetzt werden müssen
  - Analysiere anhand Lebenszyklusdaten ob Bedarf für Änderung austauschbarer Einheit bei Kundenfunktionen besteht
- Aktualisiere Diagnosespezifikation

Output

- Beschreibung Systemarchitektur und Feature Network für Facelift
- Anforderungen für neue/geänderte Systeme auf Ebene atomarer Kundenfunktionen
- Aktualisierte Spezifikationen für nicht geänderte Kundenfunktionen
- Diagnosespezifikation auf Ebene Kundenfunktionen und Interaktionen
- Erste Analyse austauschbare Einheiten für Kundenfunktionen

Design Funktionalitäten

Input

- Beschreibung Systemarchitektur und Feature-Network
- Anforderungen an Kundenfunktionen für neue/geänderte Systeme
- Aktualisierte Spezifikationen für nicht geänderte Kundenfunktionen
- Diagnosespezifikation auf Ebene Kundenfunktionen und Interaktionen
- Erste Analyse austauschbarer Einheiten für Kundenfunktion

Aktivitäten

- Untersuche, welche Funktionsbausteine geändert werden müssen aufgrund Interaktionen mit neuen/geänderten Kundenfunktionen
- Für neue Funktionsbausteine:
  - Erstelle Use Cases zur Identifikation Funktionsbausteine
  - Zerlege neue Kundenfunktionen in Funktionsbausteine
  - Identifiziere Anforderungen Funktionsbausteine
  - Berücksichtige für sicherheitsrelevante Systeme, daß Funktionsbausteine relevanten Standards genügen müssen, bspw. ISO 26262 [ISO09a]
- Erstelle erweiterte FMEA auf geänderte Funktionsbausteine und Interaktionen
- Erweitere Diagnosespezifikation um DTC-Spezifikation für Funktionsbausteine, sodaß austauschbare Einheit im Fehlerfalle identifizierbar ist
- Analysiere bei vorhandenen Funktionsbausteinen anhand Lebenszyklusdaten, ob definierte austauschbare Einheit auf diesem Level akzeptabel ist
- Prüfe, ob gewählte austauschbare Einheit zugänglich ist für Werkstatt
- Definiere Sicherheitsanforderungen auf Ebene Funktionsbausteine
- Beginne mit Implementierung Diagnose-Standard-Jobs

Output

- Liste aller Funktionsbausteine und ihrer Interaktionen
- Anforderungen Funktionsbausteine
- Diagnosespezifikation auf Ebene Funktionsbausteine und Interaktionen
- Analyse austauschbare Einheiten auf Ebene Funktionsbausteine
- Sicherheitsanforderungen auf Ebene Funktionsbausteine
- Erste Diagnose-Standard-Jobs

Plane Software

Aufgrund thematischer Ähnlichkeit werden sowohl die Software-Bausteine als auch die Software Units in dieser Phase zusammengefaßt.

Input

- Liste aller Funktionsbausteine und ihrer Interaktionen

- Anforderungen Funktionsbausteine
- Fertige Spezifikationen für nicht geänderte Software-Bausteine
- Diagnosespezifikation auf Ebene Funktionsbausteine und Interaktionen
- Analyse austauschbare Einheiten auf Ebene Funktionsbausteine
- Sicherheitsanforderungen auf Ebene Funktionsbausteine

Aktivitäten

- Untersuche welche Software-Bausteine geändert werden müssen aufgrund Interaktionen mit neuen/geänderten Funktionsbausteinen
- Berücksichtige für sicherheitsrelevante Systeme, daß Ereignis-DTC gesetzt werden müssen
- Für neue/geänderte Software-Bausteine:
  - Erstelle Use Cases zur Identifikation Software-Bausteine
  - Zerlege neue Funktionsbausteine in Software-Bausteine
  - Erstelle Anforderungen an diese Software-Bausteine
  - Beschreibe Software-Bausteine und ihre Interaktionen mit MSC
  - Berücksichtige für sicherheitsrelevante Systeme, daß Software-Bausteine relevanten Standards genügen müssen, bspw. ISO 26262
- Erstelle Diagnosemodell auf Ebene Software-Bausteine und ihren Interaktionen oder verwende vorhandene, gewartete Bausteine wieder
- Berücksichtige für sicherheitsrelevante Systeme, daß Ereignisfehler gesetzt werden müssen. Aktualisiere Zuweisung dieser Fehler an interagierende Software-Bausteine.
- Prüfe genauere Zuweisung der vorhandenen DTC der Kundenfunktionen auf Software-Bausteine aus Qualitätsmanagementgründen. Diese Verfeinerung kann aufgrund von Lebenszyklusdaten erfolgen
- Untersuche, welche Software Units aufgrund von Software-Bausteinen geändert werden müssen
- Berücksichtige für sicherheitsrelevante Systeme, daß Ereignis-DTC gesetzt werden müssen
- Für neue/geänderte Software Units:
  - Zerlege neue Software-Bausteine in Software Units
  - Erstelle Anforderungen an diese Software Units
  - Beschreibe Software Units, bspw. mit State Machines
  - Berücksichtige für sicherheitsrelevante Systeme, daß Software Units relevanten Standards genügen müssen, bspw. ISO 26262
  - Erstelle Diagnosemodell auf geänderte Software Units
- Prüfe, basierend auf Lebenszyklusdaten, noch genauere Verfeinerung DTC auf Ebene Software Units, wiederum aus Qualitätsmanagementgründen

- Passe Diagnosespezifikation an

Output

- Liste aller Software-Bausteine und ihren Interaktionen
- Anforderungen Software-Bausteine
- Diagnosespezifikation auf Ebene Software-Bausteine und Interaktionen
- Analyse austauschbare Einheiten auf Ebene Software-Bausteine
- Sicherheitsanforderungen auf Ebene Software-Bausteine

Vervollständige Funktionalitäten

Input

- Liste aller Software-Bausteine und SW-Units und ihrer Interaktionen
- Anforderungen Software-Bausteine und Software Units
- Diagnosespezifikation auf Ebenen Software-Bausteine, SW-Units und Interaktionen
- Analyse austauschbare Einheiten auf Ebenen Software-Bausteine und SW-Units
- Sicherheitsanforderungen auf Ebenen Software-Bausteine und SW-Units

Aktivitäten

- Beginne Implementierung der Software
- Beginne Implementierung der Diagnose
  - Beginne mit Programmierung Diagnose
  - Erweitere Standard-Jobs so durch komponentenspezifische Jobs, sodaß gewählte kleinste austauschbare Einheit identifizierbar ist
  - Implementiere DTC für Systemfunktionen
  - Erweitere Diagnose-Job und DTC-Liste

Output

- Implementierung Diagnose
- Implementierung Software Units und Software-Bausteine

Software-Test

Input

- Implementierung Diagnose
- Implementierung Software Units und Software-Bausteine
- Sicherheitsanforderungen auf Ebene Software-Bausteine
- Anforderungen Software-Bausteine und Software Units (Phase Plane Software)

- Diagnosespezifikation auf Ebenen SW-Bausteine, SW-Units (Phase Plane Software)
- Analyse austauschbare Einheiten auf Ebenen SW-Bausteine und SW-Units (Phase Plane Software)
- Mögliche Verfeinerung DTC (Phase Plane Software)

Aktivitäten

- Validiere Software auf Software Unit-Ebene
- Validiere Diagnosespezifikation auf Ebene SW-Unit
- Passe Diagnosespezifikation an Testergebnisse an ($\rightarrow$ transitive Abhängigkeiten!)
- Validiere Diagnosespezifikation für DTC auf Software Unit-Ebene
- Validiere Software-Bausteine gegen Design
- Validiere Diagnosespezifikation auf Ebene SW-Baustein
- Validiere Diagnosespezifikation für DTC auf Software-Bausteinebene
- Passe Diagnosespezifikation an Testergebnisse an ($\rightarrow$ transitive Abhängigkeiten!)

Output

- Validierte Software auf Software-Bausteinebene
- Validierte Diagnosespezifikation auf SW-Bausteinebene
- Validierte DTC auf Software-Bausteinebene

Bestätige Funktionalitäten

Input

- Validierte Software auf Software-Bausteinebene
- Validierte Diagnosespezifikation auf SW-Bausteinebene
- Validierte DTC auf Software-Bausteinebene
- Liste aller Funktionsbausteine und Interaktionen (Phase Design Funktionalitäten)
- Anforderungen Funktionsbausteine (Phase Design Funktionalitäten)
- Diagnosespezifikation auf Ebene Funktionsbausteine und Interaktionen (Phase Design Funktionalitäten)
- Sicherheitsanforderungen auf Ebene Funktionsbausteine (Phase Design Funktionalitäten)
- Erste Diagnose-Standard-Jobs (Phase Design Funktionalitäten)

Aktivitäten

- Validiere Funktionsbausteine und Interaktionen

- Validiere Diagnosespezifikation auf Ebene Funktionsbaustein
- Bestätige DTC für Systemfunktionen (je nach Genauigkeit DTC)
- Passe Diagnose-Job und DTC-Liste sowie restliches Diagnosemodell an Testergebnisse an (→ transitive Abhängigkeiten!)

Output

- Validierte Interaktionen auf Funktionsbausteinebene
- Validierte Diagnose auf Ebene Funktionsbausteine
- Report, daß DTC auf Funktionsbausteinebene bestätigt, falls so gewählt

Bestätige Systemarchitektur

Input

- Validierte Interaktionen auf Funktionsbausteinebene
- Validierte Diagnose auf Ebene Funktionsbausteine
- DTC auf Funktionsbausteinebene bestätigt
- Beschreibung System Architektur und Feature Network (Phase Def. E/E-SA)
- Anforderungen Kundenfunktionen für neue/geänderte Systeme (Phase Def. E/E-SA)
- Fertige Spezifikationen für nicht geänderte Kundenfunktionen (Phase Def. E/E-SA)
- Diagnosespezifikation auf Ebene Kundenfunktionen und Interaktionen (Phase Def. E/E-SA)
- Erste Analyse austauschbare Einheiten für vorhandene/alte Kundenfunktion (Phase Def. E/E-SA)

Aktivitäten

- Validiere erweiterte/neue Kundenfunktionalitäten und Interaktionen
- Validiere komponentenübergreifende DTC
  → Diagnose für atomare Kundenfunktionen akzeptiert
- Aktualisiere Diagnosedokumente

Output

- Validierte Diagnose auf Ebene Kundenfunktionen
- Validierte atomare Kundenfunktionen

Facelift-Akzeptanztest

Input

- Validierte atomare Kundenfunktionen und Interaktionen

- Feature Liste Fahrzeug (Gesamtfahrzeug Facelift-Konzept)
- Use Case Modell Feature List (Gesamtfahrzeug Facelift-Konzept)
- generelle Anforderungen Diagnose (Gesamtfahrzeug Facelift-Konzept)

Aktivitäten

- Validiere Anforderungen Feature List
- Validiere Diagnose genau dann wenn allgemeine Anforderungen Diagnose erfüllt (bspw. Dokumentation vollständig für Werkstatt)

Output

- Fahrzeug akzeptiert und zur Produktion freigegeben
- Diagnose vollständig validiert und akzeptiert

Anhang B

Diagnosemodell des Fallbeispiels Fensterheber

c Diagnosemodell für Fallbeispiel Fensterheberfunktion automatisches Schließen.
c Mapping:
c x1 := C2 : Ausgabe Fensterheber
c x2 := CC1 : Fenster bewegt sich nicht dauerhaft
c x3 := CC2 : Fensterlauf stoppt abrupt
c x4 := Fault1 : Kabel von Schaltersensor zu ECU defekt
c x5 := Fault2 : Kabel von ECU zu Fensterhebermotor defekt
c x6 := Fault3 : falsche Polarität Hall-Sensoren
c x7 := Fault4 : Schalter defekt
c x8 := Fault5 : Schaltersensor defekt
c x9 := Fault6 : ECU defekt
c x10 := Fault7 : Fensterheber defekt
c x11 := Fault8 : fehlende Rechenzeit für Einklemmschutzalgorithmus
c x12 := Fault9 : Variablenüberlauf in Einklemmschutzalgorithmus
c x13 := G1 : Prüfe Kabel vom Schalter zum Steuergerät auf Strom
c x14 := G2 : Prüfe Kabel vom Steuergerät zum Fenster auf Strom
c x15 := G3 : Prüfe Kabel von Hall-Sensoren zum Steuergerät auf Strom
c x16 := T1 : keine Änderung diskretisiertes Hall-Signal nach Fensterbewegung
c x18 := T2 : kein diskretisiertes Hall-Signal nach Fensterbewegung
c x20 := T3 : Hall-Signal deutet auf Fensterbewegung in falsche Richtung hin
c x22 := T4 : Überlauf des Drehzahlgradienten
c x24 := T5 : nicht genügend Rechenzeit für Einklemmalgorithmus
c x17 := DTC1, x19 := DTC2, x21 := DTC3, x23 := DTC4, x25 := DTC5
c x26 := Gegenmaßnahme CM1: Fenster permanent deaktivieren
c x27 := Gegenmaßnahme CM2: Fenster abrupt stoppen
p cnf 27 23
-1 2 3 0 c Kundenebene: Ausgabe nicht wie erwartet $\rightarrow CC1 \vee CC2$
-2 4 5 6 7 8 9 10 0 c Werkstattebene: Fenster bewegt sich nicht dauerhaft $\rightarrow F1 \vee ... \vee F7$
-3 11 12 0 c Fensterlauf stoppt abrupt $\rightarrow F8 \vee F9$ vorhanden
-1 4 5 6 7 8 9 10 11 12 0 c Werkstattebene: Ausgabe nicht wie erwartet $\rightarrow F1 \vee ... \vee F9$

-13 4 7 8 0 c Werkstattebene: Auswertung Monitor G1
-14 5 9 0 c Werkstattebene: Auswertung Monitor G2
-15 6 10 0 c Werkstattebene: Auswertung Monitor G3
-16 17 0 c ECU-Ebene: T1 $\rightarrow$ DTC1
-18 19 0 c ECU-Ebene: T2 $\rightarrow$ DTC2
-20 21 0 c ECU-Ebene: T3 $\rightarrow$ DTC3
-22 23 0 c ECU-Ebene: T4 $\rightarrow$ DTC4
-24 25 0 c ECU-Ebene: T5 $\rightarrow$ DTC5
-21 6 0 c ECU-Ebene: DTC3 $\rightarrow$ F3
-23 11 0 c ECU-Ebene: DTC4 $\rightarrow$ F8
-25 12 0 c ECU-Ebene: DTC5 $\rightarrow$ F9
-6 26 0 c ECU-Ebene: Fehler F3 $\rightarrow$ Gegenmaßnahme CM1
-26 1 0 c ECU-Ebene: CM1 $\rightarrow \neg C2$
-11 27 0 c ECU-Ebene: Fehler F8 $\rightarrow$ Gegenmaßnahme CM2
-12 27 0 c ECU-Ebene: Fehler F9 $\rightarrow$ Gegenmaßnahme CM2
-27 1 0 c ECU-Ebene: CM2 $\rightarrow \neg C2$
14 17 5 0 c ECU und Werkstatt-Ebene: $(\neg G2 \wedge DTC1) \rightarrow F2 \equiv G2 \vee DTC1 \vee F2$
14 -17 9 0 c ECU und Werkstatt-Ebene: $(\neg G2 \wedge \neg DTC1) \rightarrow F6 \equiv G2 \vee \neg DTC1 \vee F6$
-15 19 10 0 c ECU und Werkstatt-Ebene: $(\neg G3 \wedge DTC2) \rightarrow F7 \equiv G3 \vee DTC2 \vee F7$
c ab hier werden dann zur Laufzeit die ausgewerteten Beobachtungen als Klauseln
c hinzugefügt