Herausforderungen beim Testen von Fahrerassistenzsystemen

Remo Lachmann und Ina Schaefer

Institut für Softwaretechnik und Fahrzeuginformatik
Technische Universität Braunschweig, 38106 Braunschweig
Email: {r.lachmann, i.schaefer}@tu-bs.de

Abstract: Moderne Fahrerassistenzsysteme (FAS) werden immer komplexer. Wegen ihres sicherheitskritischen Charakters müssen sie ausgiebig getestet werden, um Serienreife zu erreichen. Das Testen von FAS bringt verschiedene Herausforderungen mit sich, die überwunden werden müssen. Wir stellen in diesem Artikel Herausforderungen aus verschiedenen Domänen vor und präsentieren Lösungsansätze, wie diese überwunden werden können. Die untersuchten Herausforderungen bestehen im Softwareentwicklungsprozess, der Testphase selbst sowie in der Organisation eines FAS-Entwicklungsprojekts.

1 Einleitung und Motivation

Unter dem Begriff Fahrerassistenzsysteme (FAS) werden in der Literatur verschiedene technische Hilfssysteme zusammengefasst, wobei die Bezeichnung meistens für *Fahrerassistenzsysteme mit maschineller Wirkung* [Mau12] verwendet wird. Kraiss [Kra98] definiert den Begriff als redundant-paralleles System, bei dem gewisse Aufgaben von Mensch und Maschine parallel erledigt werden. FAS sollen dem Fahrer Beistand bzw. Mithilfe beim Steuern seines Fahrzeugs geben. Die Entwicklung derartiger Systeme ist mittlerweile bei vielen Automobilherstellern in die Serienproduktion von Fahrzeugen integriert worden. Die entwickelten Systeme werden zunehmend komplexer. Vor allem in der Domäne der automatisierten Fahrmanöver wird erheblicher Aufwand betrieben, um diese noch umfangreicher zu gestalten. Ein Ziel hierbei ist es, das Fahrzeugverhalten autonom durch Software zu steuern. Forschungsergebnisse in diesem Bereich wurden beispielsweise in der *DARPA Urban Challenge* präsentiert [BIS10]. Hierbei müssen teilnehmende Fahrzeuge eine bestimmte Strecke ohne menschlichen Fahrer zurücklegen.

Auf Grund sicherheitskritischer Aspekte in der Automobilentwicklung ist es zwingend notwendig, FAS ausführlich zu testen. Bevor ein FAS im normalen Straßenverkehr freigegeben werden kann, muss der Entwickler sicherstellen, dass das erwartete Risiko der betroffenen Verkehrsteilnehmer durch den Einsatz von FAS nicht höher ist, als das Verkehrsrisiko des derzeitigen Istzustands [Hom05]. Dies führt zu einem erheblichen Testaufwand, der von den Entwicklern bewältigt werden muss. Testen gehört allgemein zu den zeit- und kosten-intensivsten Phasen der Softwareentwicklung [Lig09]. Daher müssen neue Metho-

den gefunden werden, um FAS effizient zu testen und gleichzeitig sicherzustellen, dass sie den gestellten Anforderungen von Kunden, Herstellern und Staat gerecht werden [Mau12]. So wird das umfangreiche Testen von Software im Automobil im Standard ISO 26262 - "Road vehicles – Functional safety" explizit von den Automobilherstellern gefordert.

Neben den sicherheitskritischen Aspekten gibt es weitere, das Testen erschwerende Faktoren, die bei der Entwicklung von FAS beachtet werden müssen. So werden derartige Softwaresysteme oft in Form von verteilten Komponenten entworfen. Hierbei werden verschiedene Komponenten meist von unterschiedlichen Zulieferern oder internen Teams entwickelt, um im Anschluss zentral, meist beim Automobilhersteller selbst, zu einem Gesamtsystem integriert zu werden. Dies hat zur Folge, dass der Quellcode sämtlicher Komponenten meist nicht zur Verfügung steht [HLS99]. Es müssen demnach Blackbox-Testverfahren angewandt werden, die für Integrationstests eingesetzt werden können.

Bei FAS handelt es sich meist um komplexe Softwareeinheiten, die anhand unterschiedlichster Parameter in verschiedenen Situationen Entscheidungen treffen müssen. Um eine korrekte Funktionalität zu gewährleisten, müssten die Hersteller theoretisch sämtliche auftretende Situationen und Umstände, in die das Fahrzeug in seiner Einsatzzeit typischerweise gelangen könnte, abbilden und prüfen. Ein derartiger Testaufwand würde sich mit Realtests jedoch nicht umsetzen lassen [WW12].

In der Literatur werden verschiedene Probleme beim Testen von FAS dargestellt. Zusätzlich konnten in der industriellen Praxis ähnliche Herausforderungen ermittelt werden. Im Rahmen dieser Arbeit stellen wir einige dieser Herausforderungen vor und geben jeweils mögliche Ansätze zur Problemlösung beim Testen von FAS. In Abbildung 1 sind die von uns betrachteten Herausforderungen dargestellt. Zunächst wird der *Softwareentwicklungsprozess* untersucht. Dabei gehen wir auf mangelhafte bzw. fehlende Anforderungs- und Architekturspezifikationen ein, die für einen strukturierten Testprozess wichtig sind. Anschließend werden Herausforderungen, die beim Testen auftreten, vorgestellt. Wir untersuchen hierbei fehlende Testkonzepte, den zu hohen Testaufwand und die unstrukturierte Testfallerstellung. Als letzte Kategorie untersuchen wir Herausforderungen, die in der *Organisation* von Projekten auftreten können, insbesondere die verteilte Entwicklung und heterogenen Entwicklungsumgebungen.

Dieser Artikel ist wie folgt strukturiert. Im zweiten Kapitel untersuchen wir Herausforderungen, die im Softwareentwicklungsprozess entstehen können. Kapitel 3 beschäftigt sich mit der Testphase. Im vierten Kapitel werden organisatorische Faktoren beleuchtet, die beim Testen von FAS zu beachten sind. Im letzten Kapitel wird eine Zusammenfassung sowie ein Ausblick gegeben.

2 Softwareentwicklungsprozess

Beim Testen von FAS haben wir im Softwareentwicklungsprozess zwei Herausforderungen ermittelt, die bei der Spezifikation des Systems auftreten. Wir unterscheiden die Problematik der *Anforderungsspezifikation* und die der *Architekturspezifikation*.

2.1 Erfassung der Anforderungsspezifikation

In der klassischen Softwareentwicklung wird für kritische Anwendungen oft das V-Modell [SZ12] als Vorgehensmodell verwendet. In Verbindung mit dem *Automotive SPICE*-Standard [Aut10] wird das V-Modell auch im Rahmen der Entwicklung von FAS eingesetzt. Bei diesem Vorgehen wird mit einer Anforderungsanalyse begonnen. Das Ziel ist es, vor der Implementierung der eigentlichen Software, eine umfassende Softwarespezifikation zu erstellen, welche die Anforderungen an das System enthält. Dieses gibt die daraus herzuleitenden Testfälle sowie deren erwartete Ergebnisse vor. Ohne Einblick in die Programmstruktur, stellt die Spezifikation den einzigen Anhaltspunkt für die Testfallerstellung dar.

Die Softwarespezifikation der verschiedenen Module spielt eine Schlüsselrolle beim Tes-

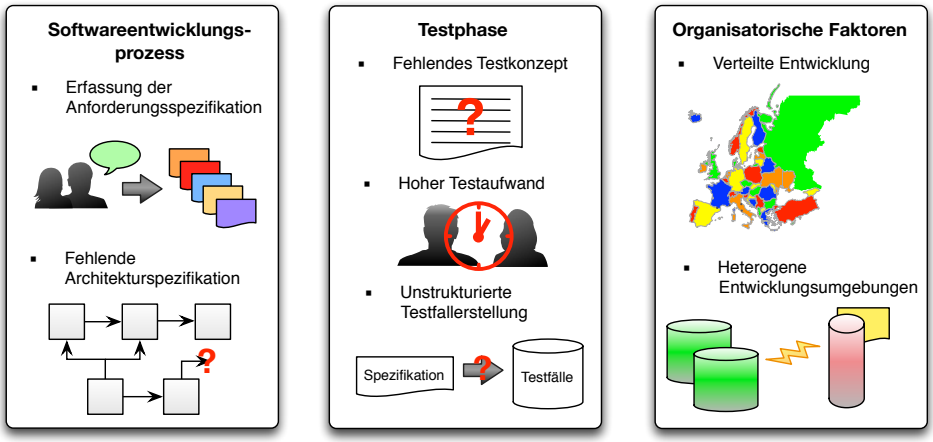

Abbildung 1: Überblick über die untersuchten Herausforderungen

ten von FAS. In der Realität sind Softwarespezifikationen jedoch meist unzureichend gepflegt, liegen nur in Prosa vor oder fehlen komplett. Dies kann verschiedene Gründe haben, z.B. kann es sich um prototypische Entwicklungsszenarien handeln, bei denen Zeit, Geld und Personal stark begrenzt sind und der Fokus der Entwickler auf der Implementierung eines lauffähigen Systems gelegt wird. Oftmals unterliegen Spezifikationen einer Evolution, die zu Inkonsistenzen und Redundanzen führen kann.

Lösungsansätze: Maurer [Mau12] sieht einen Schwerpunkt in der korrekten Erfassung der Anforderungen an das FAS. Dabei sollte eine fundierte Auswahl getroffen werden, zwischen den realisierbaren und den wünschenswerten, allerdings noch nicht realisierbaren Assistenzfunktionen. Diese Auswahl soll vor der Entwicklung der eigentlichen Systeme stattfinden.

Weber und Weisbrod [WW02] stellen fest, dass in der Praxis oftmals das Management von Anforderungen problematischer ist, als die eigentliche Erfassung. Dies begründet sich daher, dass in der Automobilindustrie Produktfamilien entwickelt werden, wobei Anforderungen evolutionär weiterentwickelt werden, anstatt sie stets von neuem aufzunehmen. Es gilt demnach, bereits vorhandenes Expertenwissen möglichst effizient zu verwalten. Hierfür eignen sich Anforderungsmanagement-Werkzeuge. Weber und Weisbrod stellen die folgenden Anforderungen an ein solches Werkzeug: Es muss die Verarbeitung von textuellen sowie anderen Objekten ermöglichen sowie eine Verlinkung derer untereinander, um Rückverfolgbarkeit zu ermöglichen. Es muss möglich sein, *Views* für einzelne Beteiligte zu definieren, so dass diese ausschließlich mit für sie relevanten Informationen arbeiten. Weiterhin sollten die darin definierten Spezifikationen auf Basis eines gemeinsamen Metamodell definiert sein. Dieses sollte auch externen Zulieferern zur Verfügung gestellt werden. Dies ermöglicht eine ”gemeinsame” Sprache, um Missverständnissen bei der Spezifikationsdefinition vorzubeugen.

Grundsätzlich ist es wichtig, die Arbeit der Ingenieure durch geeignete Werkzeuge zu unterstützen, um die Verwaltung, Nachvollziehbarkeit und Eindeutigkeit von definierten Spezifikation zu ermöglichen.

Maurer stellt zudem ein, zum V-Modell alternatives, iteratives Entwicklungsmodell vor, welches den Fokus auf die Erfassung der Anforderungen setzt. Ein besonderes Augenmerk sollte auf Funktionsdefinitionen, technischer Machbarkeit, Produktionssicherheit, Human Factors und der aktuellen Marktsituation gelegt werden. Der erste Teil einer Iterationsschleife führt zu einem Zwischenergebnis, dass bei unzureichender Qualität zu einer Abkürzung der Iteration führen kann, die anschließend von vorn beginnt. Somit wird sichergestellt, das anschließende Prozesse, die den Aufbau von prototypischen Systemen umfassen, erst durchgeführt werden, wenn die beteiligten Experten eine geeignete Funktionsdefinition gefunden haben. Diese sollte möglichst alle theoretisch gefundenen Auslegungskonflikte auflösen.

2.2 Fehlende Architekturspezifikation

Um ein komplexes Softwaresystem, wie ein FAS, zu entwickeln und zu testen, ist es dringend notwendig, eine dem System zu Grunde liegende Architektur zu definieren. Diese verschafft allen Beteiligten einen Überblick über die Struktur des Gesamtsystems. Weiterhin werden die verschiedenen Schnittstellen der einzelnen Komponenten auf Architekturebene definiert. Architekturen können zusätzlich zur Projektorganisation verwendet werden, um das Projekt zu strukturieren.

In der Praxis stellt sich jedoch oftmals heraus, dass Architekturspezifikationen entweder oberflächlich, veraltet oder gar nicht vorliegen. Vor allem im Bereich von sicherheitskritischen Systemen ist eine korrekte Kommunikation der beteiligten Komponenten absolut notwendig. Beim Testen von FAS kommt erschwerend hinzu, dass viele der Komponenten bei unterschiedlichen Zulieferern entwickelt werden. Hierbei können es zu fehlerhafte Schnittstellenspezifikationen entstehen, die dazu führen, dass entsprechende Fehler erst bei der Integration der Komponenten auftreten. Der Testaufwand lässt sich durch eine vollständige Spezifikation der Komponenten jedoch bereits im Vorhinein verringern.

Der Zugriff auf die interne Komponentenstruktur in Form von Quellcode ist in diesen Fällen meist nicht gegeben, d.h. die Softwaremodule unterliegen einer Blackbox-Annahme. Dieser Umstand wird in der Literatur als *component-based testing* bezeichnet [HLS99].

Lösungsansatz: Erfahrungen aus der Praxis machten deutlich, dass die Softwarearchitektur bei allen Beteiligten bekannt sein muss, um effizient eingesetzt werden zu können und Problemen bei der Schnittstellendefinition vorzubeugen. Dies führt langfristig zu einem reduzierten Aufwand beim Integrationstesten, effizienterer Arbeit und höherer Qualität. Hierfür eignet es sich, einen Softwarearchitekten einzustellen, der das Wissen über die Softwarearchitektur besitzt und bei deren Definition eine zentrale Rolle spielt. Diese Person sollte in Kontakt mit allen Entwicklerteams stehen, um eventuelle Änderungen, die in der Architektur anfallen, direkt kommunizieren zu können.

Bei der Entwicklung von FAS werden Architekturspezifikationen auf unterschiedlichen Abstraktionsebenen verwendet [SZ10]. Beispiele hierfür sind logische und technische Architekturen. Alle Architekturbeschreibungen können das Testen unterstützen. Bei der Entwicklung von Softwarearchitekturen sollten die Grundprinzipien der Softwarearchitekturentwicklung beachtet werden, wie sie in der Literatur definiert werden [PBG04, Reu06]. Beispiele hierfür sind das Abstraktion, hierarchische Dekomposition, das Anstreben einer hohen Kohäsion und niedrigen Kopplung etc. Die grobe Systemarchitektur und die darin enthalten Kommunikationsschnittstellen sollten vorab definiert werden, um spätere Integrationsprozesse zu verbessern. Die Komponenten sollten von den einzelnen Entwicklern bzw. Zulieferern spezifiziert werden. So lassen sich Testfälle für Komponententests bereits bei den Entwicklern entwerfen und durchführen. Integrationstests können bei der Integration der Module der entsprechenden Spezifikation entnommen werden.

3 Testphase

In der eigentlichen Testphase der FAS-Entwicklung muss die Qualität der Software gesichert werden, um das entwickelte System für die Straße zulässig zu machen. Im Folgenden stellen wir ausgewählte Herausforderungen vor, die der Literatur und in Industrieprojekten für das Testen von FAS festgestellt werden konnten.

3.1 Fehlendes Testkonzept

Damit der zeit-und kostenintensive Prozess des Testens im angestrebten Rahmen bleibt, wird ein Testkonzept benötigt, welches die durchzuführenden Testaktivitäten eines Testprojekts umfasst [SRWL11]. Ein Testkonzept sollte u.a. die zu testenden Testobjekte, Qualitätsmerkmale, Teststrategien, Dokumentation, Testendekriterien etc. umfassen. Oftmals liegen Testkonzepte jedoch nicht vor. Vielmehr werden Komponententests von den Entwicklern ”irgendwie” durchgeführt, bis eine Deadline erreicht ist oder in einem Umfang, den die aktuelle Kapazität zulässt. Dies beginnt meist mit fehlenden Testern bzw. Testverantwortlichen.

Lösungsansatz: Für das Testen von FAS ist es absolut notwendig, umfangreiche Tests durchzuführen. Daher wird ein klares Testkonzept benötigt. Der erste Schritt ein solches Testkonzept zu erhalten, ist die Ernennung eines Testmanagers. Diesem sollte eine Menge von Testern zur Verfügung stellen. Die Aufgabe des Testteams ist es, das Testkonzept zu entwerfen. Hierfür liefert der Standard IEEE 829 eine Vorlage, welche Struktur und Inhalte ein Testdokument haben sollte . Es umfasst u.a. auch den *Testplan*, der praxisnahe Strukturen für ein Testkonzept vorschlägt. Das resultierende Testkonzept muss auf den Anwendungsfall adaptiert werden. Der Testplan besitzt ebenfalls eine hohe Priorität, da er den langfristigen Testverlauf vorgibt und Meilensteine für die Tester vorgibt. Ohne Testplan gibt es keine anzustrebenden Testziele.

3.2 Hoher Testaufwand

Die hohen Sicherheitsanforderungen an FAS, die durch Sicherheitstandards, Hersteller und Kunden gefordert werden, machen ein ausführliches Testen aller beteiligten Komponenten und deren Zusammenspiel unverzichtbar. Es stellt sich jedoch oftmals heraus, dass das Testen solcher Systeme enormen Aufwand mit sich bringt [Mau12]. Dieser Problematik muss mit entsprechenden Ressourcen im Bereich des Testens begegnet werden.

Jedoch reicht der Einsatz von mehr Personal, der oftmals der erste Schritt zu einem verbesserten Testverfahren ist, nicht aus. So wird beispielsweise von FAS, die autonomes Fahren ermöglichen sollen, gefordert, dass das von ihnen ausgehende Risiko nicht höher sein darf, als das bei heutigen Fahrzeugen. Ein autonomes Fahrzeug müsste demnach auf das auftretende Unfallrisiko geprüft werden. Schwere Unfälle treten in Deutschland jedoch nur alle

5 Millionen Fahrkilometer auf. Ausgiebiges Testen würde die zu befahrene Strecke enorm vergrößern, was Dauerlauftests wirtschaftlich nicht vertretbar machen würde [WW12]. Demnach müssen Methoden und Prozesse gefunden werden, die dem hohen Testaufwand entgegensteuern und das Testen von FAS ermöglichen.

Lösungsansätze: Um die steigenden Anforderungen an den Testprozess zu gewährleisten, werden Testfälle für HiL-Prüfstände in der Praxis durch den modellgetriebenen Testentwicklungsansatz *EXAM* modelliert, ausgeführt und anschließend ausgewertet [ZT10]. EXAM-Modelle stellen eine Teilmenge der UML2-Spezifikation dar [OMG12]. Die Methode unterstützt viele Prinzipien der agilen Softwareentwicklung und ermöglicht einen hohen Grad an Testautomatisierung, was zu einer gesteigerten Testeffizienz führt, da Tests z.B. nachts autonom durchgeführt werden können. Der EXAM-Ansatz wurde in Form einer frei verfügbaren Toolsuite realisiert[1].

Modelle können auch genutzt werden, um das Testen möglichst früh in die Entwicklung von FAS zu integrieren. Ein modellbasierter Entwicklungsansatz erlaubt es, bereits in frühen Entwicklungsphasen zu verifizieren, ob sich Algorithmen und Verfahren entsprechend der Erwartung verhalten, oder ob diese fehlerbehaftet sind [GS11]. Derartige *Model-in-the-loop*-Verfahren (MiL) erlauben es, komplexe Testszenarien am Schreibtisch durchzuspielen, ohne auf ein lauffähiges Komplettsystem oder aufgebautes Testfahrzeug warten zu müssen. Weiterhin erlauben MiL-Tests komplexe Zeitverlaufs-und Signalsverlaufsanalysen, die bei anschließenden *Hardware-in-the-loop*-Tests (HiL) wiederverwendet werden können [GS11].

Eine Verlagerung von Realtests hin zu Testmodellen führt zu einer Kostenersparnis, da früh gefundene Fehler leichter zu beheben sind als Fehler, die erst im aggregierten System gefunden werden. Auch können Modelle genutzt werden, um Testfälle für die Realfahrten mit dem Versuchsfahrzeug zu entwerfen. Derartige Tests könnten beispielsweise diejenigen sein, die in virtuellen Umgebungen mehrfach fehlgeschlagen sind und als kritisch betrachtet werden.

Im Rahmen von FAS eignen sich für modellbasiertes Testen Umgebungs-und Funktionsmodelle, die z.B. durch Tools wie *Messina* [Ber], welches systematisches Testen durch realitätsnahe Umgebungssimulationen erlaubt, oder *Virtual Test Drive* [Aud] bereit gestellt werden. Letzteres erlaubt die Erstellung und Verwendung von komplexen Streckenszenarien unter verschiedensten Bedingungen zum Testen von Steuergeräten. Als Beispiel für in VTD modellierte Szenarien ist eine Strecke in zwei unterschiedlichen Situationen in Abbildung 2 dargestellt. Die Grafik zeigt die gleiche Fahrbahn in zwei unterschiedlichen Situationen, einmal ohne Verkehr bei Sonnenschein und im zweiten Bild mit anderen Fahrzeugen, Engstelle und Regen. Dies macht deutlich, dass sich virtuell Szenarien entwerfen lassen, die in der Realität nur schwer oder gar nicht konstruiert werden können.

Ein wichtiger Faktor beim Einsatz derartiger Systeme ist die Verwendung eines ausreichend präzisen Fahrzeugmodells, welches das reale Fahrzeug in der virtuellen Umgebung repräsentiert. Weicht dieses Modell zu sehr von der Realität ab, werden eventuell Fehler gefunden, die keine sind (false positives), oder es werden bestimmte Fehler nicht gefunden

[1] EXAM-Website: www.exam-ta.de/

Abbildung 2: Beispielszenarien in VTD [SSLM13]

(false negatives). Auf Grund der Komplexität moderner Fahrzeuge können Fahrzeugmodelle nicht beliebig präzise sein. Daher müssen Testfälle auf ihre Sinnhaftigkeit überprüft werden, um sicherzustellen, dass es sich um geeignete Testfälle für ein virtuelles Szenario handelt. Es ist zum Beispiel nur bedingt sinnvoll, das Verhalten der Fahrzeugregelung bis ins Detail virtuell zu testen, da das echte Fahrzeugverhalten nicht absolut realistisch nachgebildet werden kann.

Winner entwickelte einen Ansatz [Win01], der genutzt werden kann, um die Fehlerwahrscheinlichkeiten von bestimmten Fahrerassistenzfunktionen zu erproben. Hierfür wäre im Normalfall ein sehr hoher Test- bzw. Zeitaufwand notwendig. Ein möglicher Ansatz, um diesem Problem zu begegnen, ist es, die entsprechende Funktion prototypisch in Kundenfahrzeuge zu implementieren. Das Assistenzsystem hat jedoch keinen Einfluss auf das Fahrzeug, d.h. es gibt keine Verbindung zu den Aktuatoren. Stattdessen werden entsprechende Berechnungen und Anweisungen auf einem Datenspeicher protokolliert, um diese im Nachhinein auswerten zu können. Somit können Fehler in den Systemen bei ausgiebigen Tests in realen Szenarien gefahrlos entdeckt werden. Dies hat einen erhöhten Testgrad zur Folge, der keine weiteren internen Ressourcen benötigt. Es lässt sich jedoch noch nicht abschätzen, inwiefern die Testdauer steigt bzw. wie lang derartige Systeme durchschnittlich erprobt werden sollten. Diese Fragen sollten in zukünftiger Grundlagenforschung und empirisch in Fallstudien beleuchtet werden.

Regressionstests, die nach Änderungen an einem System vorgenommen werden, stellen im Bereich von autonomen Systemen ein Problem dar, da sie den hohen Testaufwand des Grundsystemes wiederholen. Hierfür stellen Winner und Weitzel einen Ausweg in Form von probabilistischen Bewertungen vor. Das System besitzt hierbei eine *Perzeptions-*, *Kognitions-* und *Aktionsleistung*, die messbar sein muss und höher oder zumindest genauso hoch wie bei der Vergleichsgruppe sein [WW12]. Kann dies nachgewiesen werden, kann das System freigegeben werden, da es sich auch in bis dato unbekannten oder ungeprüften Situationen ähnlich oder besser verhalten wird als das menschliche Pendant. Eine Metrik, die diese Leistung ausdrückt, ist laut den Autoren jedoch bis dato nicht bekannt und sollte daher Teil zukünftiger Forschung sein.

3.3 Unstrukturierte Testfallerstellung

Spezifikationen sind für die Erstellung von Testfällen essentiell, da sie die Informationen beinhalten, gegen welche Anforderungen das entwickelte System getestet wird. Der Prozess von der Fertigstellung einer Spezifikation hin zur Entwicklung von Testfällen ist jedoch nicht trivial, da es potentiell unendlich viele Testfälle unterschiedlicher Priorität geben kann, deren Strukturierung den Testaufwand bei der Entwicklung von FAS entscheidend beeinflussen und reduzieren kann.

In durchgeführten Industrieprojekten konnte festgestellt werden, dass viele Entwickler Schwierigkeiten bei einer strukturierten Testfallerstellung haben. Neben fehlenden Spezifikationen gab es oftmals keine klaren Kriterien, nach denen Testfälle gezielt erstellt werden konnten. Es werden Abdeckungskriterien benötigt, um festzustellen, welche Teile der Software durch Testfälle abgedeckt werden und für welche Abschnitte noch Testfälle benötigt werden.

Liegen Testfälle vor, müssen diese nach bestimmten Kriterien ausgewertet werden, um beispielsweise eine Priorisierung beim Beheben der Fehler durchzuführen. Hierfür sind geeignete Verfahren zum Messen der Schwere eines Fehlers oder dessen Auswirkung wünschenswert.

Im Bereich sicherheitskritischer Software, wie FAS, ist es auf Grund einer hohen Anzahl von Testfällen wichtig, diese nach verschiedenen Kriterien zu priorisieren. Die Wichtigkeit einer Komponente, die bestimmt, wie viele Testfälle erstellt werden müssen, kann z.B. durch die richtige Auswahl von Abdeckungskriterien abgebildet werden. In Industrieprojekten konnten wir jedoch feststellen, dass derartige Kriterien kaum eingesetzt wurden und dass Entwickler nur begrenztes Wissen über die verschiedenen Techniken besaßen.

Ein weiteres Problem, welches sich durch eine fehlende Struktur bei der Testfallerstellung ergibt, ist, dass die Auswertung der Testfälle keinem eindeutigem Ergebnis zugeordnet werden kann. Es fehlen Vergleichswerte und Vorgaben, nach denen sich die Tester richten könnten.

Lösungsansätze:

Um die Testabdeckung durch vorhandene Testfälle zu prüfen, können geeignete Metriken eingesetzt werden [SRWL11]. Diese könne beispielsweise die Anzahl geplanter, durchgeführter oder fehlgeschlagener Testfälle betrachten. Metriken können z.B. auch das Verhältnis von der Anzahl durchgeführter Testfälle zu den insgesamt spezifizierten Testfällen auswerten. So kann festgestellt werden, wie viele Teile der Software bereits getestet wurden und wie der aktuelle Teststand des Projekts ist.

Schuldt et al. [SSLM13] stellen ein effizientes Testkonzept für FAS vor, welches eine systematische Testfallgenerierung beinhaltet. Dabei verwenden sie eine virtuelle Umgebung zur Testdurchführung. Testfälle werden für die Testumgebung in Form von virtuellen Testszenarien zur Verfügung gestellt und können bereits am Schreibtisch durchgeführt werden. Realfahrten lassen sich nicht gänzlich ersetzen, der Aufwand dafür jedoch reduzieren. Um das Testverfahren effizient zu gestalten, sollen Testfälle systematisch aus den Anforderungen heraus entwickelt werden. Grundsätzlich unterteilen Schuldt et al. das Testkonzept in

vier verschiedene Phasen. Deren Zusammenhang ist in Abbildung 3 dargestellt.

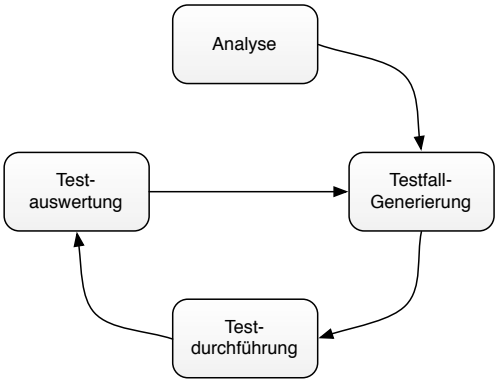

Abbildung 3: Vier Phasen zur Testfallerstellung [SSLM13]

In einer initialen Analysephase wird das FAS auf potentielle Einflussfaktoren untersucht. Dies kann auf Basis der Spezifikation durchgeführt werden (vgl. Kapitel 2). Anschließend beginnt in der zweiten Phase die Testfallgenerierung. Nach der Generierung erfolgt die Testdurchführung. Im Anschluss werden die Testergebnisse automatisiert analysiert, um die Resultate nach bestimmten Kriterien zu vergleichen. Anschließend können neue Testfälle für eine erhöhte Testtiefe generiert werden.

Schuldt et al. stellen die folgenden Methoden vor, um die Phase der Testfallgenerierung systematisch und effizient durchzuführen [SSLM13]:

1. Äquivalenzklassenbildung Der Einflussbereich jeder ermittelten Einflussgröße wird in disjunkte Partitionen aufgeteilt. Diese werden als Äquivalenzklassen (ÄK) bezeichnet und fassen Testbedingungen zusammen, die gleich behandelt werden [BM11]. Aus jeder dieser Äquivalenzklassen wird eine beliebige Anzahl von Parametern gewählt. Es ist angestrebt, eine vollständige Partitionierung durchzuführen, welche auch ungültige Werte beinhaltet. Diese können für Tests auf Robustheit des Testobjekts genutzt werden.

2. Grenzwertbetrachtung Bei der Grenzwertbetrachtung handelt es sich um eine Erweiterung der Äquivalenzklassenbildung. Anstatt zufällige Werte aus einer ÄK zu wählen, werden Randwert-Tupel sowie alle Tupel, für die ein einzelner Wert außerhalb der ÄK liegt, gewählt. Dies kann, abhängig von der Dimension der ÄK, zu einer sehr großen Menge von Testfällen führen. Dennoch wird die Anzahl der Testfälle eingegrenzt.

3. Kombinatorische Testfallgenerierung Die Phase der kombinatorischen Testfallgenerierung soll die Verwendung von redundanten Testfällen verhindern. Hierfür stehen grundsätzlich verschiedene, in der Literatur vorgestellte, Kriterien zur Testfallauswahl zur Verfügung [GOA04]. Schuldt et al. plädieren beim Testen eines FAS mindestens für das

pair-wise Abdeckungskriterium, welches fordert, dass jeder Wert eines Parameters paarweise mit den Werten der anderen Parameter getestet wird. Dieses Kriterium gilt als repräsentativ, da in der Literatur festgestellt werden konnte, dass ein Großteil der Fehler auf eine Kombination von zwei Parametern zurückgeführt werden kann [Lig09]. Auch der Einsatz eines stärkeren *t-wise* Abdeckungskriteriums wäre denkbar, erhöht den kombinatorischen Aufwand jedoch.

Zusammenfassend lässt sich festhalten, dass Abdeckungskriterien zur Ermittlung der Testabdeckung nutzen lassen. Eine systematische Testfallerstellung für virtuelle Szenarien ist umsetzbar, und der Testaufwand lässt sich verringern. Gleichzeitig wird sichergestellt, dass ein bestimmter Abdeckungsgrad der verschiedenen Einflussfaktoren erreicht werden kann.

4 Organisatorische Faktoren

Neben den bereits genannten Faktoren, müssen auch organisatorische Aspekte beim Testen von FAS in Betracht gezogen werden.

4.1 Verteilte Entwicklung

Ein strukturelles Problem, dass bei der Entwicklung von FAS gelöst werden muss, ist die Verteilung von Entwicklungsteams. FAS werden meist bei verschiedenen externen Zulieferern oder diversen internen Teams entwickelt. Bartelt et al. [BBH+09] sehen drei Hauptgründe, die für die (globale) Verteilung von Teams sprechen. Zum einen können so *ökonomische* Vorteile erreicht werden, d.h. es können Kosten gespart werden. Weiterhin gibt es *organisatorische* Vorteile. So können global agierende Unternehmen die Entwicklung ihrer eigenen Struktur anpassen. Als dritten Faktor nennen Bartelt et al. *strategische* Vorteile. Beispielsweise können die Entwickler von lokalisierter Software nah bei den lokalen Kunden sein. Die *time to market* kann somit verringert werden.

Grundsätzlich sollen durch den Einsatz von verteilten Entwicklerteams, sowohl intern als auch extern, Kosten eingespart werden. Es stellte sich jedoch in der Praxis oftmals heraus, dass Kosten nicht gespart, sondern teilweise erhöht wurden [BBH+09]. Bartelt et al. sehen das hauptsächliche Problem in der *mangelhaften Kommunikation* der verschiedenen Teams. Diese ist jedoch gerade in der verteilten Softwareentwicklung entscheidend, da viele Aufgaben voneinander abhängen und koordiniert werden müssen, damit das gewünschte Ergebnis mit entsprechender Qualität erzielt wird. In der klassischen, lokalen Entwicklung existieren meist natürlich gewachsene Kommunikationsmethoden zwischen den Beteiligten. Diese fehlen jedoch oftmals in einem verteilten Umfeld oder sind nicht im ursprünglichen Umfang einsetzbar.

Eine geographische Trennung der Entwickler führt zu erhöhten Schwierigkeiten in der Kommunikation und Abstimmung der Teams. Ist die Verteilung sogar auf verschiedene Länder bzw. Kontinente ausgeweitet, so müssen auch weitere Faktoren, wie z.B. die

Zeitverschiebung oder sprachliche Barrieren, überwunden werden. Diese Probleme gelten nicht nur für die Softwareentwicklung, sondern treten auch bei durchgeführten Testprozessen auf.

Ein komponentenbasierter Entwicklungsansatz, mit verteilt agierenden Teams, bietet Vorteile bei der Softwareentwicklung, wie beispielsweise die vereinfachte Wiederverwendung von Komponenten oder der effizienten Arbeitsaufteilung auf mehrere Parteien [Ros97], jedoch müssen vor allem beim Testen solcher Systeme einige erschwerende Faktoren beachtet werden. Es können durch mangelnde Abstimmungen und fehlende Kommunikation zwischen den Teams und den Verantwortlichen unterschiedliche Qualitätsgrade bei der Testplanung, Testdurchführung und Testabdeckung der Komponenten entstehen. Die Integration der Systeme findet oftmals erst beim Auftraggeber selbst statt. Liegen die einzelnen Komponenten für die Integration vor, fehlt die Information über deren Programmstruktur meist [Ros97, HLS99]. Dies verhindert den Einsatz von Whitebox-Testverfahren, da diese auf der internen Struktur der Codes arbeiten [BM11].

Lösungsansatz: Es müssen demnach neue Methoden gefunden werden, um eine ausreichende Kommunikation zwischen den beteiligten Entwicklern sicherzustellen. Eine Lösung, die von Bartelt et al. vorgeschlagen wird, ist der Einsatz von *virtuellen Projekten*, welche verschiedene *Views* für die verschiedenen Beteiligten zur Verfügung stellen. Views können über Artefakten, Prozessen und Organisationsmodellen gebildet werden. Diese Views besitzen die Eigenschaften, dass sie *standortunabhängig*, *aufgabengesteuert* und *rollenspezifisch* sind. So können Entwickler unabhängig von ihrem Standort Informationen abrufen. Sie sollten stets nur für ihre Aufgabe relevante Elemente präsentiert bekommen, und die aufgeführten Artefakte sollten stets der entsprechenden Rolle eines Entwicklers angepasst sein. Eine große Herausforderung besteht in der Definition der verfügbaren Views. Um Prozesse zu vereinheitlichen, sollten die Schnittstellen über die verschiedenen Teams hinweg klar definiert werden. Als Ziel sollte eine Form von einheitlichem Entwicklungsprozess vorliegen. Die Subprozesse sollten harmonisiert werden, um einen Überblick über das Gesamtprojekt zu erhalten.

Die Projektorganisation muss klar strukturiert sein. Es sollte explizite Ansprechpartner geben. Die Kommunikationswege sollten fest definiert sein. Weiterhin müssen die verschiedenen Artefakte der unterschiedlichen Entwickler verwaltet werden. Es muss geklärt sein, wem welches Artefakt gehört und ob diese konsistent zueinander sind. Sie sollten ebenso auf Redundanzen geprüft werden.

Durch die fehlenden Quellcodes der einzelnen Komponenten müssen Blackbox-Methoden für den Test eingesetzt werden. Hierfür eignet sich die Verwendung der Komponentenspezfikationen (vgl. Kapitel 2). Diese können genutzt werden, um Testszenarien zu entwerfen, welche die Anforderungen des FAS abdecken (vgl. Kapitel 3.3).

4.2 Heterogene Entwicklungsumgebungen

Wegen der verschiedenen Entwicklerteams werden oftmals auch unterschiedliche Entwicklungsumgebungen verwendet. Vor allem im Bereich von FAS müssen oftmals ver-

schiedene Komponenten ineinandergreifen, um eine übergreifende Funktion zu realisieren. Beispielsweise müssen hardwarenahe Strukturen wie CAN- oder FlexRay-Busse die Kommunikation der verschiedenen Steuergeräte und Rechner übernehmen. Es müssen Sensoren und Aktuatoren angesprochen und ausgewertet werden. Teilweise müssen Berechnungen über verschiedene Methoden redundant durchgeführt werden, um ein eindeutiges Ergebnis zu erhalten. Im Bereich des autonomen Fahrens müssen auch Längs-und Querregelung durch Software angesprochen werden. Ausgaben erfolgen entweder an die Aktuatoren des Fahrzeugs oder über ein HMI an den Nutzer.

In der Praxis werden diese verschiedenen Komponenten durch unterschiedliche Programmiersprachen, Frameworks und Werkzeuge implementiert. Gleichzeitig müssen diese jedoch über gemeinsame Schnittstellen verfügen, über die die Kommunikation und der Datenaustausch stattfinden kann. Beispiele aus der Automobilindustrie sind ADTF/C/C++ und Java-basierte Software, Werkzeuge wie beispielsweise CANoe, Versionierungsprogramme, wie SVN oder git sowie Integrationssoftware, wie z.B. Jenkins.

Lösungsansatz: Heterogene Entwicklungsumgebungen lassen sich meist nicht vermeiden. Daher ist es notwendig, die Schnittstellen zwischen den einzelnen Entwicklern klar zu definieren. Im Rahmen von FAS ist es von Vorteil, ein gemeinsames Übertragungsmedium zum Transport der Daten zu verwenden. Für die Automobilindustrie bieten sich hier etablierte Kommunikationsstrukturen, wie z.B. DDS (Data Distribution Service) an. So kann sichergestellt werden, dass alle beteiligten Softwarekomponenten Daten in ähnlichen Strukturen verwenden. Eine Verknüpfung der Softwarearchitektur mit der Hardwarearchitektur sollte möglichst früh geschehen, um mögliche Engpässe oder Kompatibilitätsschwierigkeiten zu erkennen.

Zwischen den Entwicklern sollten Kommunikationsstrukturen etabliert werden, die es erlauben, dass diese intern Schnittstellen und potentielle Probleme frühzeitig kommunizieren können. Grundsätzlich sollten möglichst standardisierte Strukturen bei den Kommunikationsschnittstellen der Komponenten verwendet werden. So kann sichergestellt werden, dass diese eindeutig definiert sind und auch für zukünftige Weiterentwicklungen leicht wiederverwendet werden können.

5 Fazit und Ausblick

Die vorgestellten Herausforderungen treten in ihrer Gesamtheit bei der Entwicklung und dem Testen von FAS auf und müssen von den Herstellern überwunden werden, um sichere Systeme zu entwickeln, die serienreife erreichen können. Diese Herausforderungen gelten allgemein für große, komplexe verteilt entwickelte Software und für die Entwicklung von FAS im Besonderen, da diese Systeme sicherheitskritisch und streng reglementiert sind. Für viele der vorgestellten Herausforderungen konnten wir Methoden vorstellen, die für das Testen moderner FAS genutzt werden können.

Eine besondere Herausforderung stellen jedoch komplexere FAS dar, die autonomes Fahren ermöglichen sollen. Diese zukunftsträchtigen Systeme erfordern spezielle Maßnah-

men, die neu aufgetretene Probleme lösen.

Auch wenn derartige Tests für ein einzelnes System eventuell verwendbar erscheinen, so wird es spätestens bei erforderlichen Regressionstests nach Änderungen am System zu aufwendig, um in der Praxis umgesetzt zu werden.

Langfristig werden FAS in verschiedenen Varianten in der Serienproduktion für Kunden verfügbar sein, ähnlich zu bereits bestehenden Komfortfunktionen. Dies führt langfristig auch zu einem Anstieg potentieller Tests, da die Variabilität dieser Systeme in Betracht gezogen werden muss. Die bisher eingesetzten Testmethoden müssen für den Einsatz auf variablen Systemen angepasst werden. Es können beispielsweise delta-orientierte, regressionstestbasierte Testmethoden, die für Softwareproduktlinien entworfen wurden, eingesetzt werden [LLSG12, DSLL13]. Die entsprechenden Methoden müssten im Rahmen von FAS anhand konkreter Fallstudien evaluiert werden.

Literatur

[Aud] Audi Electronics Venture GmbH. Virtual Test Drive / VTD; Website: `http://www.audi-electronics-venture.de/aev/brand/-de/projekte/virtual_test_drive.html`, Stand: 01.4.2013.

[Aut10] Automotive SPICE Process Assessment Model, Website: `http://www.automotivespice.com/fileadmin/software-download/automotiveSIG_PAM_v25.pdf`, 10. May 2010.

[BBH+09] Christian Bartelt, Manfred Broy, Christoph Hermann, Eric Knauss, Marco Kuhrmann, Andreas Rausch, Bernhard Rumpe und Kurt Schneider. Orchestration of Global Software Engineering Projects. In *Proceedings of the Third International Workshop on Tool Support and Requirements Management in Distributed Projects (REMIDI'09)*, 2009.

[Ber] Berner & Mattner. Messina, Website: `http://www.berner-mattner.com/-en/berner-mattner-home/products/messina/index.html`, Stand 10.4.2013.

[BIS10] Martin Buehler, Karl Iagnemma und Sanjiv Singh. *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic (Springer Tracts in Advanced Robotics)*. Springer, 2010.

[BM11] Graham Barth und Judy McKay. *Praxiswissen Softwaretest - Test Analyst und Technical Test Analyst*. dpunkt.verlag, 2. Auflage, 2011.

[DSLL13] Michael Dukaczewski, Ina Schaefer, Remo Lachmann und Malte Lochau. Requirements-based Delta-oriented SPL Testing. In *In Proceedings of the International Workshop on Product LinE Approaches in Software Engineering (PLEASE)*, 2013.

[GOA04] Mats Grindal, Jeff Offutt und Sten F. Andler. ISE-TR-04-05 - Combination Testing Strategies: A Survey. Bericht, Department of Information and Software Engineering, George Mason University, 2004.

[GS11] Dirk Gunia und Jürgen Schüling. Modellbasiertes Testen des Fahrspur-Assistenten von Ford. In *Industrie Mess-und Prüftechnik*. ATZ, 2011.

[HLS99] Mary Jean Harrold, Donglin Liang und Saurabh Sinha. An Approach To Analyzing and Testing Component-Based Systems. In *ICSE'99 Workshop on Testing distributed Component-Based Systems*, May 1999.

[Hom05] K. Homann. Wirtschaft und gesellschaftliche Akzeptanz: Fahrerassistenzsysteme auf dem Prüfstand. In M. Maurer und C. Stiller, Hrsg., *Fahrerassistenzsysteme mit machineller Wahrnehmung*, Seiten 239–244. Springer, 2005.

[Kra98] Karl-Friedrich Kraiss. Benutzergerechte Automatisierung - Grundlagen und Realisierungskonzepte. *at - Automatisierungstechnik*, 46(10):457–467, 1998.

[Lig09] Peter Liggesmeyer. *Software-Qualität - Testen, Analysieren und Verifizieren von Software*. Spektrum, 2009.

[LLSG12] Sascha Lity, Malte Lochau, Ina Schaefer und Ursula Goltz. Delta-oriented Model-based SPL Regression Testing. In *In Proceedings of the 3rd International Workshop on Product Line Approaches in Software Engineering (PLEASE)*, 2012.

[Mau12] Markus Maurer. Entwurf und Test von Fahrerassistenzsystemen. In *Handbuch Fahrerassistenzsysteme*, Kapitel 5, Seiten 43–54. Vieweg+Teubner Verlag — Springer Fachmedien Wiesbaden GmbH, 2012.

[OMG12] OMG. UML, Version 2.4.1, OMG Specification Superstructure and Infrastructure, `http://www.omg.org/spec/UML/2.4.1/`, May 2012.

[PBG04] Torsten Posch, Klaus Birken und Michael Gerdom. *Basiswissen Softwarearchitektur*. dpunkt.verlag, 2004.

[Reu06] Ralf Reussner. *Handbuch der Software-Architektur*. dpunkt.verlag, 2006.

[Ros97] David S. Rosenblum. Adequate Testing of Component-Based Software. Technical Report 97-34, Department of Information and Computer Science - University of California, August 1997.

[SRWL11] Andreas Spillner, Thomas Roßner, Mario Winter und Tilo Linz. *Praxiswissen Softwaretest - Testmanagement*. dpunkt.verlag, 2011.

[SSLM13] Fabian Schuldt, Falko Saust, Bernd Lichte und Markus Maurer. Effiziente systematische Testgenerierung für Fahrerassistenzsysteme in virtuellen Umgebungen. In *AAET 2013 Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*, 2013.

[SZ10] J. Schäuffele und T. Zurawka. *Automotive Software Engineering*. ATZ-MTZ-Fachbuch. Vieweg, 2010.

[SZ12] Jörg Schäuffele und Thomas Zurawka. *Automotive Software Engineering: Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen*. Springer Vieweg, 5. Auflage, 2012.

[Win01] H. Winner. Patent 101 02 771 A1: Einrichtung zum Bereitstellen von Signalen in einem Kraftfahrzeug, 01. 2001.

[WW02] Matthias Weber und Joachim Weisbrod. Requirements Engineering in Automotive Development - Experiences and Challenges. In *RE*, Seiten 331–340, 2002.

[WW12] Hermann Winner und Alexander Weitzel. Quo Vadis, FAS? In *Handbuch Fahrerassistenzsysteme*. Vieweg+Teubner, 2012.

[ZT10] Dirk Zitterell und Sebastian Thiel. Automatisierter funktionaler Steuergerätetest mit der EXtended Automation Method (EXAM). In *GI Jahrestagung (2)*, Seiten 351–356, 2010.