Einführung eines Produktlinienansatzes in die automotive Softwareentwicklung am Beispiel von Steuergerätesoftware

Christian Hopp**, Holger Rendel*, Bernhard Rumpe*, Fabian Wolf**

*Lehrstuhl Software Engineering, RWTH Aachen
http://www.se-rwth.de
**Elektronik-Entwicklung, Volkswagen AG Braunschweig
http://www.volkswagen.de

Abstract:
Der Anteil an Varianten in der industriellen Software ist in den letzten Jahren stetig gestiegen. Durch den Einsatz von Software-Produktlinien wird versucht die damit verbundene Komplexität zu reduzieren und beherrschbar zu machen. Jedoch ist für die Einführung von Software-Produktlinien ein gewisser Aufwand notwendig, damit diese effizient den Entwicklungsprozess unterstützen können. Es gibt für diesen initialen Schritt keine für alle möglichen Einsatzzwecke beste Lösung. Stattdessen muss individuell eine an die gegebenen Rahmenbedingungen und Herausforderungen angepasste Strategie entwickelt werden. Als eine von vielen Maßnahmen zur Umsetzung dieser Strategie wird im konkreten Kontext die Erweiterung der modellbasierten Softwareentwicklung betrachtet. Es wird in einem Erfahrungsbericht dargestellt, wie diese in der Industrie angewandt werden kann und welche Wirksamkeit die durchgeführten Maßnahmen hatten.

1 Einleitung

Die Software aktueller Steuergeräte im Automobilbereich ist wegen vieler neuer Funktionalitäten, aber auch durch Technikvarianten und Optimierungen deutlich gewachsen und durch einen komplexen Aufbau charakterisiert. Beides ist vor allem durch kurze Produktzykluszeiten mit vielen lokalen Funktionserweiterungen, Modifikationen oder Optimierungen auf Basis von vorherigen Softwaregenerationen bedingt. Insbesondere ist die Wartung von Komponenten variantenreicher Software sehr aufwändig, was in einem hohen Test- und Dokumentationsaufwand resultiert. Eine weitere Schwierigkeit ergibt sich dadurch, dass auch die Software-Architektur nicht durch die sinnvolle Kapselung von Systembestandteilen sondern vor allem durch technische und funktionale Aspekte motiviert ist. Da häufig bei der Erstellung der Software-Architektur die Komplexität und ein gerichteter Signalfluss eine untergeordnete Rolle spielen, ergeben sich hier zusätzliche Herausforderungen in der Variabilität, Dokumentation und im Test.

Eine Optimierung auf allen Schichten des Softwareentwicklungsprozesses kann vor allem durch Nutzung eines Software-Produktlinienansatzes erfolgen. Dieser erlaubt durch

[HRRW12] C. Hopp, H. Rendel, B. Rumpe, F. Wolf
Einführung eines Produktlinienansatzes in die automotive Softwareentwicklung am Beispiel von Steuergerätesoftware
In: Software Engineering 2012: Fachtagung des GI-Fachbereichs Softwaretechnik,
27. Februar - 2. März 2012 in Berlin. pp. 181-192, LNI 198, 2012
www.se-rwth.de/publications

die gezielte Dokumentation und Nutzung von Variabilität eine schnelle und effiziente Entwicklung von Produktfamilien. Dadurch kann auch die Komplexität in zukünftigen Softwaregenerationen verringert werden. Die mit einer Software-Produktlinie verbundenen Auswirkungen auf die Software-Architektur erlauben eine agilere Entwicklung.

In Abbildung 1 wird dargestellt, wie sich die Optimierung des Software-Entwicklungsprozesses zu einer möglichen Umsetzungsstrategie für eine Produktlinie verhält. Die Optimierung des Entwicklungsprozesses hat zum Ziel Variabilität zu erkennen und zu nutzen. Die so erkannten gleichen Teile müssen nur einmal in verschiedenen Produkten entwickelt werden. Diese Optimierung kann die Basis für eine Produktlinie darstellen.

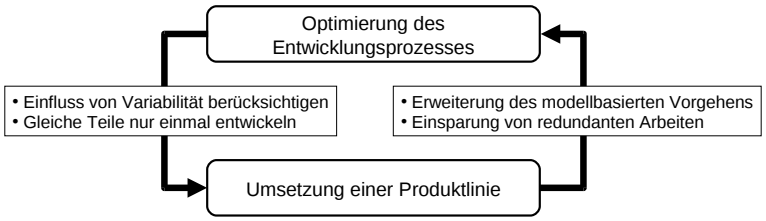

Abbildung 1: Zusammenhang zwischen Optimierung des Entwicklungsprozesses und Umsetzung von Produktlinien

Andersherum hat die Umsetzung einer Produktlinie auch immer Auswirkungen auf den Entwicklungsprozess. Im konkreten Kontext heißt das, dass für eine Produktlinie mehr Modelle genutzt werden müssen um einen Produktlinien-Ansatz umzusetzen. Ein weiteres Ziel der Produktlinie ist die Einsparung redundanter Arbeiten, also Gemeinsamkeiten gezielt zu nutzen. Beides stellt eine Optimierung des Entwicklungsprozesses dar.

Aus dieser Betrachtungsweise heraus soll eine bestehende Entwicklung für eine Steuergerätesoftware optimiert werden. Besonders die modellbasierte Entwicklung soll für den konkreten Einsatzzweck detailliert und angepasst werden. Die modellbasierte Entwicklung ist hier nur eine von mehreren Maßnahmen im Zuge der Umsetzung von Produktlinien und teilt sich im Wesentlichen in die folgenden zwei Schritte auf:

1. Identifikation oder Erstellung von Modellen
2. Nutzung der Modelle im Produktlinienkontext

Diese Schritte wurden in der Elektronik-Abteilung der Lenkungsentwicklung der Volkswagen Business Unit Braunschweig umgesetzt. Durch gezielte Optimierungsmaßnahmen in der modellbasierten Entwicklung konnten signifikanten Verbesserungen erreicht werden. Nicht nur im Hinblick auf Software-Produktlinien kann dieses Vorgehen die Entwicklung in ähnlichen Umgebungen verbessern.

Im folgenden Abschnitt 2 werden die Unterstützungsmöglichkeiten im Hinblick auf Software-Produktlinien im Detail vorgestellt. Abschnitt 3 beschreibt die Umsetzung dieser Möglichkeiten in der Industrie. Die durchgeführten Maßnahmen werden in Abschnitt 4 evaluiert. Verwandte Arbeiten werden in Abschnitt 5 diskutiert.

2 Unterstützungsmöglichkeiten für Produktlinien

Wie Software-Produktlinien theoretisch umgesetzt werden können ist in der Literatur beschrieben [PBL05, CN02, CE00]. In der Praxis gibt es jedoch kein Standard-Vorgehen, wie genau diese Umsetzung am besten erfolgen kann, denn es gibt eine Reihe von Projekt- und Unternehmens-spezifischen Rahmenbedingungen, die berücksichtigt werden müssen.

In dem hier betrachteten Umfeld wird versucht in einer laufenden Entwicklung eine Produktlinie für Lenkungssteuergerätesoftware einzuführen. Es existieren hierbei mehrere Softwarestände mit ähnlichen Funktionen, die jedoch im Wesentlichen als eigene Projekte entwickelt werden. Wiederverwendung findet nicht so strukturiert statt, wie es für Produktlinien vorgesehen ist. Die Einführung erfolgt im laufenden Betrieb. Sämtliche Änderungen dürfen nicht zur Verzögerung von Projektmeilensteinen führen.

Auch wenn bisher nicht nach einem Produktlinien-Ansatz vorgegangen wurde, heißt dies nicht, dass es keine Variabilität in den Produkten gibt. Diese ist nicht immer explizit dokumentiert und wird oft nur in sehr begrenztem Umfang genutzt. Außerdem wird diese Variabilität zu unterschiedlichen Zeiten gebunden beispielsweise zur Kompilierzeit oder erst zur Laufzeit. Für eine Produktlinie müssen diese Variabilitätsdefinitionen konsolidiert und Abhängigkeiten zwischen ihnen analysiert werden.

Die Umsetzung von Produktlinien ist auch immer abhängig von der Wettbewerbspositionierung. Je spezialisierter die zu entwickelnden Produkte für einen Kunden sind, desto weniger potentielle Gemeinsamkeiten ergeben sich zu anderen Produkten. Hier ist es lohnender den Entwicklungsprozess nach Prinzipien eines Produktlinien-Ansatzes zu optimieren als diesen komplett umzusetzen und damit Gefahr zu laufen, dass sich die Kosten für die Umsetzung später nicht amortisieren.

Weitere Rahmenbedingungen ergeben sich durch die benutzten Werkzeuge. Anforderungen an die Software werden mit dem in der Automobilindustrie weit verbreiteten Werkzeug DOORS [IBMa] erfasst. Die in einer Lenkung vorhandenen Funktionen sind in Matlab-Simulink [Sw] implementiert. Zur Verwaltung aller anfallenden Artefakte (sowohl Quellcode als auch Modelle und Dokumente) kommt Synergy [IBMb] zum Einsatz. Die Verwendung von Produktlinienansätzen sind in allen Werkzeugen nicht berücksichtigt und erfordern eine Adaption der Konzepte an das gegebene Entwicklungsumfeld.

Um eine größtmögliche Optimierung des Entwicklungsprozesses zu erreichen, sollten Modelle möglichst früh im Entwicklungsprozess eingesetzt werden. So können von vornherein Fehler durch Redundanzen oder Missverständnisse vermieden werden. Sind diese Modelle identifiziert, können darauf aufbauend Konzepten von Software-Produktlinien implementiert werden indem diese Modelle mit Variabilitätstechniken versehen werden.

2.1 Identifikation oder Erstellung von Modellen

Bekannte Modelle in der Softwareentwicklung sind beispielsweise die in [Rum11, Rum04] dargestellten Modelle der UML oder Matlab-Simulink-Modelle [Sw]. Modelle haben in

der Entwicklung die folgenden wesentlichen Eigenschaften [Sta73]:

- Abbildungsmerkmal: Jedes Modell hat ein Vorbild von dem es abgeleitet ist. Dieses Vorbild kann selbst wieder ein Modell sein.
- Verkürzungsmerkmal: Ziel des Modells ist die Abstraktion von Eigenschaften des Vorbilds. Somit stellt ein Modell gezielt nur einen Teil des Vorbilds dar.
- Pragmatisches Merkmal: Jedes Modell hat einen bestimmten Zweck und enthält die Details des Vorbilds, welche für diesen Zweck notwendig sind.

Bei der heute üblichen Komplexität ist die Übersicht über eine Gesamtfunktionalität ohne sinnvolle modellbasierte Abstraktion sehr schwierig. Deshalb ist es wichtig den Entwicklungsprozess mehr auf Modellnutzung auszurichten als es bisher schon getan wird.

Mit den oben aufgestellten Eigenschaften eines Modells lassen sich noch eine Reihe von Modellen identifizieren, die bisher noch nicht als solche wahrgenommen wurden. Dies liegt vor allem daran, dass Modelle vor allem als graphische Übersichten verstanden werden, jedoch können viele Artefakte ein Modell beschreiben. Zu Modellen zählen zum Beispiel auch Teile von Anforderungsdokumenten, wenn diese die oben aufgezählten Eigenschaften erfüllen.

Als Beispiel kann eine in DOORS abgelegte Kommunikationsmatrix die Architektur eines Systems beschreiben (Abbildungsmerkmal). Die Details, wie Komponenten implementiert sind, sind hier nicht von Belang. Aus diesem Grund werden die Komponenten nur anhand ihrer Schnittstelle dargestellt (Verkürzungsmerkmal). Mit dieser Beschreibung einer Architektur kann diese einfacher erfasst werden und auch beispielsweise mit einem geeigneten Generator Quellcode automatisch erstellt werden (Pragmatisches Merkmal).

Um solche Modelle in DOORS-Dokumenten zu finden, die eher von natürlichsprachlichen Anforderungen dominiert werden, müssen diese auf Modelleigenschaften untersucht werden. Da Modelle vor allem Elemente und deren Eigenschaften darstellen, sollte man sich dabei auf Dokumente konzentrieren, die viele Eigenschaften für ein Anforderungsobjekt definieren. Dabei ist zu untersuchen, ob diese Eigenschaften schon verkürzt und aufs Wesentliche reduziert aufgeführt sind und falls nicht, ob dies sinnvoll möglich ist. Ein weiteres Indiz sind Dokumente, aus denen per Copy und Paste Teile in weitere Artefakte übertragen werden. Hier lassen sich auch oft Optimierungen durchführen indem dieser Schritt automatisiert wird. Dabei müssen die Ausgangs-Dokumente meist noch in Teilen formalisiert werden, so dass Informationen eindeutig in ihnen abgelegt werden können. Dies ist notwendig für eine einfache Automatisierung durch Generierung.

2.2 Nutzung der Modelle im Produktlinienkontext

Sind Modelle identifiziert und in Struktur und Bedeutung verstanden, so können Produktlinien-Techniken für Modelle eingesetzt werden. Ziel ist dabei einen möglichst hohen Anteil an Modellelementen wiederzuverwenden. Im Folgenden wird beschrieben, wie hierbei vorgegangen werden kann.

Optimierung von Komponenten. Ausgehend von Matlab-Simulink-Modellen gibt es innerhalb von Komponenten meist Teile, die in mehreren Varianten der Komponenten enthalten sind (Gemeinsamkeiten) und Teile, die nur für eine spezifische Variante notwendig sind (Unterschiede). Um später die Wiederverwendung so einfach wie möglich zu gestalten, ist es sinnvoll die Komponenten möglichst modular zu gestalten und gemeinsamen Teile als eigene modulare Einheiten zu schaffen. Diese Teile lassen sich dann auch zur Qualitätssicherung gezielter Testen.

Ein Ansatzpunkt für eine solche Optimierung ist die Unterscheidung in Funktion und Schnittstelle einer Komponente. Dies ist in Abbildung 2 dargestellt. Eine Komponente erfüllt oft eine bestimmte Funktion, bei der die Umgebung abstrahiert werden kann. Diese Funktion enthält oft Entscheidungsteile, die basierend auf festgelegten Eingangssignalen zugehörige Ausgangssignale generieren. Häufig werden hier auch Statemachines verwendet, da diese sehr einfach komplexe Entscheidungen modellieren können.

Unter Schnittstelle wird in diesem Zusammenhang die Anpassung der Eingangs- und Ausgangssignale auf die Anforderungen des umgebenden Systems verstanden. So können unterschiedliche Bussysteme oder Signalbandbreiten abstrahiert werden und so konvertiert werden, dass sie der eigentlichen Funktion in verschiedenen Varianten einheitlich zur Verfügung stehen. In diesem Teil dominieren Rechenoperationen, die die notwendigen Konvertierungen oder Glättungen von Signalschwankungen durchführen.

Die Einteilung in Entscheidungsteile und Rechenteile einer Komponente kann ein grobe Orientierung bei der Komponenten-Optimierung geben, jedoch kann die Schnittstelle selber schon Entscheidungsteile enthalten und in der eigentlichen Funktion kommen sehr oft auch Rechenoperationen vor. Im Vordergrund bei dieser Einteilung steht die Wiederverwendung. Diese soll in der Kernfunktion besonders hoch sein. Der Vorteil ist, dass diese nur einmal getestet werden muss, was sehr aufwändig sein kann, wenn sehr komplexe Statemachines in der Kernfunktion vorhanden sind. Für jede Variante muss anschließend nur noch die Schnittstelle selber angepasst und getestet werden.

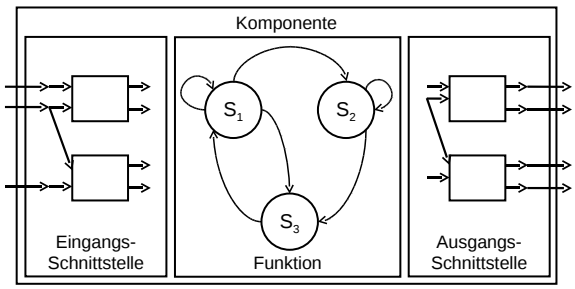

Abbildung 2: Aufteilung in Funktion und Schnittstelle

Eine ähnliche Aufteilung wird auch vom Quasar-Frameworks angestrebt [Sie04]. Die der dort definierte Anwendungsarchitektur entspricht hier dem Funktionskern bzw. den Entscheidungsteilen. Als die technische Architektur, die die Anwendungsarchitektur in einem Gesamtkontext einbettet, kann die Schnittstelle bzw. der Rechenteil gesehen werden.

Generative Softwareentwicklung. Die Generierung von Softwarebestandteilen ermöglicht eine komfortable Entwicklung basierend auf kompakten Darstellungen wie Modellen. Die Umsetzung der Modelle in Artefakte wie Quellcode muss nur einmal in einem entsprechendem Generator implementiert werden. In der Softwareentwicklung sind diese meist schon in Form von Compilern vorhanden, die den Quellcode (als Modell) beispielsweise für einen bestimmten Prozessor kompilieren.

In der generativen Softwareentwicklung besteht nicht nur die Möglichkeit aus einem Modell genau ein weiteres Artefakt zu generieren sondern ein Modell mit verschiedenen Generatoren für mehrere zu erstellende Artefakte zu nutzen. Ein Beispiel hierfür ist, dass aus einer Komponentenbeschreibung sowohl die Implementierung in Form von Quellcode als auch Testmodelle generiert werden können, die dann eine Test-Infrastruktur für diese Komponente darstellen.

In Generatoren kann die Komplexität innerhalb der Softwareentwicklung verringert werden. Dies resultiert daraus, dass mehrere Artefakte in einem Entwicklungsprozess gleichzeitig konsistent angepasst werden können indem nur das Modell modifiziert wird und die darauf basierenden Generatoren ausgeführt werden. Der generative Ansatz verhindert auch, dass nicht vorgesehene Variabilität in Folgeartefakten hinzukommt, da diese Artefakte durch die automatisierte Erstellung sich in einem vorher definierten Rahmen befinden.

3 Industrieller Einsatz

Für den industriellen Einsatz der entwickelten Maßnahmen und Methoden wurden diese exemplarisch im Rahmen der Elektronik-Entwicklung für elektromechanische Lenksysteme der Volkswagen Business Unit Braunschweig umgesetzt. Die hier entwickelten Steuergeräte werden in mehreren Lenksystemen für eine Reihe von Fahrzeugen des Volkswagen Konzerns eingesetzt. Da hier teilweise sehr spezialisierte Software für Kunden entwickelt wird, liegt der Fokus bei der Umsetzung eher auf einer Optimierung des Entwicklungsprozesses als auf eine komplette Umsetzung von Produktlinien. Die Variantenvielfalt hat sich seit der ersten Entwicklung, der APA-Lenkung (Achsparallel-Antrieb) [PH11], stetig erhöht.

3.1 Nutzung von Modellen und Erhöhung der Modellqualität

Im Rahmen des im Abschnitt 2 vorgestellten Vorgehens ließen sich einige DOORS-Dokumente identifizieren, die die Architektur des Gesamtsystems beschreiben. Um hier eine entsprechende Modellqualität zu gewährleisten, wurden Konsistenzchecks implementiert, die Modell-spezifische Eigenschaften prüfen. So können Fehler schon früh erkannt und behoben werden. Die Kompatibilität der Komponenten kann hier schon sichergestellt werden und wird nicht erst bei der Implementierung durch einen Kompilier- oder Laufzeit-Fehler erkannt.

Einfache Konsistenzchecks können vor allem syntaktische Anforderungen prüfen, beispielsweise indem einheitlich ein Punkt statt eines Kommas als Dezimaltrennzeichen genutzt wird. Komplexere Checks stellen die Kompatibilität von Schnittstellen von Komponenten fest. Solche Konsistenzchecks geben die Möglichkeit, Modelle für eine Vielzahl von Einsatzzwecken zu nutzen. Insbesondere kann im Umfeld von sicherheitskritischen Systemen früh überprüft werden ob eine gewählte Architektur den Anforderungen genügt.

Im bestehenden Entwicklungsprozess werden Matlab-Simulink-Modelle schon im Rahmen der Funktionsentwicklung eingesetzt. Diese werden in der Entwicklung von eingebetteter Software sehr oft genutzt. Bei diesen Modellen ist die Anwendung von Produktlinien-Techniken besonders lohnend, da diese schon in den Entwicklungsprozess integriert sind und es verschiedene Modellinstanzen für die einzelnen Varianten der Lenksysteme gibt.

3.2 Modulaufteilung

Für die effiziente Wiederverwendung durch Optimerung von Komponenten, wie in Abschnitt 2.2 beschrieben, ist es notwendig große Module in kleinere weniger komplexe Teile zu zerlegen. Dabei sollte die bisher physisch motivierte Architektur in Richtung einer besser testbaren Architektur geändert werden. Ein weiteres Ziel war hierbei die Schnittstelle des Moduls nach außen nicht zu ändern, da dies Auswirkungen auf weitere Teile der Software gehabt hätte, was eine nicht gewollte Verzögerung von Projektmeilensteinen zur Folge gehabt hätte. Eine weitere gesetzte Restriktion war, dass die bestehende Werkzeugkette zur Erstellung der Gesamtsoftware weiterverwendet werden sollte.

Die Aufgabe des betrachteten Moduls ist die Koordination von Lenkmomentanforderungen aus externen Steuergeräten. Das Modul ist auf Basis von Matlab-Simulink modelliert. Für die Wiederverwendung eignen sich die in dem Modul enthaltenen Statemachines, die das Verhalten bei Lenkmomentanforderungen und Fehlern bestimmen. Als variable Anteile ließen sich Teile identifizieren, die eingehende Signale plausibilisieren oder zusätzliche Ausgangssignale bereitstellen.

Das Modul wurde so umstrukturiert, dass es für den Eingangsteil, die Statemachines und den Ausgangsteil jeweils eigene Module gibt, aus denen mit Hilfe der bestehenden Code-Generierung modular Quellcode erzeugt werden kann. Dazu waren kleine Änderungen nötig, die hauptsächlich auf die durch die Modulaufteilung geänderte Namensstruktur innerhalb des Moduls zurückzuführen sind. Mit ähnlichen Aufwand können auch modular Tests definiert und ausgeführt werden.

Für die Nutzung der modularen Anteile wurde das ursprünglich Modell sowie die Code-Generierung dahin modifiziert, dass die bereits erstellten bzw. generierten Artefakte wiederverwendet werden. Somit ist die Funktion des Gesamtmoduls nach außen hin identisch geblieben und auch Quellcode wird nicht redundant verwendet. Dies sichert, dass im Gesamtmodul der getestete modular generierte Quellcode vorhanden ist.

3.3 Generierung von Test-Infrastruktur

Um Module einer Architektur zu testen werden diese in ein weiteres Matlab-Simulink-Modell eingefügt. Dieses ist abhängig von dem zu testenden Modul. Signale müssen gemäß den Schnittstellen des Moduls mit Daten gefüllt oder von ihnen Ergebniswerte gemessen werden. Die Signale haben einen bestimmten Typ, der von der jeweiligen Schnittstellendeklaration des Moduls abhängt.

Für das Hinzufügen, Löschen und Ändern von Signalen muss das Testmodell an mehreren Stellen angepasst werden. Dies ist teilweise nur umständlich über Kontextmenüs oder tiefer liegende Optionen möglich. Die notwendige manuelle gezielte Benennung von Teilen des Testmodells ist ebenfalls fehleranfällig. Zusätzlich zu den Änderungen müssen auch die enthaltenen Elemente lesbar angeordnet werden um die Verständlichkeit zu gewährleisten.

Letztendlich sind zur Erstellung der Test-Infrastruktur nur die Schnittstellen des zu testenden Moduls als Informationen nötig. Deshalb wurde hier als Beispiel für die generative Softwareentwicklung ein Werkzeug entworfen, das auf Basis dieser Informationen die dazugehörige Test-Infrastruktur erstellt. Dieser Generator ist direkt in Matlab implementiert und kann direkt mit einem Doppelklick aus dem Testmodell heraus aufgerufen werden. Alle Elemente werden durch den Generator auch lesbar angeordnet, sodass alle Testmodelle ein gleiches Aussehen haben.

4 Wirksamkeit der Maßnahmen

Um die Wirkung der vorgeschlagenen Maßnahmen zu evaluieren, wurden die erreichten Optimierungen quantitativ und qualitativ bewertet und durch Stakeholder der Entwicklungsprojekte beurteilt.

Nutzung von Modellen und Erhöhung der Modellqualität. Die identifizierten Modelle, die sich nicht nur auf Matlab-Simulink-Modelle und DOORS-Dokumente erstrecken, hatten unterschiedliche Qualität. Für die Herstellung einer syntaktischen und semantischen Konsistenz konnten Werkzeuge erstellt werden, die hier Ausnahmen identifizieren und teilweise Optimierung vorschlagen konnten. In einigen Modellen wurde Teile für eine bessere Verständlichkeit formalisiert, indem für natürlichsprachliche Eigenschaften eindeutige Werte definiert wurden.

Auch für die Nutzer der Modelle stellten diese eine Optimierung dar. Die Möglichkeit die Informationen in bekannter Weise abzulegen und sich weiteren Implementierungsaufwand zu sparen wurde sehr gut angenommen. So wurden teilweise bestehende Modelle gezielt erweitert um neuen Anforderungen gerecht zu werden und von den Entwicklern auch selber Modelle für eine mögliche auf ihnen aufbauene Automatisierung vorgeschlagen. Dies zeigt, dass die Entwickler für eine modellbasierte Nutzung sensibilisiert wurden.

Bei der Einführung von Modellen hat sich auch gezeigt, dass es nicht immer einfach ist,

eine bisher manuelle Codierung in Modelle mit spezifischen Generator zu überführen. Stellt sich die Modifikation des Quellcodes statt des Modells für den Entwickler einfacher dar, ist dieser weniger geneigt mit einem ihm bisher unbekannten Modell zu arbeiten. Aus diesem Grund ist es wichtig vor der Verwendung von neuen Modellen zu prüfen wie bisher identifizierte Modelle besser genutzt werden können. Modifikationen an bestehenden Modellen werden sehr viel eher von Entwicklern angenommen.

Eine komplette Entwicklung nur auf Modelle umzustellen ist sehr schwierig, da einige spezifische Gegebenheiten eines Systems nicht ohne Weiteres in Modellen ablegen lassen. Hier steht man oft vor der Frage ob ein Modell und damit verbundene Generatoren angepasst werden müssen oder ob das Problem besser durch gezielte Einbettung von Quellcode-Fragmenten gelöst werden kann. Diese Stellen bieten auch immer Potential für Fehler. Mit der Erweiterung des modellbasierten Vorgehens können solche Stellen gut identifizieren werden. Für die Lösung dieser potentiellen Probleme kann jedoch auch mit dem hier Ansatz keine Empfehlung gegeben werden, da sie vom jeweiligen Kontext abhängt.

Modulaufteilung. Da gerade große Module von den Optimierungen profitieren sollen, wurde ein sehr komplexes Modul für die Modulaufteilung verwendet. Nachdem diese durchgeführt wurde, wurde das maximale Einsparungspotential für den Test des Moduls bestimmt, indem exemplarisch die Ausgangsschnittstelle getestet wurde. Diese stellt einen variablen Anteil des Moduls dar, der bei zukünftigen Versionen des Moduls Änderungen unterliegt. Das Einsparungspotential aus Modulaufteilung ist in Tabelle 1 dargestellt.

Tabelle 1: Vergleich vor und nach Modulaufteilung

	durchlaufener Anteil	Testlaufzeit	benötigte Takte	Coverage		
				D1	C1	MC/DC
Original: komplettes Modul	komplettes Modul	100%	100%	85%	96%	92%
Aufgeteiltes Modul	nur Anteil für Test	11%	2%	88%	98%	94%

Durch die gezielte Anregung der Ausgangsverarbeitung des Moduls statt des Gesamtmoduls ließen sich mit weniger Aufwand als bisher eine höhere Testabdeckung schaffen. Das Verhalten der Statemachines musste nicht berücksichtigt werden, da diese separat getestet werden. Die Testlaufzeit wurde mit der Maßnahme um 89% verkürzt und die Anzahl der benötigten Takte (Ausführungsschritte) im Test sogar um 98% reduziert. Im vorliegenden Fall wurde die Testlaufzeit von fast einer Stunde auf einen einstelligen Minutenbereich optimiert, was die Entwicklung neuer Funktionialität beschleunigt.

Die Modulaufteilung ist nicht bei allen möglichen Modulen sinnvoll. Speziell bei großen und komplexen Modulen schafft die vorgeschlagene Herangehensweise einen ersten Anhaltspunkt wie eine mögliche zunächst interne Strukturierung eines Moduls aussehen kann, bei kleineren Modulen ist sehr viel weniger Einsparpotential vorhanden. Zudem zeigt die-

ser Ansatz auch, dass durch die Aufteilung in Funktion und Schnittstelle es einfacher ist, bestimmte Teile zu testen. Abhängig davon wie komplex die Schnittstellen sind und wie gut diese von einer Funktion unterscheidbar ist, kann die Modulaufteilung auch weniger gute Ergebnisse liefern. Insofern ist diese ein grober Anhaltspunkt wie Produktlinien hinsichtlich der Variabilität gestaltet werden können.

Generierung von Test-Infrastruktur. Die Generierung des Testmodells hat zu einer Reduktion des Aufwands für Moduländerungen geführt. Durch den Generator kann die umständliche, zeitaufwändige und fehleranfällige Erstellung der Test-Infrastruktur signifikant verringert werden. Teilweise wurden mehrstündige Tätigkeiten auf wenige Sekunden verkürzt, die der Generator benötigt.

Für die Entwickler bedeutete der Generator eine Vermeidung eines repetetiven einfachen Vorgangs, so dass sich diese auf anspruchsvollere Tätigkeiten konzentrieren konnten. Das einheitliche Erscheinungsbild verhindert, dass man sich in einem noch nicht bekannten Testmodell erst einmal zurecht finden muss. Somit ist der Einstieg für neue Entwickler einfacher geworden.

Im Rahmen der bisherigen Werkzeugkette wurde aus diesem Matlab-Simulink-Modell bisher lediglich Quellcode generiert, der anschließend durch einen Compiler für das jeweilige Steuergerät compiliert wird. Die Ausweitung des generativen Ansatzes auf das bisher von Hand erstellte Testmodell schafft die Möglichkeit einer agileren Vorgehensweise bei der Entwicklung von bestehenden und neuen Modulen. Die Einführung des Generators wurde sehr gut angenommen, da hier massiv Zeit eingespart werden kann.

Dieses Vorgehen funktioniert sehr gut, wenn es schon eine bestehende Trennung in Domain Engineering und Application Engineering gibt. Eine bereits erfolgte Identifikation von wiederverwendbaren Modulen ist eine notwendige Vorraussetzung. Die Erstellung eines solchen Generators ist nur dann sinnvoll, wenn es viele (auch im Rahmen der Erstellung von Prototypen) Änderungen an den Ein- und Ausgängen eines Moduls gibt. Bei Modulen bei denen die Schnittstelle eher fixiert ist, kann der Aufwand für die Erstellung des Generators höher sein als der Nutzen. Die Generierung der Test-Infrastruktur deckt im derzeitigen Stand die notwendige Dokumentation von Variabilität des Moduls selber nicht ab. Dies muss weiterhin separat geschehen.

5 Verwandte Arbeiten

In der Literatur sind mehrere Ansätze für die Einführung von Produktlinien beschrieben. Nach [SPK06] kann das in diesem Papier vorgeschlagene Vorgehen als ein reaktiver Ansatz eingeordnet werden, da Wiederverwendungsmöglichkeiten parallel zum laufenden Entwicklungsprozess genutzt werden. [BFK+99] stellt mit der PuLSE-Methodologie einen umfassenden Ansatz vor. Die Ausweitung der modellbasierten Entwicklung kann als Teil des Ansatzes im Rahmen der Customizing- und Modelling-Komponenten gesehen werden. Wie der Entwicklungsprozess für eine Produktlinie angepasst werden muss, wird auch in [YGM06] diskutiert. Dies wird jedoch nur auf sehr hohem Level getan und

beschränkt sich auf pauschale Angaben wie Restrukturierung des Build-Prozesses wohingegen in Abschnitt 2 die Maßnahmen detaillierter dargestellt werden.

Nach [TH02] stellt die Nutzung bestehender Praktiken und die Einführung neuer Entwicklungsmethoden den Kern einer erfolgreichen Modellierung von Produktlinien dar. Als eine mögliche Umsetzung dieses Gedankens lässt sich die Erweiterung des modellbasierten Vorgehens sehen. Die Abstraktion der Schnittstelle wird auch in [HFT04] am Beispiel von Fahrerassistenzsystemen bei Bosch beschrieben. Hier werden Sensoren variabel gestaltet und entsprechen so dem Vorgehen, wofür die Modulaufteilung die Grundlagen schafft.

Auch in [YFMP08] wird das modellbasierte Vorgehen bei der Umsetzung einer Produktlinie beschrieben. Aufbauend auf der PuLSE-Methodologie wird ebenfalls Matlab-Simulink verwendet um Funktionen zu modellieren und eine Methodologie für die Umsetzung von Produktlinien beschrieben. Für Teile dieses Vorgehens können die in Abschnitt 3 beschriebenen Techniken eingesetzt werden.

Die Anforderungen an Werkzeuge, die für eine Produktlinie notwendig sind, werden in [DSF07] zusammengefasst. Hier wird in einer Evaluation bestehender Werkzeuge festgestellt, dass diese sich gerade auch für bestehende Dokumente in DOORS eignen oder darauf zugeschnitten sind. Wenn Modelle, wie in Abschnitt 2.1 dargestellt, bereits in DOORS identifiziert wurden, kann beispielsweise mit Pure::Variants [pur] die Variabilität in diesen Modellen definiert werden.

6 Zusammenfassung

Die Einführung eines Software-Produktlinien-Ansatzes für die Entwicklung eines automotiven Systems ist von vielen Herausforderungen geprägt. Nicht nur für den Produktlinien-Ansatz sondern auch für die bisherige Entwicklung an sich kann durch eine Erweiterung der modellbasierten Entwicklung eine Optimierung erreicht werden. Dabei ist es wichtig Modellen anhand ihrer wesentlichen Eigenschaften wie dem Abbildungsmerkmal, Verkürzungsmerkmal und dem pragmatischen Merkmal zu identifizieren. Somit kann eine Reihe von Artefakten für die modellbasierte Entwicklung durch gezielte Werkzeugunterstützung genutzt werden.

Im industriellen Einsatz wurde gezeigt, dass vor allem auch bestehende Modelle mehr Möglichkeiten bieten können, als bisher genutzt werden. Dadurch wurde nicht nur eine Optimierung des bisherigen Entwicklungsprozesses erreicht sondern auch die Implementierung einer Produktlinie begünstigt. Auch die Nutzer der möglichen Modelle haben die Einführung von modellbasierter Entwicklung positiv aufgenommen und sind können mögliche weitere Modelle identifizieren.

Literatur

[BFK+99] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen und Jean-Marc DeBaud. PuLSE: A Methodology to Develop Software Product Lines. In *SSR*, Seiten 122–131, 1999.

[CE00] Krzysztof Czarnecki und Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[CN02] Paul Clements und Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[DSF07] Olfa Djebbi, Camille Salinesi und Gauthier Fanmuy. Industry Survey of Product Lines Management Tools: Requirements, Qualities and Open Issues. In *Proceedings of the 15th IEEE International Requirements Engineering Conference (RE)*, Seiten 301–306, 2007.

[HFT04] Andreas Hein, Thomas Fischer und Steffen Thiel. Fahrerassistenzsysteme bei der Robert Bosch GmbH. In *Software-Produktlinien: Methoden, Einführung und Praxis*, Seiten 193–205. dpunkt, 2004.

[IBMa] IBM Rational DOORS website `http://www.ibm.com/software/awdtools/doors/`.

[IBMb] IBM Rational Synergy website `http://www.ibm.com/software/awdtools/synergy/`.

[PBL05] Klaus Pohl, Günter Böckle und Frank van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[PH11] Peter Pfeffer und Manfred Harrer, Hrsg. *Lenkungshandbuch: Lenksysteme, Lenkgefühl, Fahrdynamik von Kraftfahrzeugen*. Vieweg+Teubner Verlag, 2011.

[pur] pure systems. pure::variants.

[Rum04] Bernhard Rumpe. *Agile Modellierung mit UML : Codegenerierung, Testfälle, Refactoring*. Springer, 2004.

[Rum11] Bernhard Rumpe. *Modellierung mit UML*. Springer, 2. Auflage, 2011.

[Sie04] Johannes Siedersleben. *Moderne Software-Architektur: Umsichtig planen, robust bauen mit Quasar*. Dpunkt-Verlag, 2004.

[SPK06] V. Sugumaran, S. Park und K.C. Kang. Software Product Line Engineering. *Communications of the ACM*, 49(12):29–32, 2006.

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, Wien New York, 1973.

[Sw] `http://www.mathworks.com/products/simulink/` Simulink website.

[TH02] Steffen Thiel und Andreas Hein. Modeling and Using Product Line Variability in Automotive Systems. *IEEE Software*, 19:66–72, 2002.

[YFMP08] Kentaro Yoshimura, Thomas Forster, Dirk Muthig und Daniel Pech. Model-Based Design of Product Line Components in the Automotive Domain. In *Proceedings of the 12th International Software Product Line Conference (SPLC)*, Seiten 170–179, 2008.

[YGM06] Kentaro Yoshimura, Dharmalingam Ganesan und Dirk Muthig. Defining a Strategy to Introduce a Software Product Line using Existing Embedded Systems. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software (EMSOFT)*, Seiten 63–72, 2006.