RWTH Aachen

Department of Computer Science
Technical Report

Modellbasierte Entwicklung automobiler Steuerungssysteme in kleinen und mittelständischen Unternehmen

Michael Reke

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2013-02

RWTH Aachen · Department of Computer Science · Mai 2013

The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

`http://aib.informatik.rwth-aachen.de/`

Michael Reke
michael.reke@rwth-aachen.de

Aachener Informatik Bericht AIB-2013-02

Herausgeber: Fachgruppe Informatik
RWTH Aachen University
Ahornstr. 55
52074 Aachen
GERMANY

ISSN 0935-3232

Die Druckfassung dieser Dissertation ist unter der ISBN 978-3-8440-1842-4 erschienen.

Modellbasierte Entwicklung automobiler Steuerungssysteme in kleinen und mittelständischen Unternehmen

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Ingenieur
Michael Reke
aus
Eslohe / Sauerland

Berichter: Professor Dr.-Ing. Stefan Kowalewski
Professor Dr. Thomas Rose

Tag der mündlichen Prüfung: 03. Juli 2012

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.

Zusammenfassung

Kleine und mittelständische Unternehmen (KMU) sind die Innovationsträger der Industrie. Dies gilt insbesondere für eine an Technologie orientierte Branche wie die Automobilindustrie. Für eine stärkere Integration von KMU in die Softwareentwicklung von Großunternehmen müssen jedoch viele Hürden überwunden werden, die aufgrund der großen strukturellen und kulturellen Unterschiede bestehen. So ist die Arbeit in den großen Unternehmen geprägt von stark reglementierten Prozessen und oft langwierigen Entscheidungswegen, die aufgrund der auf viele Mitarbeiter verteilten Entwicklung notwendig sind. Im Gegensatz dazu steht die Flexibilität und vergleichsweise einfache Prozessorganisation bei KMU, die oftmals zu kreativen Lösungen, schnellen Entscheidungen und einer kundenorientierten Arbeitsweise führen.
Der in dieser Arbeit vorgestellte Ansatz für einen Entwicklungsprozess basiert auf Schnittstellen und Methoden der Prozesse, die von Großunternehmen verwendet werden. Die Prozesse selbst werden jedoch für den Einsatz in KMU vereinfacht, kombiniert und anders gewichtet. Das Ziel ist es dabei auf die Elemente zu verzichten, die eingeführt wurden, um die massiv parallele Entwicklung vieler Ingenieure zu organisieren. Die Prozesselemente, die hingegen eingeführt wurden, um eine hohe Softwarequalität zu erreichen, werden beibehalten. Dabei wird vorgestellt, dass insbesondere die modellbasierte Entwicklung geeignet ist, dieses Ziel zu unterstützen. Dafür wird zusätzlich zu der Prozessbeschreibung ein Entwicklungswerkzeug vorgestellt, das eine modellbasierte Softwareentwicklung ermöglicht.
Die in dieser Arbeit diskutierten Prozesse wurden in einem kleinen Unternehmen für Entwicklungsdienstleistungen eingeführt und evaluiert. Zusätzlich wird dargestellt, dass die Prozesse die Ziele und Arbeitpraktiken erfüllen, die von der Automotive SPICE Assessment Methode gefordert werden. Zusätzlich werden Anforderungen für das unterstützende Entwicklungswerkzeug definiert und dessen Softwarearchitektur wird präsentiert. Das zentrale Element ist dabei eine Abstraktionsschicht, die die Hardwarekonfiguration von der Modellierungsumgebung trennt und die für Simulation und Codegenerierung notwendigen Parameter integriert. Zur Evaluation werden unterschiedliche Fallstudien aus dem Bereich elektronischer Steuergeräte für Automobilanwendungen vorgestellt, anhand derer der Einsatz des Werkzeugs, die Entwicklungsmethoden und die Schnittstellen zu externer Softwarewerkzeugen diskutiert werden.
Insgesamt stellt diese Arbeit einen Ansatz vor, wie kleine und mittelständische

Unternehmen einen werthaltigen Beitrag in der Entwicklung automobiler Steuerungssysteme liefern, wenn die eingesetzten Prozesse und Werkzeuge die besonderen Vorteile dieser Unternehmen unterstützen.

Abstract

Small and medium enterprises (SME) are the innovation carriers of the industry and especially for the technology-oriented automotive industry. For a more extensive integration of SME in the software development many obstacles have to be overcome, because of the big structural and cultural differences to large-scale enterprises. In these companies the development is organized by tightly regulated development processes and often long-term decision making processes, which are necessary because of the distributed development by many engineers. In opposite to that is the advantage of SME their big flexibility and the more simple process organisation, which leads to creative solutions, fast decisions and a customer oriented work.
The approach in this thesis introduces a development process, which bases on interfaces and methods from large-scale enterprises processes. But the processes itself are simplified, combined and weighted in a different way. The goal is to skip the elements, which were introduced to organize the massive parallel work of engineers, but to keep those elements, which ensure a high software quality. It is shown that especially model-based development is best suited to support this goal. Additionally a development tool was realized, that fits to the new process approach and enables model-based software development.
The introduced processes were implemented and evaluated in a small development company. It is shown, that the processes are successfully approved by the assessment method Automotive SPICE. For the development tool requirements were defined and the architecture is presented. The main design element is an abstraction layer, which separates the configuration from the modeling-tool and integrates the parameters for simulations and code generation. For evaluation different case studies are presented. The case studies show and discuss the usage of the tool and methods in developing electronic control units for automotive applications. The interface to different external software tools is evaluated and discussed.
In summary the thesis presents an approach, how small and medium enterprises give a beneficial contribution to the development of automotive control-systems, when processes and tools support the individual advantages of these enterprises.

Danksagung

Diese Arbeit entstand auf der Basis der kooperativen und freundschaftlichen Zusammenarbeit des RWTH Lehrstuhls Informatik 11 - Software für eingebettete Systeme mit der VEMAC GmbH & Co. KG.
Ich danke Herrn Prof. Dr.-Ing. Stefan Kowalewski für die fachliche Anleitung sowie der Möglichkeit, an den Lehrstuhlveranstaltungen teilzuhaben und meine Dissertation zu schreiben. Herrn Prof. Dr. Thomas Rose danke ich für die wertvollen Hinweise zur Arbeit und seine Bereitschaft ein Gutachten zu verfassen. Herrn Prof. Dr. Vöcking und Herrn Prof. Dr. Rumpe gilt mein Dank für ihre Teilnahme an der Prüfungskommission.
Weiterhin danke ich Frau Margrit Offergeld und Herrn Dr. Johannes Offergeld für das entgegenbrachte Vertrauen in meine Arbeit bei VEMAC in dessen Rahmen diese Dissertation entstand. Herrn Dr.-Ing. Martin Düsterhöft danke ich für seine persönliche Unterstützung.
Besonders danken möchte ich auch dem VEMAC Team und meinen ehemaligen Kollegen für Ihre Hilfe und Unterstützung. Und mein Dank gilt auch den Mitarbeitern des Lehrstuhls für die freundliche und hilfsbereite Unterstützung während der Arbeit. Besonders danken möchte ich den Mitstreitern aus dem ZAMOMO Projekt Ralf Blum, Andreas Polzer, Jacob Palczynski, Dr. Dominik Schmitz, Dr.-Ing. Frank-Josef Hesseler und Dr.-Ing. Peter Drews, den Studenten, die an dieser Arbeit aktiv mitgewirkt haben, Sven Theissen, Tote Gasomson und David Rixen, sowie den Kollegen, die selbst am besten wissen wie ausgesprochen groß ihr Anteil an dieser Arbeit und unserer gemeinsamen Freundin VeRa ist: Sebastian Grobosch, Thomas Töpfer, Michael Kiausch und Axel Koblenz.
Aber möglich wurde diese Dissertation erst durch meine liebe Frau Judith, die mir die Zeit und die Geduld geschenkt hat, daran zu arbeiten. Ihr und meinen Kindern Luis und Lukas ist dieses Buch gewidmet.

Aachen, im Februar 2013

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Kleine und mittelständische Unternehmen in der Entwicklung 1
1.2 Entwicklung automobiler Steuerungssysteme 2
1.3 Zielsetzung 4
1.4 Beitrag 5
1.5 Gliederung 6

2 Grundlagen und Stand der Technik **9**

2.1 Modellbasierte Entwicklung 9
2.1.1 Definition 9
2.1.2 ZAMOMO 11
2.1.3 Eigenschaften und treibende Faktoren von modellbasierter Entwicklung 13
2.1.4 Modellierungssprachen- und werkzeuge 14
2.1.5 Stand der Praxis 18
2.2 Softwarequalität 20
2.3 Anforderungserfassung 21
2.4 Softwarearchitektur 22
2.5 Architekturbewertung 25
2.5.1 Kosten und Nutzen 25
2.5.2 Kategorisierung von Architekturbewertungstechniken 27
2.5.3 Architekturbewertungsmethoden 29
2.6 Prototypen in der Softwareentwicklung 30
2.6.1 Definition 30
2.6.2 Prototypenentwicklung 32
2.6.3 Rapid Control Prototyping 33
2.7 Prozessmodelle in der Softwareentwicklung 34
2.7.1 V-Modell 34
2.7.2 Agile Methoden 37
2.7.3 Prozessbewertungsmethoden 42
2.7.4 Prozessnotation 47

3 Ansatz für einen Software-Entwicklungsprozess **49**

3.1 Anforderungserfassung . . . 52
3.2 Konzepterstellung . . . 53
3.3 Softwareerstellung . . . 54
3.4 Reviews . . . 56
3.5 Projektmanagement . . . 57
3.6 Wiederverwendung . . . 58

4 Ein Werkzeug für die modellbasierte Entwicklung 59
4.1 Anforderungen . . . 59
4.2 Architektur . . . 63
4.2.1 Übersicht . . . 63
4.2.2 Hardware . . . 65
4.2.3 VeRa Tool-Chain . . . 68
4.2.4 Varianten und Konfiguration . . . 71
4.3 Modellierung und Simulation mit VeRa . . . 75
4.3.1 Übersicht . . . 75
4.3.2 Realisierung in VeRa . . . 76
4.4 Codegenerierung mit VeRa . . . 79
4.4.1 Übersicht . . . 79
4.4.2 Realisierung in VeRa . . . 79
4.5 Ausführungssystem und Kalibrierarbeit mit VeRa . . . 81
4.5.1 Übersicht . . . 82
4.5.2 Realisierung in VeRa . . . 82
4.6 Zuordnungstabelle . . . 86

5 Evaluierung des KMU-orientierten, modellbasierten Entwicklungsprozesses 89
5.1 Einführung . . . 89
5.1.1 Bewertungsmetrik . . . 89
5.2 Anforderungserfassung . . . 92
5.2.1 Beschreibung . . . 92
5.2.2 Erfahrungen . . . 93
5.2.3 Bewertung . . . 95
5.3 Konzepterstellung . . . 98
5.3.1 Beschreibung . . . 98
5.3.2 Erfahrungen . . . 99
5.3.3 Bewertung . . . 99
5.4 Softwareerstellung . . . 102
5.4.1 Beschreibung . . . 102
5.4.2 Erfahrungen . . . 103
5.4.3 Bewertung . . . 104

5.5 Review Prozess . . . 104
5.5.1 Beschreibung . . . 104
5.5.2 Erfahrungen . . . 108
5.5.3 Bewertung . . . 113
5.6 Weitere Prozesse . . . 115

6 Evaluierung der modellbasierten Werkzeugkette 117
6.1 Anwendungsfeld Motorsteuerung . . . 117
6.1.1 Versuchsaufbau . . . 117
6.1.2 Ergebnisse . . . 121
6.2 OSEK Tasks mit unterschiedlicher Zeitbasis . . . 127
6.2.1 Versuchsaufbau . . . 128
6.2.2 Ergebnisse . . . 128
6.3 Luftpfadregelung in einer HIL Umgebung . . . 131
6.3.1 XCP Protokoll . . . 132
6.3.2 Versuchsaufbau . . . 132
6.3.3 Ergebnisse . . . 135
6.4 Ankopplung von Targetlink mit bestehender Architektur . . . 136
6.4.1 Realisierung . . . 136
6.4.2 Versuchsaufbau . . . 140
6.4.3 Ergebnisse . . . 141
6.5 Weitere Arbeiten . . . 144
6.6 Zuordnungstabelle . . . 146

7 Zusammenfassung und Ausblick 149
7.1 Zusammenfassung . . . 149
7.2 Ausblick . . . 153

1 Einleitung

1.1 Kleine und mittelständische Unternehmen in der Entwicklung

Kleine und mittelständische Industrieunternehmen (KMU) sind die Innovationsträger der Wirtschaft, da sie durch ihre Organisations- und Kostenstruktur Vorteile bei der Durchführung von innovativen Entwicklungsprojekten haben. Weiterhin sind aber Innovationen insbesondere für eine an Technologie orientierte Branche wie die Automobilindustrie besonders wichtig. Objektiv zeigt sich dies z.B. daran, dass trotz der Wirtschaftkrise deutsche Unternehmen im Jahr 2009 55,9 Mrd. EUR für Forschung und Entwicklung (FuE) aufwendeten [67]. Davon entfiel mit mehr als 22 Mrd. EUR (fast 40 %) der weitaus größte Teil auf den Sektor Fahrzeugbau, von dem aber nur ca. 6 Mrd. EUR extern vergeben wurden [67]. Weiterhin zeigt die Statistik, dass diese Summe zum überwiegenden Teil (ca. 90 %) von Großunternehmen verausgabt wird [38].
Betrachtet man auf der anderen Seite den Umsatz je Beschäftigten eines Unternehmens als ein Maß für dessen Wirtschaftlichkeit, so liegen kleine und mittlere Unternehmen, die im Bereich Forschung und Entwicklung sowie sonstigen Dienstleistungen arbeiten[1] deutlich vorn [37]. Laut dem statistischen Bundesamt lag der Umsatz je Beschäftigtem im Jahre 2007 für kleine und mittlere Unternehmen im Bereich FuE bei 94.906 bzw. 102.937 EUR im Gegensatz zu lediglich 76.830 EUR bei Großunternehmen [37]. Diese Zahlen belegen das wirtschaftliche Potenzial für die Auftraggeber von FuE, die in der Regel Großunternehmen sind (90 % aller verausgabten Mittel, s.o.), betrachtet man gleichzeitig, dass nur weniger als die Hälfte aller in diesem Bereich Beschäftigten (ca. 45 %, [37]) in kleinen und mittleren Unternehmen beschäftigt sind. Es kann daher geschlussfolgert werden, dass die stärkere Verlagerung von FuE Aktivitäten auf kleine und mittlere Unternehmen aufgrund deren deutlich höheren Wirtschaftlichkeit letzlich einen Kostenvorteil für die Auftraggeber erbringt.
Dennoch müssen für eine stärkere Einbeziehung von kleinen und mittleren Unternehmen in Forschung und Entwicklung auch Hindernisse überwunden werden. Zwischen KMUs und Großunternehmen bestehen große strukturelle und kulturelle Unterschie-

[1] Dies sind die Unternehmen der Klassen 73 und 74 entsprechend der europäischen NACE Klassifikation.

de. So sind die Entwicklung und die daraus entstehenden Innovationen in großen Unternehmen geprägt von stark reglementierten Entwicklungsprozessen und langwierigen Entscheidungswegen. Sie erlauben aber auch durch die Bildung von großen Entwicklungsteams eine Entwicklung zur Serienreife mit sehr hoher Geschwindigkeit und Gründlichkeit, wenn die wichtigsten Entscheidungen einmal gefällt wurden. Dies erfordert natürlich ein hohes Maß an Disziplin, Organisation und Regeln, die sich letztendlich in den gelebten Prozessen wiederfinden. Im Gegensatz dazu besteht die Stärke von kleinen und mittelständischen Unternehmen in ihrer Flexibilität, die sich in schnellen Entscheidungsprozessen und in der starken Kundenorientierung zeigt. Demgegenüber müssen KMUs in der Regel mit beschränkten personellen und finanziellen Ressourcen arbeiten, so dass in der Regel nur die Bearbeitung kleinerer Projekte oder Teilprojekte im Verhältnis zu Großunternehmen sinnvoll ist.
Dennoch erfordert insbesondere Forschungs- und Entwicklungstätigkeit ein hohes Maß an Kreativität und individuellem Einsatz, für das kleine und mittelständische Unternehmensstrukturen das richtige Umfeld bieten. Objektiv lässt sich dies an dem höheren Umsatz je Mitarbeiter ablesen (s.o., [37]). Dies bedeutet, dass Wege gefunden werden müssen, wie kleine und mittelständische Unternehmen stärker in die FuE Aktivitäten großer Unternehmen integriert werden können, um die oben gezeigten Potenziale zu nutzen. In der Abbildung 1.1 sind diese Unterschiede im Spannungsfeld zwischen Kosten, Zeit und Qualität aufgeführt.

1.2 Entwicklung automobiler Steuerungssysteme

Die Entwicklung automobiler Steuerungssysteme ist heute geprägt von großen Unternehmen und großen Entwicklerteams. Neben den oben beschriebenen Kostenvorteilen führt zudem auch das Bestreben der Automobilindustrie Wissen auf mehrere Unternehmen zu verteilen, um Monopolstellungen zu vermeiden, heute zur verstärkten Einbindung auch kleinerer Unternehmen in die Entwicklung. Die früher entworfenen Methoden und Prozesse müssen daher an diese neue Struktur angepasst werden, um die ursprünglichen Ziele Qualität und Effizienz in der Entwicklung beizubehalten und die besonderen Vorteile von kleinen und mittelständischen Unternehmensstrukturen zu fördern.
Für eine allgemeine Prozessbewertung im Sinne des Qualitätsmanagements hat sich mit der Normenreihe ISO 9001 [29] in den letzten Jahren ein Standard etabliert, den auch die Automobilindustrie mit der ISO/TS 16949 [31] übernommen hat. Zusätzlich zu diesen Standards orientiert sich die Automobilindustrie für den Bereich der Softwareentwicklung an allgemeinen Prozessbewertungsmethoden wie SPICE (ISO/IEC 15504, [32]) und CMMI [58]. Die europäische Autombilindustrie hat hierfür mit Automotive SPICE [63] sogar einen eigenen Standard abgeleitet und setzt diesen auch für die Lieferantenbewertung ein. Diese Bewertungsverfahren

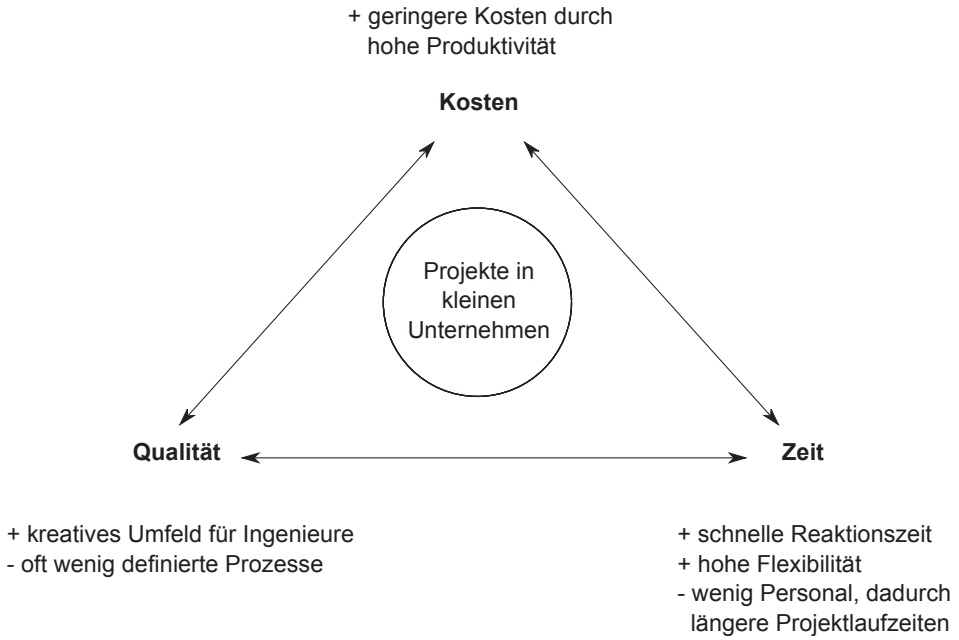

Abbildung 1.1: *Projekte in kleinen und mittlere Entwicklungsunternehmen im Spannungsfeld zwischen Kosten, Qualität und Zeit*

definieren bewusst nur die Prozessziele und die wesentlichen Basispraktiken, nicht aber die Ausgestaltung der Prozesse selbst.
Neben der starken Beachtung von Prozessen hat sich im Laufe der letzten Jahre auch die modellbasierte Softwareentwicklung in der Entwicklung automobiler Steuergeräte mit dem Ziel der Qualitäts- und Effizienzsteigerung etabliert. Die Vorteile liegen auf der Hand: Funktionale Modelle werden in frühen Phasen der Entwicklung in der Simulation eingesetzt, um das System vor dessen eigentlichen Realisierung auf Machbarkeit zu überprüfen und im Detail auszulegen. Können diese einmal erstellten Modelle anschließend für die Realisierung von Prototypen bis hin zu Vorserien- und Serienmustern genutzt werden, so ergibt sich der Vorteil der direkten Übertragung von Ergebnissen aus der einen in die nächste Phase der Entwicklung. Neben dem Kostenvorteil, der durch Vermeidung von Doppelimplementierungen entsteht, führt insbesondere die weitgehend automatisierte Codeerzeugung zu reproduzierbaren Ergebnissen und damit zu einer hohen Qualität in der Software. Ein weiterer Aspekt ist die Übertragbarkeit von Modellierungsergebnissen aus verschiedenen Phasen der Entwicklung. Dies unterstützt die Forderung nach Transparenz und Nachverfolgbarkeit (engl. Traceability) von Ergebnissen.

1.3 Zielsetzung

Es ist das Ziel dieser Arbeit einen Ansatz für ein KMU-orientiertes Software-Engineering aufzuzeigen, das Methoden und Werkzeuge zur Verfügung stellt, welche die Vorteile von kleinen und mittelständischen Unternehmen fördern, gleichzeitig aber eine nachhaltige Qualität sicherstellen.
Die besonderen Vorteile von kleinen und mittelständischen Unternehmen in der Entwicklung sind die hohe Flexibilität, das kreative Umfeld und die starke Kundenorientierung. Dies darf durch Prozesse, die eingeführt werden, um die Qualität zu gewährleisten, nicht verloren gehen. Prozesse sind aber notwendig, um die Qualitäts- und Dokumentationsanforderungen von großen Unternehmen in der Zusammenarbeit mit kleinen und mittelständischen Entwicklungsunternehmen zu erfüllen.
Die Werkzeugketten, die von KMUs in der Entwicklung von Software eingesetzt werden, müssen anpassbar sein, um kompatibel zu unterschiedlichen Kundensystemen zu sein. Die Formate, die die Werkzeuge nutzen, müssen sicherstellen, dass Softwarekomponenten zwischen Unternehmen und den unterschiedlichen Phasen der Entwicklung ausgetauscht werden können. Dabei muss es zusätzlich ein allgemeines Ziel sein, dass die eingesetzten Werkzeuge helfen, Fehler und Unvollständigkeiten in der Spezifikation und der Software zu erkennen und zu korrigieren.

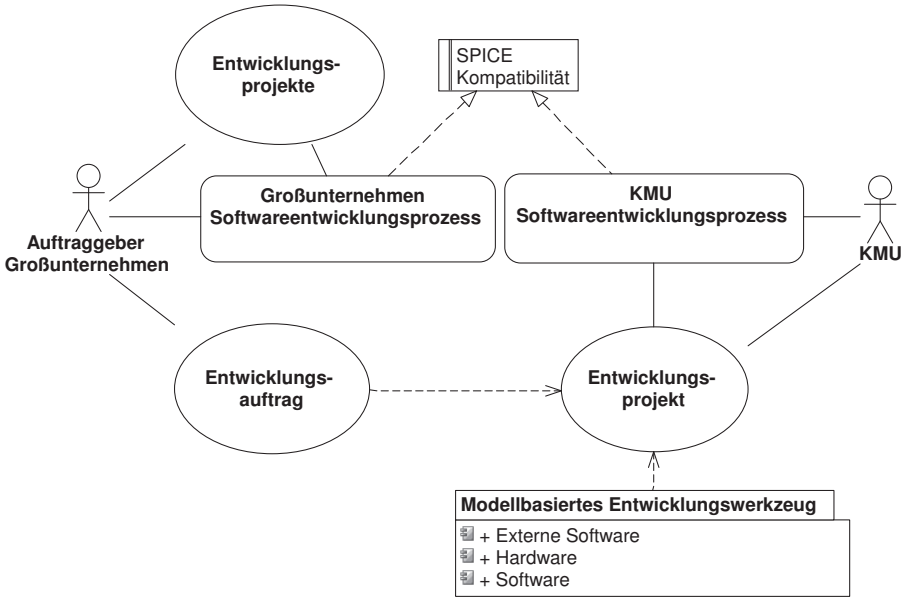

Abbildung 1.2: *Einbindung von kleinen und mittelständischen Unternehmen in die Entwicklung*

1.4 Beitrag

Der Beitrag dieser Arbeit lässt sich anhand eines Use-Case-Diagramms (vgl. Abbildung 1.2) darstellen. Wenn große Unternehmen in der Automobilindustrie, wie Fahrzeughersteller oder Komponentenzulieferer, Entwicklungsprojekte durchführen, kommt ein Softwareentwicklungsprozess zum Einsatz, der an das jeweilige Unternehmen angepasst ist. In aller Regel sind diese Prozesse kompatibel zu den Ergebnissen, die Automotive SPICE definiert. Wenn große Unternehmen in einem anderen Fall Entwicklungsaufträge an kleine und mittelständische Unternehmen vergeben, führt dies zu einem Entwicklungsprojekt, das bei dem jeweiligen Unternehmen durchgeführt wird. Der Beitrag dieser Arbeit ist es nun, den Ansatz für einen Softwareentwicklungsprozess zu beschreiben, der an die Unternehmensstruktur angepasst ist und gleichzeitig die Anforderung der Kompatibilität mit Automotive SPICE erfüllt. Zudem beschreibt die Arbeit die Struktur und Anwendung eines modellbasierten Entwicklungswerkzeuges, das Unternehmen im Sinne der Zielsetzung nutzen können.
Der in dieser Arbeit vorgestellte Ansatz für einen KMU-orientierten Softwareentwicklungsprozess basiert darauf, klassische Methoden und Prozesse für den Einsatz in

kleinen und mittelständischen Unternehmen anzupassen und zu vereinfachen. Durch die Kompatibilität mit den Zielen von Automotive SPICE ergeben sich Prozessschnittstellen, die genutzt werden können, um KMUs einfacher in die Entwicklung von großen Unternehmen zu integrieren, um so die Vorteile beider Unternehmensstrukturen zu nutzen. In dem vorgestellten Prozessansatz wird ein verstärkter Fokus auf den Einsatz von Review- und Bewertungsverfahren gerichtet, die geeignet sind, die Qualität auch bei einem beschränkten Resourceneinsatz sicherzustellen. Gleichzeitig fördert die Einbindung des Auftraggebers in diese Prozesse den Aufbau eines gemeinsamen Problemverständnisses, so dass der Vorteil der starken Kundenorientierung gefördert wird. Durch die vereinfachten Prozesse bleiben zudem die Flexibilität und das kreative Umfeld von kleinen Unternehmen erhalten.
Ebenso wie bei dem vorgestellten Prozessansatz steht auch bei dem in dieser Arbeit beschriebenen Entwicklungswerkzeug die Unterstützung von Schnittstellen für die Einbettung in Entwicklungsprojekte großer Unternehmen im Vordergrund. Dazu gehört zum einen die Nutzung von Softwarestandards und -formaten aber auch die möglichst umfassende Unabhängigkeit von externen Softwarekomponenten und Hardwareteilen. Vor allem die intensive Nutzung von modellbasierter Softwareentwicklung bietet die Möglichkeit im Sinne der Zielsetzung Softwarekomponenten auszutauschen, indem dafür Modelle genutzt werden. Das vorgestellte Werkzeug ermöglicht hierbei eine klare Strukturierung der Modellteile, so dass die Kompatibilität zwischen Unternehmen und Entwicklungsphasen gewährleistet ist. Zudem bietet sich mit der Simulation der ausführbaren Modelle, die Möglichkeit Spezifikationsfehler und -unvollständigkeiten frühzeitig zu finden und zu korrigieren. Diese Vorteile der modellbasierten Entwicklung sind allgemeingültig, kleinen und mittelständischen Unternehmen bieten sie jedoch insbesondere den Vorteil in Kundenprojekten flexibel integriert werden zu können.

1.5 Gliederung

Im ersten Teil der Arbeit werden in Kapitel 2 wichtige Begriffe und Methoden eingeführt, auf denen das diskutierte Software Engineering basiert. In Kapitel 3 wird der Ansatz für einen KMU angepassten Softwareentwicklungsprozess beschrieben. Danach wird in Kapitel 4 die konkrete Umsetzung eines Entwicklungswerkzeugs beschrieben, welches die modellbasierte Entwicklung unterstützt. Der Prozessansatz wird in Kapitel 5 konzeptionell anhand des Automotive SPICE-Bewertungsschemas evaluiert. Mit Hilfe von praktischen Untersuchungen wird der Einsatz des Entwicklungswerkzeug im Sinne des beschriebenen Prozessansatzen in Kapitel 6 dokumentiert. Die Arbeit schließt in Kapitel 7 mit einer Zusammenfassung, einer kritischen Diskussion der erreichten Ergebnisse und einem Ausblick darauf, wie die

hier gezeigten Ansätze für den praktischen Einsatz erweitert und verbessert werden können.

2 Grundlagen und Stand der Technik

Dieses Kapitel gibt eine Einführung in wichtige Begriffe und Methoden, auf denen diese Arbeit aufbaut. In dem in Kapitel 3 vorgestellten KMU angepassten Softwareentwicklungsprozess werden diese Methoden verwendet. Dabei werden sie jedoch zum Teil verändert und vereinfacht eingesetzt. Darüber hinaus wird in diesem Kapitel der aktuelle Stand der Entwicklung in den thematisch beteiligten Bereichen dargestellt.

2.1 Modellbasierte Entwicklung

2.1.1 Definition

Viele Ingenieurwissenschaften verwenden Modelle, um das zu realisierende Objekt nicht nur textuell zu beschreiben sondern auch visuell darzustellen. In der Architektur werden Modelle seit vielen Jahren verwendet um bereits vor Baubeginn einen, natürlich nur maßstäblichen, Eindruck von dem Objekt zu erhalten. Auch im Maschinenbau hat der rechnergestützte modellbasierte Entwurf von 3-dimensionalen Bauelementen mit sogenannten CAD-Werkzeugen die klassische Konstruktion mit 2-dimensionalen Zeichnungen fast vollständig verdrängt. Die Vorteile liegen auf der Hand:

- Ein Modell ermöglicht es dem Entwickler seine Idee dem Auftraggeber oder anderen Personen nicht nur zu beschreiben, sondern auch zu visualisieren. Modelle dienen somit der Kommunikation.

- Ein Modell kann auch nach der Erstellung als Abbild des Originalteils verwendet werden. Es dient somit der Dokumentation.

- Ein Modell kann verwendet werden, um es zu analysieren, oder, um es mit anderen Modellen zu kombinieren und zu verschalten. Es kann virtuell in seine spätere Einsatzumgebung eingebunden werden. Damit dienen Modelle auch der Analyse von Gesamtsystemen.

Mit der gleichen Motivation werden im Softwareentwurf und auch in der Reglerentwicklung Modelle eingesetzt. In [62] wird die modellgestützte Entwicklung insbesondere von eingebetteten Systemen im Automobilbereich wie folgt definiert:

> Bei der modellbasierten Entwicklung werden rechnergestützte Modelle verwendet um die Kommunikation, die Dokumentation, die Analyse und die Synthese als Teil der Systementwicklung zu unterstützen.

Es gibt viele Ansätze die Menge der verschiedenen Modelle einzuteilen. Beispiele für eine Unterteilung sind der Modellierungsgrund, der Abstraktionsgrad oder auch der unterschiedliche Modellierungsgegenstand. Nach Meinung des Autors spiegelt jedoch die folgende Einteilung nach [62] in drei Bereiche den heutigen Stand der Technik am besten wieder:

- *Formale Modelle*, die auch mathematische oder analytische Modelle genannt werden könnten, haben sich als sehr wichtige Werkzeuge für das Lösen ingenieurtechnischer Probleme herausgestellt. Beispiele von formalen Modellen sind die geometrische Beschreibung von Objekten anhand von CAD-Modellen oder auch regelungstechnische Modelle.

- *Konzeptionelle Modelle*, die meist in grafischer Form erstellt werden, erweisen sich als sehr wertvoll für die Kommunikation und Dokumentation von komplexer Software und komplexen Systemen. Ein Beispiele einer Modellierungssprachen dieser Kategorie ist die Unified Modeling Language (UML)[1].

- *Konstruktive Modelle* fokussieren auf das Systemverhalten und bilden eine direkte Basis für das Design selbst, von der Spezifikation bis zur detaillierten Lösung. Der Anstieg der Software- und Elektronikkomplexität der letzten Jahre führte zu einer Evolution der Abstraktionsebene auf der die Entwicklung durchgeführt wird. Beispiele hierfür sind die Evolution der Programmiersprachen von Assemblersprachen über die heutigen Hochsprachen bis hin zu standardisierten Bibliotheken und Middlewares wie z.B. AUTOSAR[2]. Auf der Elektronikseite ist beispielsweise der Übergang in Modellierung vom Gatter über Register bis hin zu heutigen Hardwarebeschreibungssprachen, wie z.B. VHDL. Im Ergebnis findet Entwicklung heute durch das Erstellen von konstruktiven Verhaltensmodellen statt, deren Grundgedanke es ist, die Implementierung automatisch aus dem Modell zu generieren.

All diese Modelle sind jedoch stark miteinander verknüpft und überlappen in ihrer Einteilung. Zudem haben sich in den jeweiligen Ingenieurdisziplinen ihre jeweiligen Werkzeuge etabliert, so dass die größte Herausforderung heute die Schnittstellen zwischen den Modellen und die Architektur des Gesamtmodells sind.
Bezogen auf die Entwicklung von Steuergeräten ist der Stand der Technik heute, dass unterschiedliche Modelle für die Entwicklung des gleichen Steuergerätes erstellt

[1] http://www.uml.org/
[2] http://www.autosar.org/

werden. So werden Modelle für die Softwarearchitektur erstellt, es werden Reglermodelle erstellt, aber die Verknüpfung der Modelle findet in der Regel nicht statt. Das führt dazu, dass insbesondere bei Änderungen alle Modelle gepflegt werden müssen. Dabei ist es schwierig die Konsistenz der Modelle sicherzustellen.

2.1.2 ZAMOMO

Die Verzahnung von modellbasierter Anforderungserfassung, modellbasiertem Regler- oder Funktionsentwurf und modellbasiertem Softwareentwurf war daher auch Inhalt des Forschungsprojekts ZAMOMO[3]. Das Zusammenwirken der beiden Disziplinen läuft in der Praxis nicht reibungslos: Unterschiedliche Begriffswelten und Perspektiven auf den Entwurfsgegenstand verursachen Missverständnisse. Die fehlende Abstimmung im Entwurfsvorgehen verhindert die durchaus mögliche gegenseitige methodische Ergänzung. Im Ergebnis wird erhebliches Potenzial zur effizienteren Erzielung von Qualität, insbesondere Sicherheits- und Zuverlässigkeitseigenschaften von Regelungssoftware verschenkt. Das Ziel des Projekts ZAMOMO war daher, diese Situation durch eine geeignete Integration von softwaretechnischem und regelungstechnischem Entwurf zu verbessern. Insbesondere sollten die in beiden Disziplinen entstandenen modellbasierten Ansätze miteinander verzahnt und ergänzt werden, so dass eine durchgängige Entwurfsmethodik von der Anforderungsanalyse für ganze Produktlinien von softwarebasierten Regelsystemen bis hin zum Feintuning einzelner Regelfunktionen unterstützt wird. Als exemplarische Anwendungsdomäne wurden Motorsteuerungen untersucht. Ein Ergebnis des Projekts ist ein integrierter Entwicklungsprozess, bei dem eine gemeinsame Modellbasis von der Anforderungserfassung bis zur Serie als Basis für die Entwicklung der Steuergerätesoftware verwendet wird (vgl. Abbildung 2.1, [51]). Der zentrale Gedanke, um Modelle von der Anforderungserfassung bis zur Testphase integrieren zu können, ist, die Kopplung zwischen Reglermodell und Streckenmodell von der Anforderungserfassung bis zur Testphase zu vereinheitlichen. Grundlegend dafür sind die realen Sensoren und Aktoren über die die reale Strecke an den Regler angebunden ist. Findet man bereits in frühen Phasen der Entwicklung eine geeignete Abstraktion dieser Schnittstelle, so ist die Übertragbarkeit von Ergebnissen gegeben.

[3]gefördert vom Bundesministerium für Bildung und Forschung (BmBF), Förderkennzeichen: 01ISE04

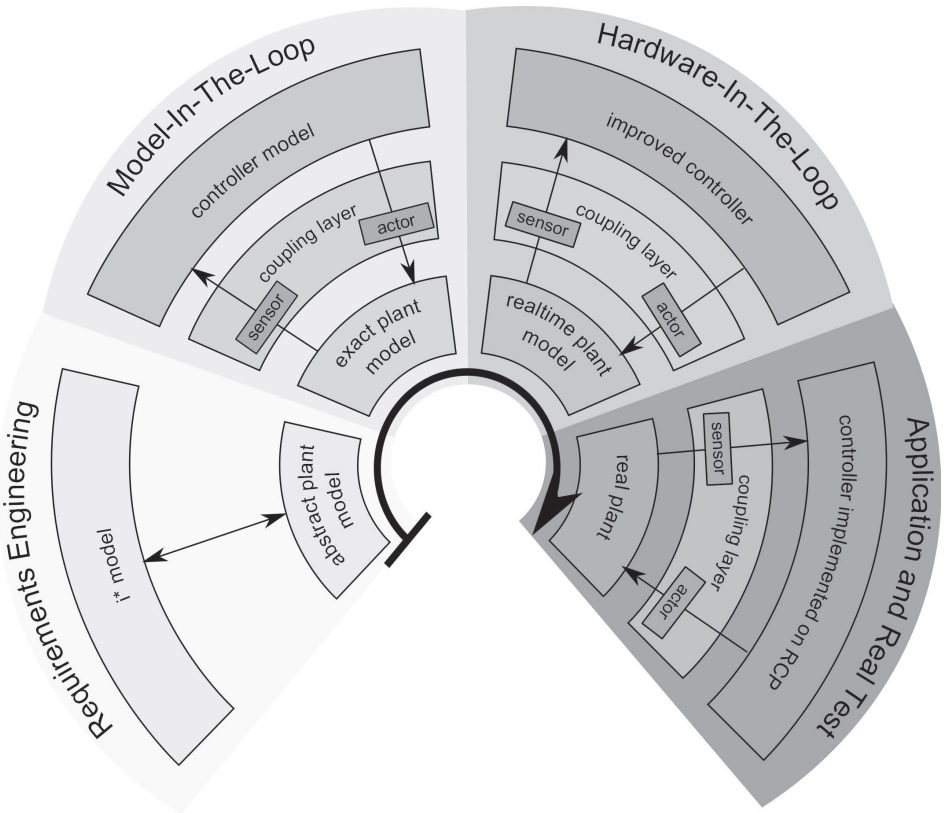

Abbildung 2.1: *ZAMOMO-Entwicklungsprozess für softwareintensive Regelungssysteme [51]*

2.1.3 Eigenschaften und treibende Faktoren von modellbasierter Entwicklung

Modelle zu erstellen bedeutet der Definition entsprechend die Kommunikation, die Dokumentation, die Analyse und die Synthese als Teil der Systementwicklung zu unterstützen. Im Einzelnen bedeutet dies nach [62]:

- *Abstraktion:* Modelle sind ein Abbild der Realität, indem beispielsweise Transferfunktionen als mathematische Beschreibung oder das Problem als Graph mit Knoten und Transitionen dargestellt werden. Derartige abstrakte Beschreibungskonzepte helfen die komplexe Realität zu vereinfachen und dadurch adäquat zu beschreiben, indem überflüssige Aspekte weggelassen und wichtige Aspekte in den Vordergrund gestellt werden.

- *Formalisierung, Parametrisierung und Strukturierung:* Die Modellierung in einer definierten Syntax und Semantik ist die Voraussetzung, um Modelle im Sinne ihrer Beschreibung, ihrer Bedeutung und ihrer Repräsentanz formal zu validieren. Somit ist eine eindeutige Definition der Modellierungssprache die Basis, um deterministische und gültige Modelle zu erstellen. Weiterhin ist eine geeignete Strukturierung des Modells notwendig, um das Modell und damit auch das Problem zu separieren und zu hierarchisieren. Das Konzept der Parametrisierung kann dafür genutzt werden, Modelle und Modellteile wiederzuverwenden und an einen neuen Modellierungsgegenstand zu adaptieren. Die formale Beschreibung liefert dafür die geeigneten Konstrukte.

- *Prädiktion:* Durch verschiedene Modellanalysetechniken ist es möglich das Systemverhalten und die Systemeigenschaften im voraus zu bestimmen.

- *Visualisierung:* Durch grafische Modellierung wird die Struktur des Modellierungsgegenstandes visualisiert. Andere Modelle visualisieren und animieren das Verhalten des modellierten Systems. Dies wird verwendet um die Struktur und das Verhalten eines Sytems besser zu verstehen.

- *Verfeinerung:* Die Verwendung von sukzessiven oder inkrementellen Modellen ermöglicht es die Struktur und die Eigenschaften der Modelle sukzessive zu verfeinern, indem neue Elemente hinzugefügt werden oder Elemente verändert werden.

- *Nachverfolgbarkeit:* Indem verschiedene Versionen oder Iterationen von Modellen untersucht und gespeichert werden, können Weiterentwicklung, Verfeinerung und Optimierung dokumentiert und nachvollzogen werden.

- *Automatisierung:* Rechnergestützte Modelle werden dafür erstellt, Simulationen automatisch durch Skripte ablaufen zu lassen. Durch Code-Generierung

werden konstruktive Modelle automatisch in eine tiefere Abstraktionsebene überführt.

Die steigende Komplexität von Systemen liegt als der wichtigste treibende Faktor für den Einsatz von modellgestützter Entwicklung auf der Hand. Durch Abstraktion, Formalisierung und Synthese werden komplexe Systeme beherrschbar. Zudem besteht ein treibender Faktor in der guten Wiederverwendbarkeit von Modellen und Modellkomponenten. Ein hohes Maß an Wiederverwendung steigert die Qualität und senkt die Kosten, dadurch dass Komponenten in unterschiedlichen Applikationen eingesetzt werden und somit breiter getestet werden. Die hohe Qualität wird ausserdem durch die verbesserten Verifikationsmöglichkeiten anhand von Modellen unterstützt. So kann auch bei sicherheitskritischen Systemen, bei denen eine hohe Testabdeckung gefordert wird, auf Basis von Modellen kosteneffizient getestet werden, da Testfälle durchgehend von der Simulations- bis zur Implementierungsphase verwendet werden können. Zusammenfassend lässt sich festhalten, das die modellbasierte Entwicklung geeignet ist, um kosteneffizient die Qualität zu steigern.

2.1.4 Modellierungssprachen- und werkzeuge

In den unterschiedlichen Ingenieursdisziplinen haben sich unzählige Modellierungssprachen und -werkzeuge etabliert. In der Entwicklung von Software im Allgemeinen nimmt die Unified Modeling Language (UML) jedoch einen Sonderplatz ein, da sie sich zu einem international anerkannten Standard entwickelt hat. In der Entwicklung von funktionalen Regelsystemen in der Automobilindustrie hat sich Matlab/Simulink quasi als Industriestandard etabliert und nimmt daher ebenfalls eine Sonderstellung für das in dieser Arbeit betrachtete Thema ein. Zunächst soll jedoch der i* Formalismus eingeführt werden, der in der Anforderungserfassung und damit in sehr frühen Phasen der Softwareerstellung eingesetzt wird.

i*

i* [68] ist ein agenten- und zielorientiertes, semi-formales Modellierungsrahmenwerk für die Anforderungserfassung [55]. i* unterscheidet zwei Modellierungsebenen: *Strategic Dependency Diagramme* zur Dokumentation strategischer Abhängigkeiten zwischen beteiligten Akteuren und *Strategic Rationale Diagramme* zur Darstellung akteurinterner strategischer Überlegungen.
Auf der höheren Ebene der *Strategic Dependency Diagramme* werden zunächst die verschiedenen Akteure (Stakeholder) in der *Umgebung* der zu entwickelnden Software erfasst. Hierzu können sowohl die zukünftigen Nutzer, deren organisatorisches Umfeld als auch etwaige bestehende Systeme und andere Hardware- oder Software-Komponenten, mit denen die neue Software in Kontakt treten muss, zählen.

i* kennt verschiedene Arten von Akteuren. *Role* bezeichnet eine abstrakte Rolle, beispielsweise die eines Motorenherstellers, die von mehreren Agenten gespielt (*plays*-Kante) werden kann. Unter einem *Agent* wird ein konkreter Akteur verstanden, der so auch in der realen Welt existiert. Darüber hinaus können Akteure durch *is-part-of*-Kanten zu übergeordneten Akteuren kombiniert sowie durch *is-a*-Kanten Spezialisierungen ausgedrückt werden [55].
Die Beziehungen zwischen verschiedenen Akteuren werden in Form von Abhängigkeiten (Dependencies) erfasst. Dabei unterscheidet i* vier Arten von Abhängigkeiten. In einer *Task*-Abhängigkeit (grafische Repräsentation: Hexagon) macht der Auftraggeber klare Vorgaben darüber, wie die delegierte Aufgabe vom Auftragnehmer zu erfüllen ist. Dagegen hat der Auftragnehmer im Falle einer *Goal*-Abhängigkeit (graf.: abgerundetes Rechteck) alle Freiheit, das beschriebene Ziel herbeizuführen; der Auftraggeber ist lediglich am Erreichen des klar definierten Ziels interessiert. Auch im Falle der *Resource*-Abhängigkeit (graf.: Rechteck) gibt es keine Vorgaben für den Auftragnehmer, dieser muss lediglich die gewünschte Resource zur Verfügung stellen. Die *Softgoal*-Abhängigkeit (graf.: wolkenartig) ähnelt zwar der Goal-Abhängigkeit allerdings gibt es im Gegensatz zu dieser keine klaren Kriterien, wann das beschriebene Ziel als erfüllt angesehen werden kann. Stattdessen kann der Auftragnehmer nur Beiträge zur Erfüllung liefern, die letztendliche Bewertung erfolgt durch den Auftraggeber. Diese Abhängigkeit ist somit geeignet, nicht-funktionale Aspekte zu erfassen [55].
Die Ebene der *Strategic Rationale Diagramme* erlaubt die Erfassung von Akteurinterna. Insbesondere kann hier der innere Aufbau eines Akteurs bis hin zu seinen externen Abhängigkeiten formuliert werden. Modellierungstechnisch werden die verschiedenen Typen von Abhängigkeiten (Kanten im Strategic Dependency Diagramm) zu Modellelementen (*Task*, *Goal*, *Resource* und *Softgoal*), die durch neue Kanten verbunden werden können. So lassen sich Tasks und Goals in mehrere Unter-Tasks und/oder -Goals unterteilen (*Decomposition*-Kante). Für ein Goal können außerdem verschiedene Alternativen, die zur Erfüllung des Goals führen, durch *Means-Ends*-Kanten angebunden werden, gleiches gilt für Resourcen. Und schließlich werden *Contribution*-Kanten verwendet, um Beiträge zu Softgoals zu spezifizieren. Der Grad des Beitrags wird dabei durch ein qualitatives Attribut beschrieben, das die Werte *unknown*, *make*, *help*, *some positive*, *break*, *hurt*, *some negative* annehmen kann. Zudem können Softgoals mittels *and*- und *or*-Kanten zu übergeordneten nicht-funktionalen Anforderungen kombiniert werden [55].

UML

UML wurde in der ersten Version wie auch der Unified Process [40] von Rumbaugh, Booch und Jacobsen entwickelt. Nach Einbindung von Unternehmen wie

IBM oder HP wurde UML von der Object Management Group (OMG)[4] im Jahre 1997 schließlich als Standard veröffentlicht. UML wurde in der Version 1.4.2 als ISO/IEC 19501 [33] Norm standardisiert. Aktuell liegt UML in der Version 2.3[5] vor. UML wurde dafür entworfen, Entwicklern ein visuelles Kommunikationsmittel zur Verfügung zu stellen, um Anforderungen und Entwürfe ihres Produktes graphisch zu beschreiben. Das Spektrum der Systeme, bei denen UML zur Beschreibung verwendet wird ist vielfältig und reicht von „klassischen“ Datenbankanwendungen, über Workflow-Anwendungen bis hin zu Echtzeitsystemen. Dies zeigt das einheitliche Ziel, das mit der Anwendung von UML verfolgt wird: „Eine einheitliche Darstellung einer Vielzahl von Elementen von Softwaresystemen mittels einer einheitlichen Notation.“ [70]
UML verfügt über unterschiedliche Diagrammtypen, die unterschiedliche Sichtweisen auf das Modell ermöglichen. So werden z.B. Elemente, die bei einer Aufteilung der Architektur in unterschiedliche Klassen auch bei einer Sequenzdarstellung wiederverwendet. Insgesamt kennt UML in der aktuellen Version sieben Strukturdiagramme (Klassendiagramm, Kompositionsstrukturdiagramm (auch: Montagediagramm), Komponentendiagramm, Verteilungsdiagramm, Objektdiagramm, Paketdiagramm und Profildiagramm) und sieben Verhaltensdiagramme (Aktivitätsdiagramm, Anwendungsfalldiagramm (auch: Use-Case-Diagramm), Interaktionsübersichtsdiagramm, Kommunikationsdiagramm, Sequenzdiagramm, Zeitverlaufsdiagramm und Zustandsdiagramm). Alle Diagrammtypen können genutzt werden, um das gleiche Modell zu beschreiben, aber auch um jeweils einen anderen Aspekt des Modells in den Vordergrund zu stellen. Für weitere Informationen siehe [70] oder auch die Internetseite der OMG.
Um Modelle zu erzeugen werden Werkzeuge eingesetzt, die es ermöglichen Diagramme in den unterschiedichen Diagrammtypen komfortabel zu editieren. Viele dieser Werkzeuge bieten zudem die Möglichkeit ganz im Sinne der modellgetriebenen Entwicklung aus dem Modell automatisch Code zu generieren. Je nach Modellierungstiefe handelt es sich jedoch bei dem erzeugten Code weniger um ausführbaren, funktionalen Code als vielmehr um einen Rahmen aus Modulen, die anschließend mit Funktionen hinterlegt werden müssen. Dieses Vorgehen ist ein Teil der von der OMG spezifizierten Model Driven Architecture (MDA)[6], für die die Unified Modeling laguage (UML) die Basis darstellt.

[4] http://www.omg.org
[5] veröffentlicht im Mai 2010
[6] http://www.omg.org/mda/

Matlab/Simulink

Matlab/Simulink ist heute in der Automobilindustrie die am häufigsten eingesetzte Modellierungs- und Simulationsumgebung. Ausgehend von dem Grundmodul Matlab, einem Softwarepaket für numerische Mathematik, werden viele Erweiterungspakete, die sogenannten Toolboxes angeboten, mit deren Hilfe das Werkzeug an die unterschiedlichsten Anwendungsgebiete angepasst werden kann. Der Name *MATLAB*, leitet sich von *MATrix LABoratory* ab, was schon zeigt, dass dessen besondere Stärke in der Vektor- und Matrizenrechnung liegt [3]. Mit Hilfe der grafischen Bedienoberfläche Simulink lassen sich Signalflusspläne aus verknüpften Funktionsblöcken erstellen. Diese werden zur Modellierung, Simulation und zur Analyse dynamischer Systeme verwendet. Hinzu kommt eine umfangreiche Bibliothek von vorgefertigten Funktionsblöcken für lineare, nichtlineare und diskrete Systeme [3].
Vor allem in der funktionalen Regler- und Steuerungsentwicklung wird Matlab/Simulink von der Automobilindustrie verwendet. Physikalische Modelle der Strecken erlauben in der Simulation eine frühe Validierung von Regler- und Steuerungsfunktionen. Bei sorgfältigem Aufbau der Modelle, können diese anschließend dazu verwendet werden, C-Code automatisch zu erzeugen. Dafür werden zusätzliche Code-Generatoren wie Realtime Workshop[7], Embedded Coder[8] oder Targetlink[9] benötigt. Ein besonderes Merkmal von Matlab/Simulink besteht in den flexiblen Erweiterungs- und Konfigurationsmöglichkeiten, die es erlauben eigene Bibliotheken und Erweiterungen, wie die oben genannten Code-Generatoren hinzuzufügen und kommerziell anzubieten. Kritisiert wird jedoch häufig, dass es sich bei den von Matlab/Simulink verwendeten Dateiformaten nicht um Standard- sondern um proprietäre Formate handelt, deren Aufbau zudem kaum dokumentiert ist. Insbesondere bei Erscheinen von neuen Versionen führt dies häufig zu einem Anpassungsbedarf der Erweiterungsmodule. Bei der Arbeit an dem hier vorgestellten Entwicklungswerkzeug kam dieser Umstand auch zum Tragen.
Der eher funktionsorientierten Modellierung mit Matlab/Simulink steht mit der Model Driven Architecture (MDA) eine Initiative der ObjectManagement Group (OMG) auf der Software-Architektur-orientierten Seite gegenüber. Diese auf den ersten Blick unterschiedlichen Ansätze weisen jedoch auf den zweiten Blick interessante Parallelen auf, die in [14] wie folgt formuliert wurden:

- „Das physikalische Modell hat zum Ziel, Steuerungs- und Regelungsalgorithmen unabhängig von der im Serienfahrzeug letztendlich verfügbaren Steuergeräte-Infrastruktur zu beschreiben. Dieser Ansatz ist vergleichbar mit den Eigenschaften, die durch die Betrachtung von Platform Independent Models (PIM) im Rahmen der MDA verfolgt werden.“

[7] http://www.mathworks.de/products/rtw/
[8] http://www.mathworks.de/products/rtwembedded/
[9] http://www.dspace.de

- „Das Implementierungsmodell berücksichtigt die speziellen Eigenschaften von Ressourcen und realen Schnittstellen (CPU, Busse, Speicher, Kommunikationsinfrastruktur) und ist auf deren Verwendung und optimale Nutzung ausgerichtet. Hier lassen sich Parallelen zu den typischen Eigenschaften eines Platform Specific Models (PSM) erkennen.“

- „Automatische bzw. teil-automatisierte Modelltransformationen sind zentraler Gegenstand des MDA Gedankens, von dem sich die OMG einen entscheidenden Effizienzgewinn verspricht. Im durch Matlab/Simulink getriebenen modellbasierten automotiven Entwicklungsprozess sind Modelltransformationen heute gängige Praxis. Dies betrifft sowohl die Transformation zwischen Modellen untereinander als auch die automatische Generierung von Programmcode für Steuergeräte, die letztendlich auch eine Modelltransformation darstellt.“

- „Die von der OMG getriebene formale Fundierung von Notationen durch Meta-Modelle wird derzeit von der Automobilindustrie als relevantes Tätigkeitsfeld erkannt und in Form von allgemeinen Informations-Meta-Modellen für die Elektronik-Entwicklung umgesetzt.“

Mögliche Vorteile, durch die die Automobilindustrie in Zukunft profitieren könnte, werden in [14] wie folgt beschrieben:

- „im Hinblick auf Funktionalität und semantische Fundierung ausgereifte Entwicklungswerkzeuge (durch eine Erweiterung des Marktpotenzials für die Werkzeughersteller im Vergleich zu rein durch die Automobilindustrie getriebenen Speziallösungen)“

- „Zugang zu einem größeren Kreis von Experten (durch Verwendung einer weit verbreiteten Notation, Methodik und Werkzeuglandschaft)“

- „Erschließung von Synergiepotenzialen über verschiedene Anwendungsdomänen für eingebettete Systeme (z.B. Luft- und Raumfahrt, Verteidigungstechnik, Bahntechnik, Telekommunikation)“

2.1.5 Stand der Praxis

Insbesondere wenn unterschiedliche Disziplinen in der Softwareentwicklung zusammenarbeiten, erfordert dies ein gemeinsames Problem- und Lösungsverständnis. Dieser Bedarf verstärkt sich zudem noch durch den steigenden Vernetzungsgrad der verschiedenen Steuergeräte in modernen Fahrzeugen, aufgrund von verteilten Software-Funktionen. Weiterhin müssen komplexe Randbedingungen durch Sicherheits- und Zuverlässigkeitsanforderungen beachtet werden. Daher lösen in der Software-Entwicklung geeignete modellbasierte Methoden mit Notationen wie Blockdiagrammen und Zustandsautomaten die Software-Spezifikationen in Prosaform

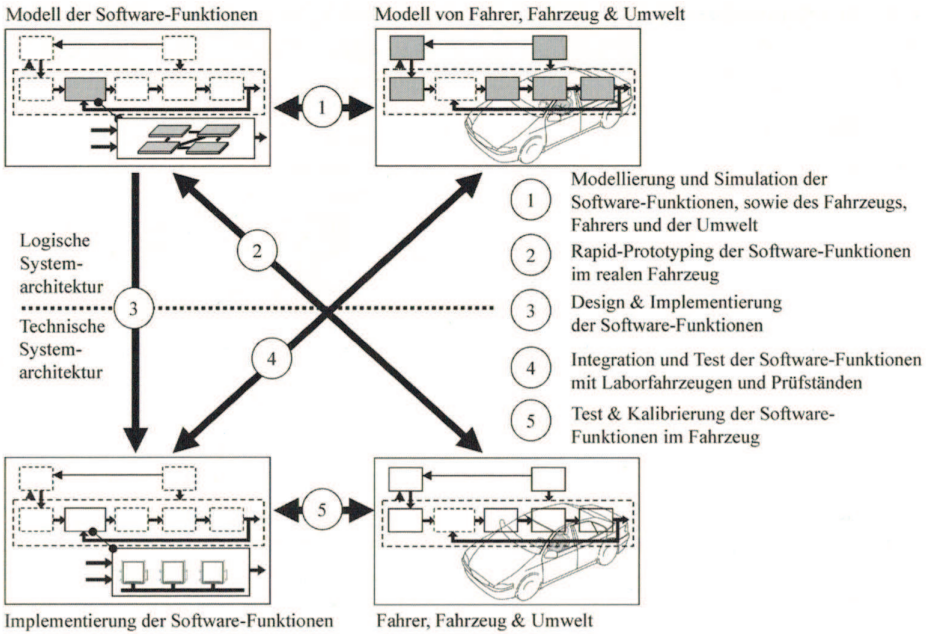

Abbildung 2.2: *Modellbasierte Entwicklung aus [54]*

zunehmend ab, indem ein grafisches Funktionsmodell als Basis für die gemeinsame Entwicklung verwendet wird (siehe auch [54]).
Weiterhin werden Modelle heute ganz selbstverständlich zur Simulation und für den Aufbau von frühen Prototypen („Rapid-Prototyping“, vgl. Abschnitt 2.6) verwendet. Laut [54] hat das „digitale Pflichtenheft“ daher inzwischen eine hohe Verbreitung gefunden.
Im Anschluss an diese Prototypenphase werden die einmal in der Entwicklung spezifizierten Funktionsmodelle durch automatische Codegenerierung auf Softwarekomponenten für Steuergeräte abgebildet. Die so im Steuergerät implementierte Funktion wird dann auf Hardware-In-The-Loop (HIL) Prüfständen (auch „Laborfahrzeug“ genannt) frühzeitig getestet. Dadurch wird insbesondere gegenüber Fahrversuchen eine höhere Flexibilität und einfachere Reproduzierbarkeit der Testfälle erreicht. Während sich dieses Vorgehen für den Softwaretest sehr gut eignet, ersetzt es dennoch die spätere Kalibrierung am realen Fahrzeug im laufenden System nicht.
Insgesamt kann das modellbasierte Vorgehen z.B. wie in Abbildung 2.2 dargestellt werden.

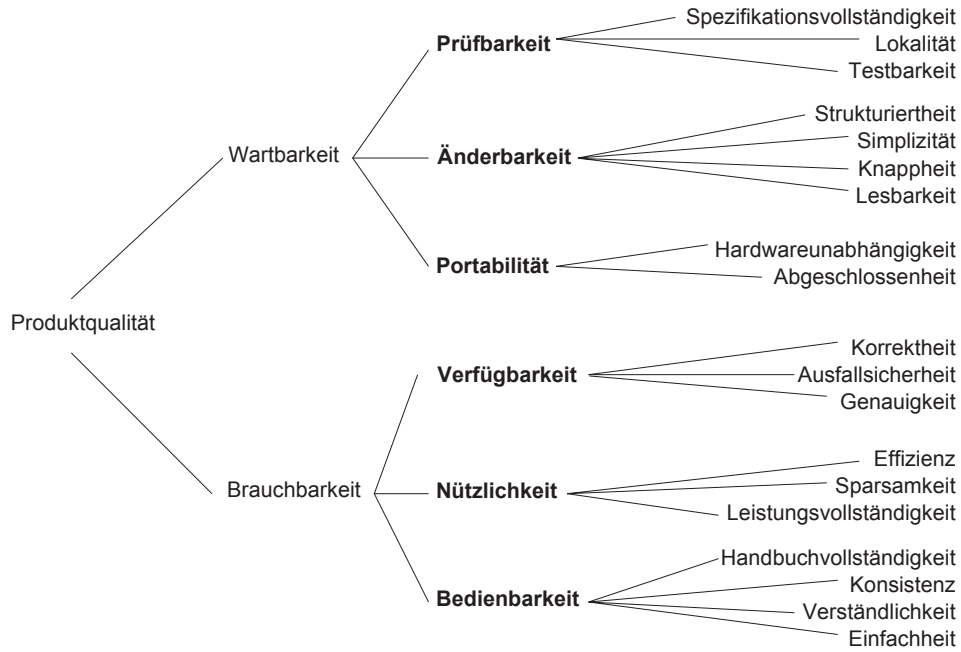

Abbildung 2.3: *Qualitätenbaum nach [41]*

2.2 Softwarequalität

Das Ziel des in dieser Arbeit beschriebenen Software-Engineering-Ansatzes ist es die Softwarequalität zu sichern und zu steigern. Daher wird der Begriff der Softwarequalität hier zunächst eingeführt.
Basierend auf der Definition von Boehm [10] differenziert Lichter [41] die Qualität eines Produkts in Gebrauchs- und Wartungsqualität. Gebrauchsqualität ist die Qualität aus Sicht eines Benutzers der mit der Software arbeitet. Die Qualität aus Sicht einer Person, die an der Software arbeitet, wird als Wartungsqualität bezeichnet. In Übereinstimmung mit [6] kann zwischen sechs Qualitäten unterschieden werden. Von diesen wird die *Zuverlässigkeit* bzw. *Verfügbarkeit*), die *Nützlichkeit* und die *Bedienbarkeit* den Gebrauchsqualitäten zugeordnet, während die *Prüfbarkeit*, die *Änderbarkeit* und die *Portabilität* den Wartungsqualitäten zugeordnet wird.
Die Qualitäten können durch einen *Qualitätenbaum* wie in Abbildung 2.3 dargestellt werden. Seine Struktur verdeutlicht, dass verschiedene konkurrierende Qualitäten angestrebt werden können. Daher müssen in den Anforderungen die Prioritäten so genau wie möglich geklärt sein. Außerdem dient der Qualitätenbaum als Checkliste für die Spezifikation. Dadurch wird sichergestellt, dass alle Qualitätsmerkmale berücksichtigt werden [41].

2.3 Anforderungserfassung

Am Anfang einer jeden Softwareentwicklung müssen die Anforderungen des Kunden erfasst und dokumentiert werden. Anhand der zuvor definierten Softwarequalität lassen sich die Anforderungen in funktionale und Qualitätsanforderungen unterteilen. In manchen Einteilungen (wie z.B. [15]) werden zusätzlich Randbedingungen separat aufgeführt.
Der Nutzen einer Software liegt in ihrer Funktion. Softwarequalitäten wie Zuverlässigkeit oder Wartbarkeit spielen keine Rolle, wenn ein Programm seine Funktion nicht erfüllt. Daher stehen die funktionalen Anforderungen im Vordergrund [41]. Nach [15] beschreibt „eine funktionale Anforderung eine vom System oder einer Systemkomponente bereitzustellende Funktion des betrachteten Systems. Sie beschreibt wie das System reagieren soll. Dies kann z.B. die Abbildung von Eingangsparametern auf Ausgangsparameter durch einen bestimmten Algorithmus sein“.
Eine Qualitätsanforderung hingegen beschreibt eine qualitative Eigenschaft, die das betrachtete System oder einzelne Funktionen des Systems aufweisen müssen. Qualitätsanforderungen, die auch als nicht-funktionale Anforderungen bezeichnet werden, ergänzen somit die funktionalen Anforderungen im Sinne der Softwarequalität (vgl. Kapitel 2.2).
Eine Randbedingung ist nach [15] „eine organisatorische oder technische Anforderung, die die Art und Weise einschränkt, wie das betrachtete System realisiert werden kann“. Randbedingungen ergänzen somit die funktionalen Anforderung und die Qualitätsanforderungen. Beispiele für Randbedingungen sind Kosten, Geschäftsprozesse oder Gesetze.
Anforderungen sollten auch klar von Lösungen getrennt werden. Es sollte daher vermieden werden, in der Anforderungsbeschreibung bereits den Lösungsansatz vorwegzunehmen. Demzufolge wird nach der VDI-Richtlinie VDI 2519 [64] auch in zwei grundlegende Perspektiven einer Spezifikation unterschieden:

- **Anforderungsspezifikation** - „Sie beschreibt was und wofür etwas erstellt werden soll und wird als *Lastenheft* bezeichnet. Das Lastenheft gehört dem Auftraggeber und ist vertragsrelevant.“

- „**Lösungsspezifikation** - „Sie beschreibt wie etwas realisiert werden soll und wird als *Pflichtenheft* bezeichnet. Das Pflichtenheft gehört dem Auftragnehmer und ist die Basis für die durchzuführende Entwicklung.“

Die Kundenanforderungen werden demzufolge im Lastenheft festgehalten. Sie werden in das Pflichtenheft überführt und dort durch Anforderungen ergänzt, die durch das entwickelnde Unternehmen aufgestellt werden.

Für die Erfassung von Anforderungen eignen sich Workshops sehr gut, die mit verschiedenen Interessengruppen durchgeführt werden. Das Ziel eines solchen Workshops ist es, die möglichen Inhalte, Anforderungen und die Produktvision gemeinsam mit allen Interessengruppen gleichzeitig und im Zusammenhang abzustimmen. Der große Vorteil eines Workshops im Unterschied zu individuellen Sammlungstechniken besteht darin, dass die unterschiedlichen Bedürfnisse und die daraus resultierenden Konflikte offen ausgetragen werden und nicht später im Projekt geklärt werden müssen [15]. Methodisch werden einer 2003 durchgeführten Studie [13] zufolge bei der Anforderungserfassung überwiegend szenarien- und Use-Case-basierte Techniken eingesetzt (52% der Befragten).

2.4 Softwarearchitektur

Basierend auf den Anforderungen wird für die zu erstellende Software zunächst eine Architektur festgelegt. Dabei ist der Begriff der Softwarearchitektur nicht eindeutig definiert. Im IEEE-Standard 1471-2000 wird beispielsweise wie folgt definiert:

> „Architecture - The fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution. (IEEE 1471-2000, ISO/IEC 42010 [34])."

Eine aus Sicht des Autors etwas präzisere Definition geben Bass, Clements and Kazman in [6]:

> „The software architecture of a program or computing system is the structure or structures of the system which comprise software elements, the externally visible properties of those elements, and the relationships among them."

Dieser Definition folgend besteht die Softwarearchitektur eben nicht nur aus den Software-Elementen und deren Verknüpfung untereinander, sondern auch aus den Eigenschaften dieser Elemente. Diese können von funktionaler aber im wesentlichen Maß auch von nichtfunktionaler Natur sein. Die Eigenschaften der Software-Elemente wird durch unterschiedliche Personen und Institutionen (engl. Stakeholder) bestimmt, die am Architekturentwurf teilhaben. Neben den Softwarearchitekten sind dies z.B. der Kunde, der Endkunde oder User, das Projektmanagement oder auch die Geschäftsführung des entwickelnden Unternehmens. Dabei verfolgt jede der beteiligten Parteien unterschiedliche und zum Teil gegenläufige Ziele, denen die Architektur Rechnung tragen soll. Es ist die Aufgabe des Architekturentwurfs diese Zielkonflikte aufzulösen, da nur so eine von allen Seiten akzeptierte Software entstehen kann.
Weiterhin wird die Architektur beeinflusst durch die Organisation des entwickelnden

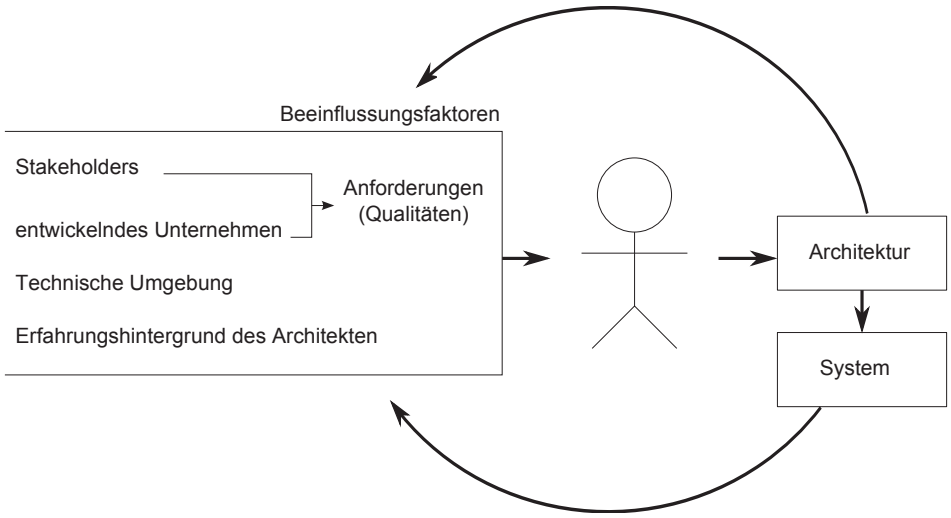

Abbildung 2.4: *Architecture Business Cycle nach [6]*

Unternehmens. Wenn ein Unternehmen beispielsweise einen Großteil seines Umsatzes mit Client-Server Strukturen erwirtschaftet, wird ein technischer Lösungsvorschlag basierend auf einem anderen Architekturmuster nur schwer die Akzeptanz der Geschäftsführung finden. Nach [6] gibt es drei Klassen von Beeinflussung der Architektur durch die Organisation des entwickelnden Unternehmens: Kurzfristiger und, dem entgegen, langfristiger wirtschaftlicher Erfolg sowie die Organisationsstruktur des Unternehmens. Daher muss auch insbesondere in dieser Arbeit diskutiert werden, inwieweit die hier vorgestellte Architektur durch die Struktur des entwickelnden Unternehmens beeinflusst ist.
Weitere Einflussfaktoren sind der Erfahrungshintergrund der Softwarearchitekten. Dieser setzt sich zusammen aus der bisherigen beruflichen Praxis, der Ausbildung und auch der Persönlichkeit des Architekten. Während der eine eher auf bewährte Muster zurückgreift hat der andere Freude am Experimentieren und wählt vielleicht einen neuartigen Ansatz.
Sehr wesentlich für das Verständnis von Architektur ist daher die Betrachtungsweise des Architecture Business Cycles (ABC) nach [6] (vgl. Abbildung 2.4). Dieser Ansatz geht davon aus, dass die Architektur eines Softwaresystems, wie oben beschrieben, durch die Umgebung (technisch, wirtschaftlich und sozial) beeinflusst wird, aber gleichzeitig auch die Softwarearchitektur wiederum die Umgebung beeinflusst. Der Einfluss des Architecture Business Cycles kann in Anlehnung an [6] aus unterschiedlichen Perspektiven betrachtet werden:

1. **Die Architektur beeinflusst die Struktur des entwickelnden Unternehmens.** Die Architektur beschreibt ein System aus unterschiedlichen Soft-

warekomponenten. Die Realisierung jeder einzelnen Komponente erfordert ein spezielles Team z.B. aus Softwareentwicklern, Testspezialisten und Modellierungsfachleuten. Mit jedem neuen Projekt, wird das Unternehmen versuchen diese Struktur möglichst beizubehalten um das Potenzial aus Synergieeffekten zu heben. So entsteht in einem Unternehmen eine Personalzusammensetzung, die zugeschnitten ist auf die Erarbeitung von Projekten, die nach dem gleichen Architekturmuster aufgebaut sind. Dies hat in einem kleinen Unternehmen einen besonders großen Einfluss, da der Personalbestand insgesamt nicht so hoch ist und die Personalplanungszeiträume in der Regel länger sind als Projektlaufzeiten. Ein Spezialist, der für einen bestimmten Teil der Architektur eingestellt wurde, muss daher auch im nächsten Projekt effektiv zum Einsatz gebracht werden, und dies gelingt am Besten, wenn auch in diesem Projekt wieder ähnliche Architekturmuster vorhanden sind.

2. **Die Architektur beeinflusst die Ziele des entwickelnden Unternehmens.** Ein erfolgreich durchgeführtes Projekt aufgrund einer erfolgreichen Softwarearchitektur führt dazu, dass das Unternehmen versuchen wird diesen Erfolg zu wiederholen. Die Akquisition ähnlich gelagerter Projekte wird also zu einem strategischen Ziel für das Unternehmen. Besonders für kleine Unternehmen ergibt sich dadurch die Chance ein bestimmtes Marktsegment für sich zu erschließen und sich gegen oftmals größere Unternehmen durchzusetzen.

3. **Die Architektur beeinflusst die Kundenanforderungen.** Ebenso wie für das entwickelnde Unternehmen, kann es auch für den Kunden das Ziel sein, für ein neues Projekt die Anforderung zu stellen, die Softwarearchitektur an ein erfolgreiches, bereits durchgeführtes Projekt anzulehnen. Auf diese Weise wird der Kunde eine qualitativ hochwertige Software in kurzer Zeit und in wirtschaftlicher Weise erhalten. Ein weiterer Effekt ist, dass Kunden, die ihre Anforderungen nicht vollständig spezifizieren können, sich somit erhoffen, dass die Anforderungen, die an ein früheres Projekt gestellt wurden auch für das neue Projekt implizit erfüllt werden.

4. **Die Architektur beeinflusst die Erfahrung der Softwarearchitekten.** Der technische und der wirtschaftliche Erfolg einer Softwarearchitektur geben dem Softwarearchitekten wichtige Hinweise dafür, ob deren Verwendung für weitere Projekte sinnvoll oder nicht ist. Sie tragen somit zum individuellen Erfahrungsschatz der Architekten, aber auch des entwickelnden Unternehmens bei.

5. **Einige Architekturen beeinflussen sogar die Software-Engineering-Kultur.** So wie beispielsweise web-basierte Anwendungen, den großen Bereich der Unternehmenssoftware revolutioniert haben, hat sich die modellbasierte

Softwareentwicklung im Automobilbereich durchgesetzt. Dieser Gesichtspunkt ist jedoch als deutlich weniger unmittelbar und konkret einzustufen, wie die vorangegangenen Punkte

Diese und weitere Rückkopplungsschleifen bilden den Architecture Business Cycle. Die Erkenntnis des Vorhandenseins dieser Rückkopplungsmechanismen helfen allen am Softwareentwicklungsprozess Beteiligten die besondere Bedeutung einer elaborierten Softwarearchitektur zu verstehen.

Insbesondere aufgrund der zentralen Bedeutung einer guten Softwarearchitektur wurden in der Vergangenheit verschiedene Architekturbewertungsmethoden entwickelt, die sich am Begriff der Softwarequalität orientieren.

2.5 Architekturbewertung

Zusammen mit der vollständigen Spezifikation von funktionalen und nicht-funktionalen Anforderungen bildet die Architekturspezifikation die Grundlage für die Architekturbewertung. Mit ihrer Hilfe wird überprüft, ob der Architekturentwurd die definierten Anforderungen erfüllt. Besteht der Prüfling die Überprüfung, so ist der Nachweis zur potenziellen Erfüllung der Anforderungen erbracht. Wird die Prüfung nicht bestanden, so deutet dies entweder auf einen Mangel im Architekturentwurf oder auf eine inkonsistente Anforderungsspezifikation hin. Im Anschluss an die Überarbeitung von Architektur bzw. Anforderungen wird erneut eine Architekturbewertung vorgenommen.
Nach [7] sollte die Bewertung der Architektur nach den Prinzipien eines Reviews durchgeführt werden. Das Review gehört zu den analytischen Maßnahmen der Software Qualitätssicherung, die das Ziel haben, Fehler und Mängel in den Arbeitsergebnissen zu finden.
Im Folgenden werden nach [7] die Kosten und der Nutzen einer Architekturbewertung gegenübergestellt und es wird anschließend eine allgemeine Kategorisierung von Bewertungstechniken vorgestellt. Hierauf basierend kann beurteilt werden, welchen Wert Bewertungstechniken auch für kleine und mittelständische Unternehmen darstellen können.

2.5.1 Kosten und Nutzen

Unter dem Qualitätsgesichtspunkt gibt es gute Gründe für die Durchführung einer Architekturbewertung. Dennoch ist eine Bewertung immer mit Kosten verbunden, die sich aus Zeit und gebundenen Ressourcen zusammensetzen. Ein wichtiger Punkt ist zu entscheiden, ob der Nutzen die Kosten übersteigt. Die einzelnen Aspekte wurden in [61] zusammengetragen und werden im Folgenden dargestellt.

Kosten

Die Abschätzung der Kosten ist zumeist nur auf Basis von Erfahrungswerten aus bereits durchgeführten Bewertungen möglich. Gutachter des *SEI (Software Engineering Institute)* haben zehn Bewertungen mittels der *Software Architecture Analysis Method (SAAM)* (siehe Abschnitt 2.5.3) für Projekte der Größenordnung 5-100 KLOC (Kilo Lines of Code) durchgeführt und berichteten über Kosten von 14 Personentage. Zu beachten sind auch die hohen Anfangskosten, die bei der Einführung der Prüfungen aufgrund von mangelnder Erfahrung entstehen. Aus [7] ist zu entnehmen, dass viele Unternehmen wie z.B. Siemens, Motorola, AT&T und Lookhead Martin bereits eigene Abteilungen zur Definition und Durchführung von Architekturbewertungen gegründet haben. Mittels dieser Zentralisierung soll eine Maximierung des Ertrages durch die Wiederverwendung auf der Architekturebene erreicht werden.

Nutzen

Dokumentation und Verständnis des Systems:
Ein Nutzen dieser Prüfung ist, dass der Software-Architekt sich im Vorfeld auf diese Prüfung vorbereiten muss. Eine Bewertung erfordert eine Dokumentation der Architektur, bevor sie durchgeführt werden kann. Diese Dokumentation ist von großem Wert, da häufig die Architektur auf oberster Abstraktionsebene nicht von allen Entwicklern verstanden wird oder ihnen nicht einmal bekannt ist. Außerdem kommt es in vielen Fällen zu Missverständnissen zwischen den Entwicklern durch falsche Annahmen bezüglich ihrer Komponenten. Die Vorbereitung und Dokumentation vor einer Bewertung enthüllt viele Ungereimtheiten und hilft das Verständnis des Systems zu erhöhen. Außerdem führt diese Maßnahme zu einem unternehmenseinheitlichen Sprachgebrauch.

Frühzeitige Erkennung von Schwächen einer Architektur:
Je früher in einem Entwicklungsprozess Mängel entdeckt werden, desto kostengünstiger sind diese zu beseitigen.

Klärung und Priorisierung der Anforderungen:
Die Diskussion und Überprüfung, wie gut eine Architektur die Anforderungen erfüllt, kann auch zu einer Diskussion der Anforderungen und deren Priorisierung führen. Oft sind nicht alle geforderten Qualitsmerkmale eines Systems gemeinsam zu realisieren. Architekturbewertungen können solche Konflikte aufdecken und Anregungen für Kompromisse (*Trade-offs*) bieten.

Lerneffekte:
Unternehmen die Architekturbewertungen in ihren Entwicklungsprozess fest integriert haben, berichten von einer Qualitätssteigerung der zu prüfenden Architekturen [7]. Die Entwicklungsabteilungen werden mit der Zeit immer erfahrener und vertrauter im Umgang mit den Prüfungen. Dadurch besteht die Möglichkeit sich immer besser auf diese Prüfungen vorzubereiten. Die Folge sind qualitativ bessere und vor allem besser dokumentierte Architekturen. Das heißt, Architekturbewertungen führen nicht nur zu besseren Architekturen nach der Prüfung, sondern sind auch schon vor der Prüfung qualitativ hochwertiger. Mit der Zeit wird sich in den Entwicklungsabteilungen eine Kultur herausbilden, die ein gutes Architekturdesign unterstützt.

Zusammenfassend lässt sich festhalten, dass Architekturbewertungen unterstützend wirken auf die Erhöhung der Qualität, die Kontrolle der Kosten und die Minimierung des Budget-Risikos. Die Architektur ist der Rahmen für alle technischen Entscheidungen und hat damit einen großen Einfluss auf die Kosten und Qualität des Produkts. Eine Architekturbewertung ist keine Garantie für hohe Qualität und niedrige Kosten, aber sie kann etwaige Risiken in einem Design aufdecken und führt zu einem Qualitätsbewusstsein und Kostenbewusstsein bezüglich der Architektur [7].

2.5.2 Kategorisierung von Architekturbewertungstechniken

In der Praxis existieren unterschiedliche Techniken zur Durchführung einer Architekturbewertung, die jeweils unterschiedlich hohe Kosten verursachen und verschiedene Informationen über eine Architektur offenbaren. Nach [7] lassen sich die Techniken in zwei Kategorien unterteilen: Zum einen in solche, die qualitative Fragen an die Architektur stellen (*Fragetechniken*) und zum anderen in solche, die quantitative Messungen für die Bewertung vorschlagen (*Messtechniken*). Die *Messtechnik* wird genutzt, um einzelne, spezifische Qualitätsmerkmale wie z.B. Performanz zu bewerten. Daher ist es schwierig sie so umfassend anzuwenden wie die *Fragetechniken*, bei denen verschiedene Qualitätsmerkmale untersucht werden. Die beiden Techniken werden unten nach [7] erläutert. Die Zusammenstellung basiert auf [61].

Fragetechniken

Die Gruppe der Fragetechniken besteht aus Szenarien, Checklisten und Fragebögen. Diese Techniken dienen zur Diskussion der Architekur und zur Prüfung gegen ihre Anforderungen. Sie werden auch in dem in dieser Arbeit vorgestellten Ansatz eingesetzt.

Szenarien
Wartbarkeit, Bedienbarkeit, Zuverlässigkeit etc. sind Qualitätsmerkmale anhand dere viele Probleme beschrieben und kommuniziert werden können. Aber die meisten Merkmale sind zu komplex und zu abstrakt, um sie auf einer einfachen Skala abzubilden. Außerdem lassen sie sich nicht isoliert betrachten, sondern erhalten ihre Bedeutung durch den Kontext. So ist ein System z.B. modifizierbar in Abhängigkeit der vorzunehmenden Änderungen. Diese kontext-basierte Ansicht der Bewertung führt zu Szenarien. Diese dienen einer anschaulichen Spezifizierung und Bewertung der Qualitätsmerkmale in einem bestimmten Kontext. Ein Szenario beschreibt dabei einen Ablauf von bestimmten Forderungen an das System, aus dem die Nutzung oder Modifikation des Systems folgt. Beispielsweise testet ein Szenario die Modifizierbarkeit, indem eine Menge von spezifischen Änderungen des Systems angenommen werden. Zur Ermittlung und Organisation der Szenarien sollten verschiedene *Stakeholder* mit einbezogen werden. Dadurch wird vermieden, dass nur „einseitige" Szenarien entwickelt werden. Die Entwickler und Wartungsarbeiter tendieren eher dazu, Szenarien bezüglich der Qualitäten, die für die Entwicklung relevant sind, zu entwerfen. Die Anwender betrachten eher die funktionalen und kommerziellen Aspekte des Systems.

Fragebogen
Ein Fragebogen besteht aus allgemeinen und relativ offenen Fragen, welche generell auf Architekturen angewandt werden können. Die Fragen können sich z.B. an die Art und Weise der Erstellung der Architektur richten.

Checklisten
Eine Checkliste enthält detailiertere Fragen als die oben erläuterten Fragebögen. Die Fragen resultieren meist aus der Erfahrung vieler durchgeführter Bewertungen innerhalb einer Domäne. Dadurch wird bei der Diskussion der spezifischen Qualitätsmerkmale viel mehr in die Tiefe gegangen.

Messtechniken

Wie bereits oben beschrieben, bieten Fragetechniken die Möglichkeit, Fragen an die Architektur zu formulieren. Die Messtechniken dagegen liefern Antworten auf bestehende Fragen. Über Messtechniken kann geprüft werden, wie gut eine Qualität umgesetzt ist. Das heißt, Messtechniken führen zu quantitativen Resultaten. In der Praxis werden diese Techniken meist zur Überprüfung der Performanz und der Modifizierbarkeit herangezogen. Eine bekannte Technik zur Bewertung der Modifizierbarkeit ist das Prinzip der Kopplung und Kohäsion.

2.5.3 Architekturbewertungsmethoden

In der Literatur werden viele Bewertungsmethoden beschrieben (allein nach [16] 18 Methoden). Sie setzen die in Abschnitt 2.5.2 genannten Techniken konkret um und unterscheiden sich in ihren Schwerpunkten und Zielen. Dabei sind die einzelnen Methoden eng miteinander verknüpft, da manche entstanden sind, indem Teilaspekte einer anderen Methode eingeflossen sind und mit neuen Aspekten kombiniert wurden. Andere Methoden sind lediglich Spezialisierungen einer weiteren. Die Auswahl einer Methode in einem konkreten Fall gestaltet sich daher recht schwierig, da eine strukturierte Unterstützung für den Auswahlprozess bisher nicht existiert. Nach [16] ist das ein Grund, warum die Bewertung der Architektur trotz des guten Kosten-/Nutzen-Verhältnisses in der Praxis häufig kein fester Bestandteil des Softwareentwicklungsprozesses ist.
Exemplarisch sollen mit *SAAM* und *ATAM* zwei szenariobasierte Bewertungsmethoden eingeführt werden. In [61] wurden diese Methoden auch mit weiteren verglichen.

- *Software Architecture Analysis Methode (SAAM):*
  Die Methode *SAAM* wurde am *SEI (Software Engineering Institute)* entwickelt. Sie basiert auf der Szenariotechnik und legt besonderen Wert auf die Einbeziehung der verschiedenen Stakeholder. Der Kontext in dem die Qualitätsmerkmale geprüft werden erfordert die Einführung von Szenarien, welche die Bewertung anschaulich und erklärend unterstützen. Eine Vielzahl von Szenarien beschreiben z.B. die Interaktion des Benutzers mit dem System.

- *Architecture Tradeoff Analysis Method (ATAM):*
  Die Methode *ATAM* ist ebenfalls am SEI entwickelt worden und greift auf Analyseverfahren und -techniken der *SAAM* zurück. Die Beziehungen zwischen den Entwurfsentscheidungen und den beeinflussten Qualitätsmerkmalen werden jedoch detaillierter untersucht. Außerdem wurden neue Konzepte wie der *Utility-Baum*, *Tradeoff Points* und *Sensitivity Points* eingeführt [16]. Ein *Sensitivity Point* ist eine Erfassung von Komponenten der Architektur, die bezüglich eines bestimmten Qualitätsmerkmales kritisch einzustufen sind. Ein *Tradeoff Point* ist ein *Sensitivity Point* bezüglich mehrerer Qualitätsmerkmale [28]. Mit dieser Methode können analog zu *SAAM* viele verschiedene Qualitätsmerkmale untersucht werden.

Beide Methoden wurden entweder mit Fallstudien oder in Industrieprojekten validiert. Daher kann von einer gewissen Reife der Methoden ausgegangen werden. In [28] werden szenariobasierte Bewertungsmethoden vorgestellt und verglichen. Für detailliertere Informationen wird daher auf diese Quelle bzw. auf die Veröffentlichungen des SEI verwiesen.

Insbesondere für die in dieser Arbeit betrachteteten kleinen und mittelständigen Unternehmen bietet der Einsatz von Architekturbewertung eine geeignete Maßnahme zur Qualitätssteigerung. Vielfach ist die Expertise in diesen Unternehmen auf einige wenige Personen verteilt. Wenn diese in strukturierter Weise ihre Anforderungen und Erfahrungen in Form einer Architekturbewertung kombinieren und dokumentieren, entsteht daraus eine Architektur, die auf einer großen Erfahrung basiert. Allerdings müssen die Architekturbewertungsmethoden in den meisten Fällen angepasst werden, da zur Besetzung aller Rollen oft sehr viele Personen benötigt werden (bei ATAM z.B. mindestens 18), die oft nicht zur Verfügung stehen.

2.6 Prototypen in der Softwareentwicklung

2.6.1 Definition

Der Begriff des Prototyps ist aus vielen Ingenieurdisziplinen bekannt. Insbesondere in der Automobilindustrie werden Prototypen der unterschiedlichen Fahrzeugtypen intensiv genutzt um die Funktionalität vor Produktion des eigentlichen Fahrzeugmodells zu erproben. Aber auch nichtfunktionale Eigenschaften wie das Design oder die Bedienbarkeit werden anhand von Prototypen überprüft und weiterentwickelt. Daher verwundert es nicht, dass auch die Entwicklung automobiler Steuergeräte stark prototypenbasiert durchgeführt wird. Während im gesamten Fahrzeugentwicklungsprozess unterschiedlich reife Prototypen entwickelt werden, müssen auch Steuergeräte für diese Prototypen in ebenso abgestuften Reifegraden vorliegen. Entsprechend werden nach den in frühen Phasen der Entwicklung eingesetzten Rapid-Control-Prototyping Systemen A-, B- und C-Muster Steuergeräte bis hin zum D-Muster, dem eigentlichen Seriensteuergerät [54], entwickelt.
In der klassischen Softwaretechnik wird der Begriff des Prototypen ebenfalls verwendet, jedoch unterscheidet sich dieser in seiner Bedeutung von dem Prototyp, wie er im Fahrzeugbau Verwendung findet. Im Fahrzeugbau ist die größte Herausforderung nach Fertigstellung des Prototyps der Aufbau der Serienproduktion. Diese Aufgabe ist in der Softwareentwicklung jedoch trivial, da die erstellte Software lediglich kopiert werden muss. Vielmehr ist der Software-Prototyp im herkömmlichen Sinne des Fahrzeugbaus eher ein *Mock-Up* oder eine *Attrappe*, die bestimmte Merkmale oder Funktionalitäten aufweist, andere aber vernachlässigt. Nach [41] wird „mit dem Wort *Prototyping* in der Software-Entwicklung eine Vorgehensweise bezeichnet, bei der Attrappen, die als Prototypen bezeichnet werden, entworfen, konstruiert, bewertet und revidiert werden. Der Zweck des Software-Prototypings ist es die Anforderungen zu klären und damit eine lange und teure Entwicklung zu vermeiden, die am Ende ein Produkt liefert, das der Kunde so nicht haben will“.
Das Glossar der IEEE gibt folgende Definitionen:

> **prototype** - „A preliminary type, form, or instance of a system that serves as a model for later stages or for the final, complete version of the system.“
>
> **prototyping** - „A hardware and software development technique in which a preliminary version of part or all of the hardware or software is developed to permit user feedback, determine feasibility, or investigate timing or other issues in support of the development process.“
>
> **rapid prototyping** - „A type of prototyping in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process.“
>
> IEEE Std. 610.12 (1990) [27]

Darüberhinaus lassen sich in Übereinstimmung mit [41] und [54] folgende allgemeine Aussagen zu Software-Prototypen ableiten:

- „Ein Prototyp ist lauffähig und funktionstüchtig.“
- „Ein Prototyp realisiert ausgewählte Aspekte des Zielsystems. Welche Aspekte in einem Prototyp umgesetzt werden, richtet sich nach den speziellen Fragen, die mit Hilfe des Prototypen geklärt werden sollen. Wichtig ist, dass die Aspekte, die ein Prototyp realisieren soll, vor seiner Entwicklung festgelegt werden, da diese die Konstruktion des Prototypen beeinflussen.“
- „Ein Prototyp wird vom Klienten geprüft und detailliert bewertet. Zeigt sich dabei, dass der Prototyp stark vom gewünschten System abweicht, so wird er modifiziert, bis die Klienten einverstanden sind.“

Vielfach wird in der Literatur noch nach zwei Arten von Prototypen unterschieden, die sich aus den im Prototyp abgebildeten Gegenstand ableitet. Diese Typen werden in [54] wie folgt definiert:

- „Horizontale Prototypen zielen auf die Darstellung eines Bereichs eines Software-Systems, stellen aber eine abstrakte Sicht dar und vernachlässigen Details.“
- „Vertikale Prototypen stellen dagegen einen eingeschränkten Bereich eines Software-Systems recht detailliert dar.“

Diese Unterscheidung ist nach Ansicht des Autors jedoch von untergeordneter Bedeutung, da vielmehr die Fragestellung im Vordergrund steht, die mit dem Prototyp beantwortet werden soll. Daher erfolgt zunächst eine Beschreibung des Prototypenentwicklungsprozess und eine Einordnung der Prototypentwicklung in den gesamten Softwareentwicklungsprozess.

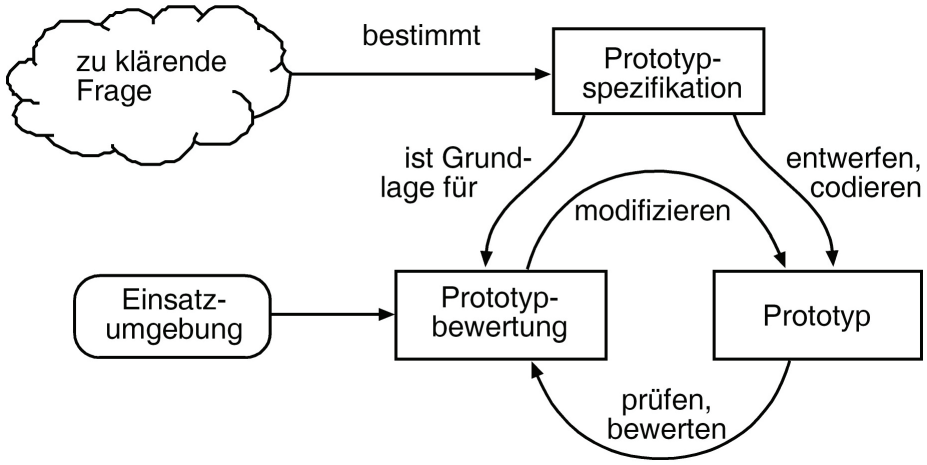

Abbildung 2.5: *Allgemeine Vorgehensweise beim Prototyping [41]*

2.6.2 Prototypenentwicklung

Häufig stehen die Anforderungen zu Beginn einer Entwicklung nur unvollständig und wenig konkret fest. Wenn dies der Fall ist, bietet es sich an zunächst einen Prototyp zu erstellen, um mehr Klarheit zu schaffen. Damit der Prototyp jedoch auch seinen Zweck erfüllt sollten nach [41] im Vorfeld folgende Fragen geklärt werden:

- „Welche Aufgabe und welchen Zweck hat der Prototyp, wie lauten also die offenen Fragen?“
- „Welche Personengruppen sind an der Entwicklung und insbesondere an der Bewertung des Prototyps beteiligt?“
- „Wie lange darf die Prototypentwicklung dauern und welche Kosten dürfen entstehen?“

Mit den Antworten auf diese Fragen als Basis, wird der Prototyp entwickelt. Dafür schlägt [41] einen Zyklus vor, wie er in Abbildung 2.5 dargestellt ist.
Der Zyklus, der rechts unten in der Grafik dargestellt ist, wird so lange durchlaufen, bis der untersuchte Aspekt zufriedenstellend im Prototyp modelliert ist. Der Prototyp kann im ersten Schritt als ein deskriptives Modell und damit als Abbild der ursprünglich vermuteten Anforderungen bezeichnet werden. Mit dem Durchlaufen des Zyklus von Prüfung und Modifikation wird der Prototyp Schritt für Schritt zu zu einem präskriptiven Modell, d.h. einer Vorgabe für das Zielsystem. Es handelt also bei einem Prototypen nach diesem Vorgehen um ein exploratives Modell [41].

Weiterhin muss die Entwicklung des Prototypen in den gesamten Steuergeräteentwicklungsprozess eingeordnet werden. Dient der Prototyp lediglich dem Testen einer bestimmten Funktion, ohne diese in das spätere Zielsystem zu übernehmen so spricht man von einem *Wegwerfprototyp*. Dabei kann es durchaus sein, dass ein solcher Prototyp trotzdem einen nicht unerheblichen Anteil an der erfolgreichen Steuergeräteentwicklung hat. Wird der Prototyp hingegen als Basis für das spätere Produkt verwendet, wird er *evolutionärer Prototyp* genannt. Beide Arten werden in der Industrie verwendet [54].
Das bekannteste Beispiel der evolutionären Entwicklungsweise stellen die bereits oben erwähnten A-, B-, C- und D-Mustersteuergeräte dar. Hierbei wird der Funktionsumfang von einer Evolutionsstufe zur nächsten ergänzt, bis das Zielsystem vollständig ist. Gleichzeitig dienen die Entwicklungssteuergeräte, ganz im Sinne des Prototyps, als Testobjekte, die bewertet und wieder modifiziert werden, um so den vollen Funktionsumfang iterativ zu erarbeiten (vgl. Abbildung 2.5).
Insbesondere bei Verwendung von Prototyping in der Serienentwicklung wird vielfach eine möglichst hohe Übereinstimmung des Verhaltens zwischen Experimentiersystem und Steuergerät angestrebt. In diesem Fall spricht man auch von einem zielsystemidentischen Verhalten des Prototypen [54].

2.6.3 Rapid Control Prototyping

Im Unterschied zum zielsystem-identischen Prototyping verfügen die als Experimentiersysteme ausgelegten klassischen Rapid-Control-Prototyping Systeme in der Regel über eine wesentlich höhere Rechenleistung [54]. Viele Anforderungen, die bei der Implementierung einer Software-Funktion auf dem Steuergerät berücksichtigt werden müssen, wie Festpunktarithmethik oder die begrenzten Hardware-Ressourcen, können deshalb vernachlässigt werden. Änderungen an den Software-Funktionen sind so einfacher und schneller möglich. Die Einbindung zusätzlicher I/O-Schnittstellen bei Experimentiersystemen ermöglicht zudem die frühzeitige Bewertung verschiedener Realisierungsalternativen etwa bei Sensoren und Aktuatoren.
Rapid-Control-Prototyping Systeme speziell für die Automobilindustrie werden heute von Unternehmen, wie z.B. dSpace[10] oder ETAS[11] angeboten. Die gemeinsamen Merkmale dieser Systeme können wie folgt zusammengefasst werden:

- Modellbasierte Software-Entwicklung mit Werkzeugen wie Matlab/Simulink, Targetlink oder ASCET
- Gut geeignet als Experimentiersystem in der Entwicklung
- Große Vielfalt von Schnittstellen

[10] http://www.dspace.de
[11] http://www.etas.com

- Prozessorleistung liegt deutlich über Serienniveau
- Schnittstellen zu realen Aktuatoren wie z.B. Einspritzventilen müssen über externe Endstufenmodule realisiert werden

Damit sind diese System sehr gut geeignet um Funktionen, wie Regler, zu entwickeln und unter realen Bedingungen zu testen. Sie sind ebenfalls gut geeignet um die Anforderungsbeschreibung anhand der erstellen Prototypen, wie in 2.6.1 beschrieben, von dem deskriptiven Charakter zum präskriptiven Charakter weiterzuentwickeln. Dafür können Simulationsergebnisse aus der Software-In-The-Loop (SIL) wie auch der Hardware-In-The-Loop (HIL) (vgl. [1]) Phase als Sollwerte oder Entwicklungsziele beim Übergang zur Serienentwicklung verwendet werden. Weiterhin können die erstellten Modelle zum Teil ebenfalls weiterverwendet werden. Sie müssen jedoch in die Entwicklungsumgebung für das Seriensystem überführt werden.
Beim Übergang auf die seriennahe Hardware müssen jedoch alle Ergebnisse aus der Rapid-Prototyping-Phase unter den geänderten Voraussetzungen validiert werden, so dass zusätzlicher Aufwand entsteht. Ein weiterer Nachteil ist, dass Ergebnisse in späteren Phasen der Entwicklung die durch Modellmodifikationen, im Hinblick auf die geänderten Hardwarevoraussetzungen erzielt werden, für zukünftige Projekte nicht ohne weiteres auf dem Rapid-Control-Prototyping-System nachgestellt werden können.

2.7 Prozessmodelle in der Softwareentwicklung

2.7.1 V-Modell

Der in der Automobilindustrie am Weitesten verbreitete Software- und Systementwicklungsprozess ist das V-Modell. Basierend auf den Arbeiten von Barry Boehm wurde das V-Modell unter anderem auch von den deutschen Bundesbehörden weiterentwickelt und verbindlich für öffentliche Softwareprojekte vorgeschrieben (vgl. Abschnitt 2.7.1). An diesem Standard orientiert sich auch die Automobilindustrie in ihrem Vorgehen bei vielen Entwicklungsprojekten. Eine gute Übersicht und Einführung in die V-Modelle gibt Hesse in [52]. Die folgende Beschreibung lehnt sich an diese Quelle an.
Abbildung 2.6 zeigt die erste V-förmige Darstellung eines Softwareentwicklungsprozesses, wie sie von Boehm 1979 in [9] veröffentlicht wurde. Der lineare Projektverlauf, wie er in dem bis dahin vorherrschenden Wasserfallmodell verwendet wurde, ist nach wie vor zu erkennen. Durch die zweidimensionale Darstellung lassen sich jedoch neben der horizontal von links nach rechts fortschreitenden Zeit, auch weitere Details in dem vertikalen Verlauf darstellen. Den entwerfenden Aktivitäten im linken, absteigenden Ast des „V“ stehen im rechten, aufsteigenden Ast testende Aktivitäten gegenüber.

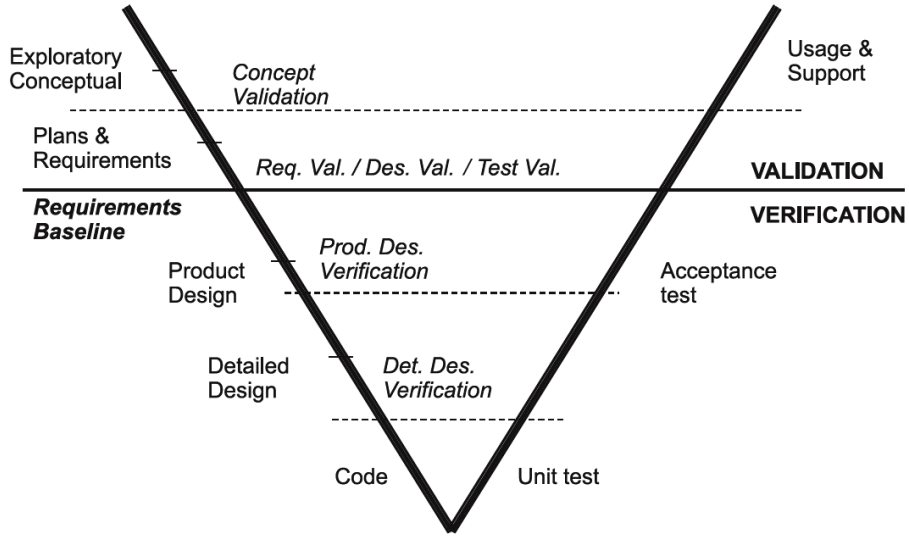

Abbildung 2.6: *Das V-Modell nach Boehm [9]*

Weiterhin ist die horizontale, mit Requirements Baseline bezeichnete, Linie auffallend. Sie dient dazu den Auftraggeber und Nutzer (oben) vom Entwickler (unten) abzugrenzen. Die Linie wird weiterhin auch zur begrifflichen Abgrenzung zwischen Validierung (oben) und Verifikation (unten) verwendet. Mit Verifikation ist hierbei eine mehr oder weniger systematische und formale Überprüfung der Spezifikation bezeichnet, deren Vorhandensein auf den einzelnen Stufen zwingend erforderlich ist. Im oberen Bereich gibt es in der Regel jedoch keine formale Spezifikation. Und wenn es sie z.B. in Form von niedergeschriebenen Anforderungen gibt, so ist sie in den meisten Fällen unvollständig. Daher entspricht die Überprüfung hier einer Validierung, die typischer Weise nicht mit dem Ergebnis „falsch“ oder „richtig“, sondern höchstens mit „angemessen“ oder „nicht angemessen“ endet. Boehm drückt es nach [52] so aus: „*Verification* beantwortet die Frage: *Haben wir das Produkt richtig gebaut?* während *Validation* auf die Frage: *Haben wir das richtige Produkt gebaut?* antwortet.“
In der Automobilindustrie steht an der Stelle der Requirements Baseline häufig ebenfalls ein Übergang von Auftraggeber zu Lieferant (in der Regel OEM zu Supplier). Die Abnahme der Entwicklung durch den Auftraggeber entspricht demnach der Verifikation, während die Validierung beim Auftraggeber durchgeführt wird.

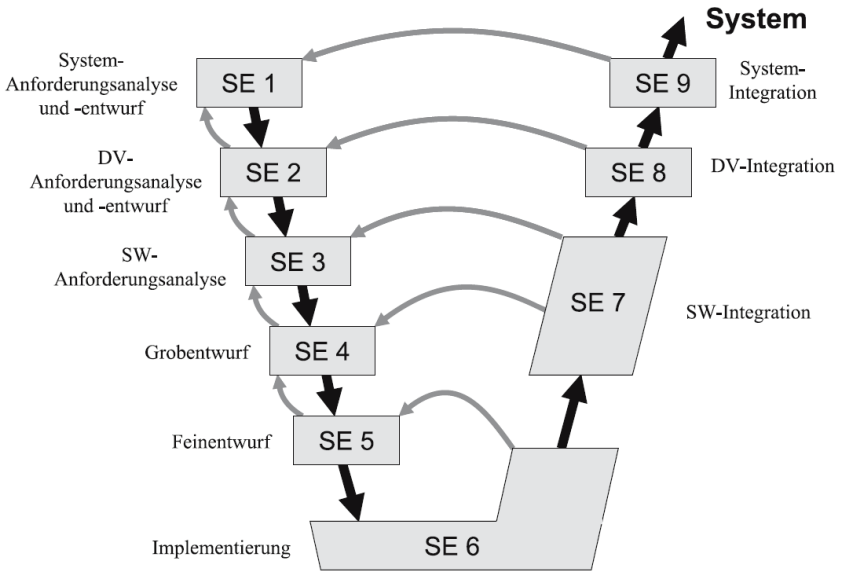

Abbildung 2.7: *V-Modell 97: Tätigkeiten zur Systemerstellung [52]*

Die V-Modelle der deutschen Bundesbehörden

Während der 1980-er Jahre wurde im Auftrag der deutschen Bundesbehörden ein Entwicklungsstandard für Software-Projekte mit dem Namen „V-Modell“ entworfen. Nach [52] „wurde dieser unter Federführung der IABG (Industrieanlagen-Betriebsgesellschaft) in Ottobrunn ursprünglich für das Bundesministerium für Verteidigung entwickelt und dort 1986 offiziell eingeführt. Später wurde sein Anwendungsbereich ausgedehnt, 1992 vom Innenministerium übernommen und ab Mitte der 1990-er Jahre wurde das V-Modell zum verbindlichen Standard für die Entwicklung von IT-Systemen der Bundesbehörden im zivilen und militärischen Bereich.Wesentliche Ziele dieser Entwicklungen bestanden darin, den Software-Entwicklungsprozess für die genannten Bereiche zu standardisieren, transparenter zu machen und damit die Qualität solcher Prozesse zu verbessern, ihre Kosten zu begrenzen und in Angebots- und Vergabeverfahren für bessere Vergleichbarkeit zu sorgen.“
Einer der wichtigsten Beiträge des *V-Modell 97* für die Weiterentwicklung von Prozessmodellen liegt in dem Rollenkonzept, das den Prozess-Beteiligten Rollen zuwies (wie z.B. Entwickler, Qualitätssicherer). Die beiden wichtigsten Rollen, die der Auftraggeber und Benutzer, wurden dabei allerdings nicht berücksichtigt.
Die Tätigkeiten zur Systemerstellung (SE) sind in Abbildung 2.7 dargestellt. Sie sehen ebenfalls eine V-förmige Darstellung wie im Boehm’schen Modell vor. Dabei

fällt auf, dass die Requirements Baseline im Modell nicht mehr vorhanden ist. Jedoch sind die bei Boehm nur angedeuteten Querbezüge zwischen den beiden Ästen des „V“ auf der jeweils gleichen Ebene nun durch Pfeile ersetzt, die formelle Prozessschritte zur Qulitätssicherung darstellen. Die Boehm'sche Unterscheidung zwischen *validation* und *verification* ist (wie die Requirements Baseline) nicht mehr sichtbar [52], obwohl sie in der Praxis der Automobilindustrie wie oben erläutert noch eine große Rolle spielt.
Als Nachfolger des V-Modells '97 wurde in den Jahren 2003 bis 2005 der Nachfolger das V-Modell XT entwickelt und im Februar 2005 erstmals in der Version 1.0 veröffentlicht [52]. Eine der wesentlichsten Weiterentwicklungen ist im V-Modell XT ist das weiter ausgebaute Konzept des *Tailorings*. Damit ist die Anpassbarkeit des Modells an unterschiedliche Projekte gemeint. So unterstützt das V-Modell XT z.B. auch drei verschiedene Vorgehensarten für die Entwicklung: Inkrementell, komponentenbasiert und sogar agil. Insbesondere für den Einsatz in der Automobilindustrie muss dabei darauf geachtet werden, das der angepasste Prozess auch noch den Forderungen von CMMI oder Automotive SPICE entspricht. Die größte Hürde für die Projektbeteiligten beim ersten Einsatz von V-Modellen ist allerdings die Vielzahl neuer Dokumente, die zu erstellen sind. So enthält ein V-Modell XT, bei dem noch kein Tailoring erfolgt ist, bereits über 500 Projekt-Artefakte, die in Dokumenten festgehalten werden müssen. Man sollte sich daher auf die Frage „Sollen wir nun Dokumentation erstellen oder ein Produkt bauen?“ vorbereiten [43]. Eine erste Diskussion zum V-Modell XT gibt [41], umfangreiche Dokumentation ist neben der öffentlich zugängliches Dokumentation des IABG [12] in [52] zu finden.

2.7.2 Agile Methoden

Als Gegenentwurf zu den sogenannten schwergewichtigen Prozessen wie dem V-Modell (vgl. 2.7.1) oder dem Unified Process [35] veröffentlichten Kent et al. im Jahre 2001 das Manifest für agile Softwareentwicklung[13]. Daraufhin wurden von den Autoren in den vergangenen Jahren zahlreiche Methoden vorgeschlagen, die alle die im Folgenden nach [20] frei übersetzten gemeinsamen Ziele verfolgen:

- „Prozesse und Werkzeuge sind wichtig, aber wichtiger sind im Zweifelsfall die am Projekt Beteiligten und ihre Interaktionen.“
- „Dokumentation (auch eine umfassende) ist wichtig, aber im Zweifelsfall hat die Ablieferung einer lauffähigen Software Priorität.“
- „Verträge sind wichtig, aber im Zweifelsfall ist die Zusammenarbeit mit dem Kunden wichtiger.“

[12] http://www.v-modell.iabg.de/
[13] Manifesto for Agile Software Development, http://www.agilemanifesto.org/

- „Pläne sind wichtig, aber wenn es erforderlich ist, müssen die Anforderungen eben im Sinne des Kunden geändert (also angepasst) werden."

Diesen Grundsätzen kann insbesondere in kleineren Entwicklungsteams, wie sie bei KMUs üblich sind, zugestimmt werden. Bezogen auf das hier fokussierte Anwendungsfeld sollen zwei agile Methoden herausgegriffen und vorgestellt werden, die im Umfeld automotiver Embedded-Software Projekte verstärkt diskutiert werden.

- *Crystal*, vorgestellt von Cockburn [12]
- *Scrum*, vorgestellt von Schwaber und Sutherland [57]

Crystal

Die Crystal-Familie mit agilen Prozessmodellen wurde von Alistair Cockburn in den 1990 er Jahren entwickelt. Der zentrale Gedanke dieser Modellfamile ist es den Entwicklungsprozess der jeweiligen Problemstelleung bzw. dem jeweiligen Projekt in Bezug auf die Teamgröße und und der Kritikalität anzupassen. Dabei entspricht Cockburn Farbenbezeichnung der Unterteilung nach Teamgröße (je dunkler die Farbe desto größer das Team). Der Härtegrad des zu verwendenden Prozessvariante entspricht der Kritikalität bezüglich eines Softwarefehlers in der Anwendung für die die Software ertellt werden soll. Diese Unterteilung ist sinnvoll, um dem erhöhten Kommunikationsaufwand bei großen Teams und dem erhöhten AUfwand für Qualitätssicherung bei kritischen Applikationen gerecht zu werden. Aber dennoch definiert Cockburn keine definierte Vorgehensweise, sondern lediglich Prinzipien und zu erreichende Projekteigenschaften. Die einfachste Variante für bis zu sechs Teammitglieder und für Projekte mit geringer Kritikalität wird Crystal Clear genannt. Ein gute Einführung in Crystal gibt [20].
Umfragen unter Teammitgliedern [12] ergaben einige Projekteigenschaften, die als wichtig identifiziert wurden. Von diesen sind nach [20] zumindest folgende auch für Crystal Clear Projekte vorgeschrieben:

- *Regelmäßige Lieferung*: „Der getestete Code soll regelmäßig an den Kunden ausgeliefert werden. Somit kann der Kunde auch in gleichmäßigen zeitlichen Abständen sein Feedback zum aktuellen Stand des Software-Projekts abgeben. Abweichungen von den Anforderungen können somit schnell erkannt werden. Für den Kunden ist es einfacher, die Umsetzung seiner Anforderungen am „laufenden System" zu überprüfen, als die Korrektheit umfassender Spezifikationsdokumente zu erkennen. Außerdem motiviert die Zustimmung des Kunden das Entwicklungsteam."
- *Reflektierte Verbesserung*: „In Reflexionsbesprechungen listet das Team auf, was im Projekt gut funktioniert und was nicht. Das Team diskutiert über

Verbesserungsmöglichkeiten und stellt Praktiken ab, die sich nicht bewährt haben. Die Ergebnisse der Teambesprechungen werden in die nächste Entwicklungsphase aufgenommen. Eine solche Teambesprechung sollte mindestens alle drei Monate oder nach Bedarf stattfinden. Sie kann von einer halben Stunde bis zu einem halben Tag dauern."

- *Osmotische Kommunikation*: „Ein Team verfügt über eine verdichtete Kommunikation, wenn kurze und ergiebige Kommunikationswege zur Verfügung stehen. Die kostengünstigste Art der Kommunikation entsteht in kleinen Teams, die in einem Raum arbeiten. Man spricht dann auch von osmotischer Kommunikation. Allein durch die räumliche Nähe der Teammitglieder fließen Informationen. Wenn sich ein Mini-Team von Entwicklern über ein Thema unterhält, bekommen das die anderen Teammitglieder mit. Vorausgesetzt das Gespräch wird nicht zu laut geführt, können sie das Gespräch wegfiltern, es sei denn, sie sind auch von dem angesprochenen Thema betroffen. Dann können sie sich einmischen und aktiv am Gespräch des Mini-Teams teilnehmen."

Obwohl Crystal in der Praxis wenig Verwendung findet, stellen die genannten Projekteigenschaften doch wichtige Merkmale in den Vordergrund, die sich auch für Entwicklungen nach anderen Prozessmodellen eignen. Dies gilt nach Meinung des Autors insbesondere für die Entwicklungen in KMU.

Scrum

Bei Scrum handelt es sich im Gegensatz zu anderen agilen Prozessmodellen um eine agile *Projektmanagementmethode*. Im Vordergrund steht insbesondere ein definierter Projektablauf mit geringem Management-Aufwand. Damit folgt Scrum der Idee des Lean Management.
Scrum wurde von Ken Schwaber und Jeff Sutherland zu Beginn der 1990er Jahre entwickelt und unter anderem in [57] veröffentlicht. Aufallend sind die in Scrum festgelegten Projektrollen

- Product Owner
- Team
- ScrumMaster

bei denen insbesondere die Rolle des „klassischen" Projektmanagers fehlt. Dieses Fehlen führt in der Praxis häufig zu Problemen bei der Einführung von Scrum, da mit dem „Projektmanager" in vielen Unternehmen auch eine Arbeitsposition bezeichnet wird und diese Mitarbeiter erst einmal einen neuen Platz innerhalb von Scrum finden müsssen.

Die Definition der Rollen wird in [20] wie folgt beschrieben:

- „*Der Produktverantwortliche (Product Owner)* übernimmt die Sichtweise des Endkunden. Er steuert die Software-Entwicklung und arbeitet eng mit dem Entwicklungsteam zusammen. Er ist für das Entwicklungsteam ständig verfügbar und er bestimmt im Projekt alleine „wohin die Reise geht“. Die Rolle des Product Owners ist bivalent. Er darf nicht die „Unschuld“ des Kunden annehmen und sich neue Features wünschen, sondern wird auch sofort auf die Realisierungsmöglichkeit achten. Es besteht somit die Gefahr, dass tolle Ideen nicht realisiert werden, weil der Product Owner nicht an ihre Realisierungsmöglichkeit glaubt“.

- „*Das Team* entwickelt das Produkt, also die Software. Seine Zusammensetzung ist entscheidend für das Gelingen des Projekts. Dabei geht es nicht nur um die fachliche Qualifikation, sondern auch um die Teampsychologie, also das „Zusammenpassen“ der Teammitglieder. Wichtig ist, potenziellen Teammitgliedern die Möglichkeit zu geben, sich für interessante Projekte zu bewerben. Damit ist zumindest sichergestellt, dass Interesse vorhanden ist. Scrum geht nicht weiter auf die Rollenverteilung im Team ein. Scrum ist, wie schon erwähnt, eine Managementmethode. Es ist jedoch klar, dass alle Rollen oder besser gesagt Fähigkeiten, die für eine erfolgreiche Projektentwicklung notwendig sind, vorhanden sein müssen. Ein Team muss klein und selbstorganisiert sein und kommt ohne Teamleiter aus. Das agile Team muss osmotisch kommunizieren [12].“

- „*Der ScrumMaster (Change Agent)* ist der Prozessverantwortliche. Er ist der Freund des Teams. Er coacht das Team, wenn es um die Einhaltung des Prozesses geht. Insbesondere bei mit Scrum unerfahrenen Teams hilft er, den Prozess zu etablieren. Macht er seine Arbeit gut, macht er sich idealerweise überflüssig! Die Erfahrung zeigt jedoch, dass es nicht soweit kommt, da Teams dazu neigen, den gewählten Prozess (manchmal bis zur Unkenntlichkeit) abzuändern. Deswegen muss der ScrumMaster immer wieder eingreifen und die Korrektheit des Prozesses überprüfen. Bei erfahrenen Teams kann er allerdings die Zeit, die er mit dem Team verbringt, langfristig reduzieren. Der Beziehung zwischen ScrumMaster und Team ist eine Eins-zu-Eins-Beziehung. Der ScrumMaster arbeitet beim Team.“

Ein einfacher Ablauf charakterisiert den Scrum-Prozess (vgl. Abbildung 2.8). Der Ablauf beginnt mit der Vision des Product Owners. Nachdem aus der Vision durch

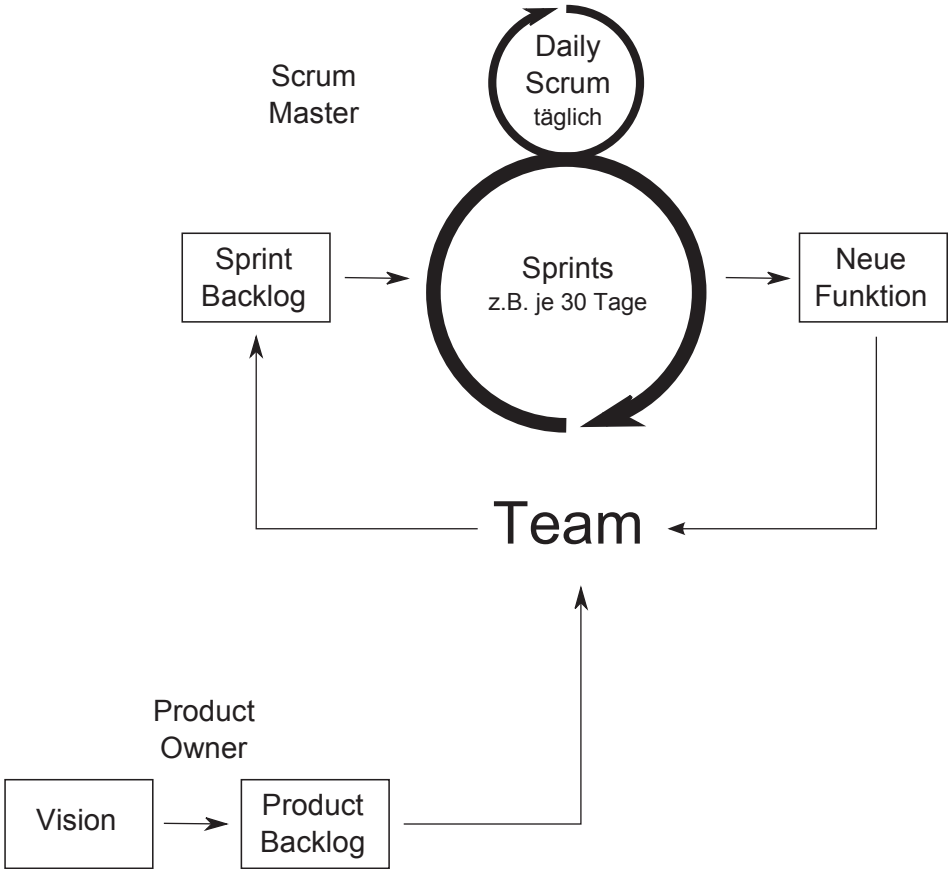

Abbildung 2.8: *Ablauf des Scrum-Prozesses in Anlehnung an [20]*

eine Management-Entscheidung ein konkretes Projekt entstanden ist, wird der Product Backlog vom Product Owner gefüllt. Dabei werden die Anforderungen an die Software dokumentiert und gleichzeitig priorisiert.
In Iterationszyklen, den sogenannten Sprints, wird die Software anschließend vom Team erstellt. Dabei wird für jeden Sprint ein Endtermin vereinbart der eingehalten werden muss. Die Dauer eines Sprints sollte 30 Tage nicht überschreiten. Zu Beginn eines Sprints wird der sogenannte Sprint-Backlog vom Team erstellt. Hierbei werden die Anforderungen für das im Sprint zu realisierende Software-Inkrement festgehalten. Anschließend erfolgt die eigentliche Implementierung mit den im Sprint Backlog festgelegten Features. Dabei entwickelt das Team ohne Störung von außen.
Weiterhin findet jeden Tag eine kurze Teambesprechung (Daily Scrum) statt. Hierbei ist neben dem Team auch der ScrumMaster (zwingend) und der Product Owner (optional) anwesend. Hierbei werden tagesaktuell Probleme angesprochen und Aufgaben verteilt
Am Ende jeden Sprints wird ein Sprint Revies abgehalten, bei dem das enstandene Software-Inkrement vom Produact Owner geprüft und abgenommen wird. Nur vollständige und fehlerfreie Arbeitsergebnisse werden akzeptiert.

2.7.3 Prozessbewertungsmethoden

Die Gründung des Software Engineering Instituts (SEI) auf dem Gelände der Carnegie Mellon University in Pittsburgh, Pennsylvania, im Jahre 1987 wurde durch das US-Verteidigungsministerium initiiert, das bis heute der weltweit größte Auftraggeber für Software-Entwicklung ist. Aufgabe des neuen Instituts war es die Qualität der zahlreichen Softwareentwicklungsprojekte zu erhöhen, da man festgestellt hatte, das nur ein Bruchteil der gelieferten Software aus den unterschiedlichsten Gründen wirklich nutzbar war. Der Ansatz des Instituts war es, die Produktqualität durch die Prozessqualität bei der Softwareerstellung sicherzustellen. Daher entwickelte man ein Modell, um die Entwicklungsprozesse in den zuliefernden Unternehmen zu bewerten. Bereits 1991 konnte so die erste Version des Capability Maturity Models (CMM) vorgestellt werden [25]. Dieses Modell legte auf einer Skala von 1 bis 5 Reifegrade von Entwicklungsprozessen fest. In den ersten Jahren nach seiner Veröffentlichung wurden Bewertungen von lediglich 1 bis 3 erreicht. Erst ab 1995 schafften es einige Unternehmen den Reifegrad 5 zu erreichen. Basierend auf den Erfahrungen mit CMM in den ersten Jahren wurde im Jahre 1997 das Nachfolgemodell CMMI (Capability Maturity Model Integrated) vorgestellt, das heute in der Version 1.2 aus dem Jahre 2006 vorliegt [58]. CMMI wird heute nicht nur für Softwareprojekte sondern auch für andere Dienstleistungsprojekte verwendet.
Als europäische Reaktion auf das amerikanische CMM wurde in den Neunzigerjahren ein EU Projekt initiiert, in dem ein Bewertungsverfahren entwickelt werden sollte,

das besser an die europäischen Gegenbenheiten angepasst ist. Das Verfahren wurde BOOTSTRAP genannt [49]. Basierend auf CMM, Bootstrap und anderen Ansätzen wurde schließlich durch die ISO das Bewertungsverfahren SPICE (Software Process Improvement and Capability dEtermination) entwickelt. SPICE harmonisiert die bestehenden Ansätze und definiert einen Standard für Bewertungsverfahren. SPICE liegt jedoch erst seit 2003 in der fünfteiligen internationationalen Norm ISO/IEC 15504 [32] vor. Das heisst zwischen der Veröffentlichung von CMM und SPICE liegen mehr als 10 Jahre, was die weite Verbreitung von CMM erklärt.
Insbesondere die deutsche Automobilindustrie setzte jedoch schon früh auf den neuen Standard, indem die Herstellerinitiative Software (HIS)[14] beschloss den neuen Standard einheitlich für die Lieferantenbewertung einzusetzen. Die ISO/IEC 15504 (SPICE) erlaubt die Verwendung verschiedener Prozessreferenzmodelle (PRM), und Prozessassessmentmodelle (PAM) sowie Assessmentmethoden. Dadurch wird es ermöglicht, auf verschiedene Bedürfnisse angepasste Modelle zu definieren. Dies führte zur Definition von „SPICE for SPACE“ und „Automotive SPICE©“[15]. Da in dieser Arbeit die Softwareentwicklung für die Automobilindustrie fokussiert wird, sollen die Grundlagen von Automotive SPICE im Folgenden näher erläutert werden, ohne weiter auf CMM(I) oder SPICE einzugehen.

Automotive SPICE besteht aus den zwei Komponenten *Prozessreferenzmodell (PRM)* und dem *Prozessassessmentmodell (PAM)*. Für die praktische Anwendung ist die Kenntnis des Prozessassessmentmodells relevant und ausreichend. Nach [45] kann mit ihm die Prozessreife anhand von Indikatoren in zwei Dimensionen unterteilt werden:

- *Prozessdimension:* „Diese enthält für alle Prozesse die Indikatoren für die Beurteilung, inwieweit die Prozesse durchgeführt werden. Diese Indikatoren sind für jeden Prozess verschieden und bilden eine wichtige Voraussetzung zur Erreichung von Level 1.“

- *Reifegraddimension:* „Diese enthält die Indikatoren zur Beurteilung der verschiedenen Reifegradstufen. Diese Indikatoren sind für alle Prozesse gleich.“

Die Prozesse, die nach [45] in Automotive SPICE verwendeten Prozesse sind in Abbildung 2.9 dargestellt. Die Definitionen der Prozesse in Automotive SPICE haben eine einheitliche Struktur. Unter anderem werden hier die Basispraktiken (*base practices*) als Aktivitäten definiert, durch die spezifiziert wird *was* umgesetzt werden soll, aber nicht *wie*. Mit Hilfe der Basispraktiken werden die ebenfalls definierten Prozessergebnisse (*process outcomes*) erzielt, die wiederum als eine Detaillierung des

[14] http://www.automotive-his.de
[15] http://www.automotivespice.com

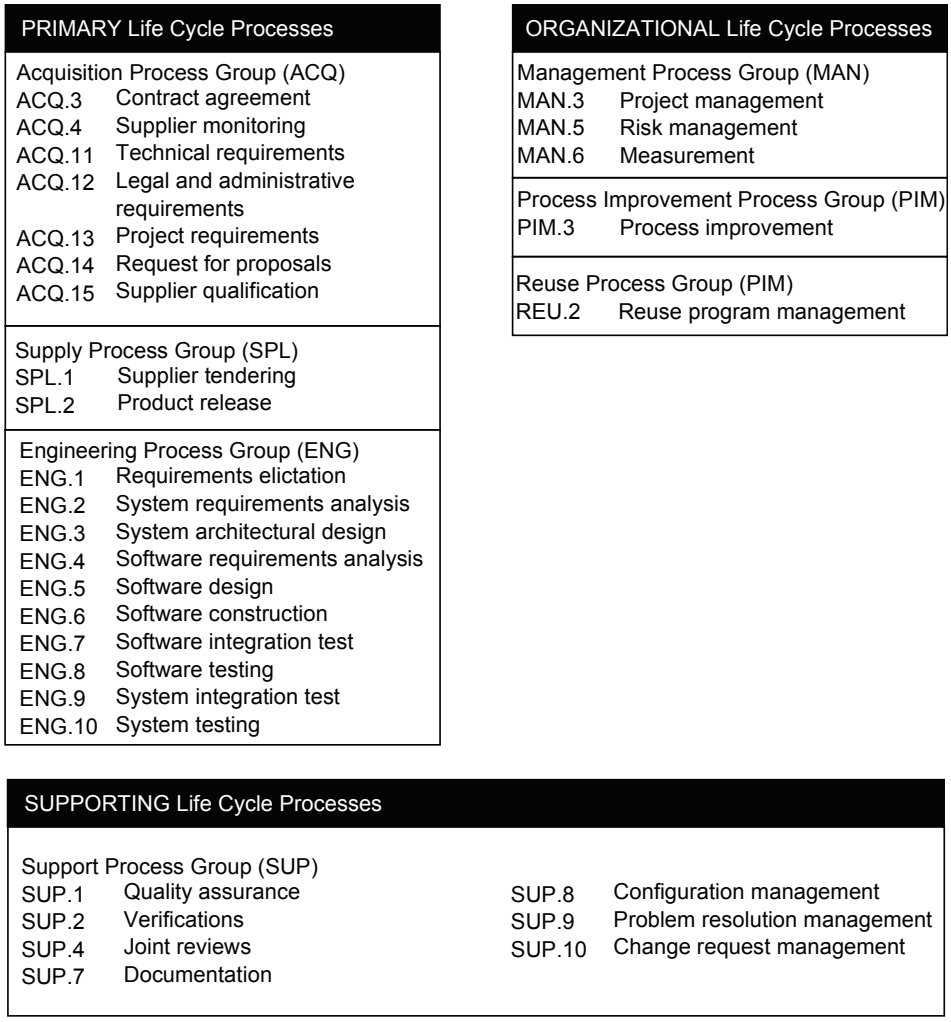

Abbildung 2.9: *Prozesse in Automotive SPICE [45]*

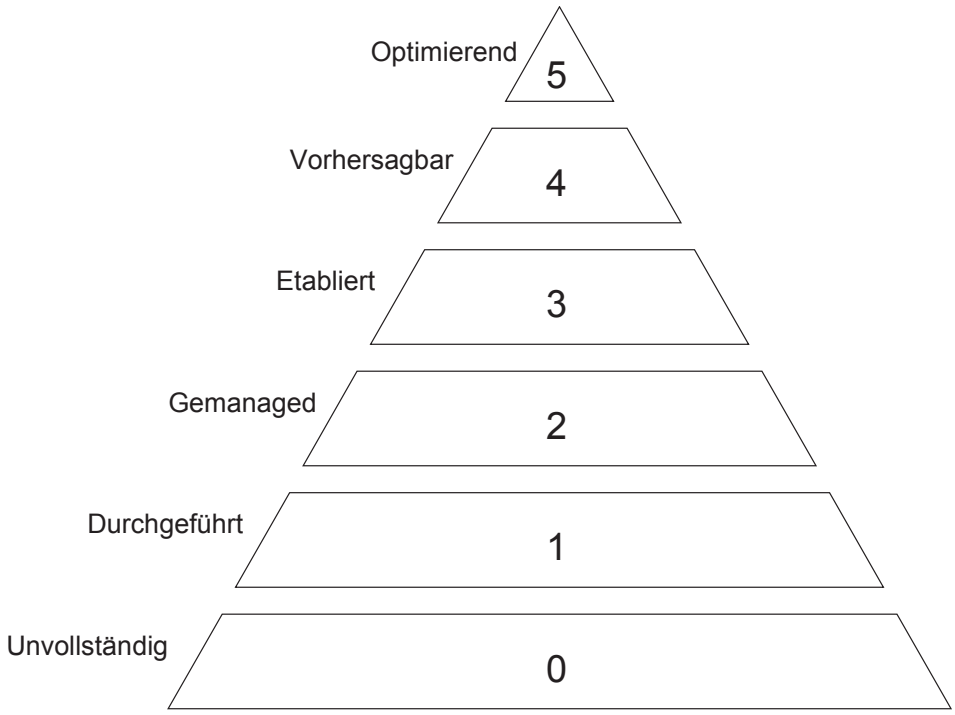

Abbildung 2.10: *Die sechs Reifegradstufen in Automotive SPICE*

Prozesszweckes (*process purpose*) bezeichnet werden können. Weiterhin werden Arbeitsprodukte (*output work products*) typische Ergebnisse eines Prozesses, die jedoch nicht zwingend sind. Gemeinsam mit den Basispraktiken sind sie jedoch als objektiver Nachweis für die Erfüllung des Prozesszweckes zu verstehen. Sie werden daher auch als Indikatoren für die Prozessdurchführung (*process performance indicators*) bezeichnet und sind die Kriterien für die Erreichung von Level 1 [45]. Automotive SPICE verwendet sechs Reifegradstufen für Prozesse (vgl. Abbildung 2.10), die in [45] wie folgt beschrieben werden:

- Level 0: Unvollständig (*Incomplete*)
  „Der Prozess ist nicht implementiert oder der Zweck des Prozesses wird nicht erfüllt. Projekterfolge sind durchaus möglich, basieren dann aber alleine auf den individuellen Leistungen der Mitarbeiter."

- Level 1: Durchgeführt (*Performed*)
  „Der implementierte Prozess erfüllt den Zweck des Prozesses. Dies bedeutet, dass grundlegende Praktiken implementiert sind und definierte Prozessergebnisse erzielt werden."

- Level 2: Gemanaged (*Managed*)
  „Die Prozessausführung wird zunächst geplant und verfolgt und die Planung wird ständig fortgeschrieben. Die Arbeitsprodukte des Prozesses sind adäquat implementiert, stehen unter Konfigurationsmanagement, werden qualitätsgesichert, gemanaged und fortgeschrieben."

- Level 3: Etabliert (*Established*)
  „Es existiert ein organisationseinheitlich festgelegter Standardprozess. Ein Projekt verwendet eine angepasste Version dieses Standardprozesses (sog. „definierter Prozess"), der daraus mittels *Tailoring* abgeleitet wird. Dieser ist in der Lage, definierte Prozessergebnisse zu erreichen."

- Level 4: Vorhersagbar (*Predictable*)
  „Bei der Ausführung des definierten Prozesses werden detaillierte Messungen durchgeführt und analysiert, die zu einem quantitativen Verständnis des Prozesses und einer verbesserten Vorhersagegenauigkeit führen. Mittels statistischer Methoden wird der definierte Prozess zwischen oberen und unteren Eingriffsgrenzen gesteuert. Die Qualität von Arbeitsprodukten ist quantitativ bekannt."

- Level 5: Optimierend (*Optimizing*)
  „Basierend auf den Geschäftszielen der Organisation werden quantitative Prozessziele gesetzt und deren Einhaltung kontinuierlich verfolgt. Die Prozesse werden fortlaufend verbessert, innovative Ansätze und Techniken werden erprobt und ersetzen weniger effektive Prozesse, um dadurch vorgegebene Ziele besser zu erreichen."

Mit Hilfe der sogenannten Prozessattribute wird das Vorliegen eines bestimmten Reifegrades beurteilt. Sie sind somit den Reifegradstufen zugeordnet und dienen deren inhaltlichen Charakterisierung.
Der Durchbruch zum Einsatz von Reifegradmodellen in der Automobilindustrie geschah 2001 durch die Entscheidung der Herstellerinitiative Software (HIS)[16], SPICE und später Automotive SPICE zur Lieferantenbeurteilung im Software und Elektronikbereich einzusetzen. Automotive SPICE hat sich in diesem Bereich nahezu flächendeckend durchgesetzt [45]. Für die internen Prozesse, d.h. für die eigene Entwicklung, hat jedoch CMMI einen wachsenden Anteil. Dies lässt sich vor allem auf die generische Anwendbarkeit des Modells auf Software, Hardware, Systeme und Dienstleistungen zurückführen. Für SPICE möchte man ebenso die generische Anwendbarkeit definieren. Jedoch liegt der Teil 6 der ISO Norm 15504 [32] bis heute lediglich als Entwurf vor.

[16] www.automotive-his.de

2.7.4 Prozessnotation

Neben einer textuellen Beschreibung von Prozessen eignet sich eine grafische Notation vor allem dazu die Zusammenhänge von mehreren Prozesskomponenten und Prozessen darzustellen. Hierfür wurden in der Vergangenheit überwiegend proprietäre Darstellungen verwendet, die sich häufig an den in diesem Bereich eingesetzten Softwarewerkzeugen (Business Process Management-Systemen, BPMS) orientierten [2]. Im Bereich der fachlich orientierten Geschäftprozessmodellierung wird häufig die Notation der Ereignisgesteuerten Prozesskette (EPK) eingesetzt. Deren Verbreitung erfolgte insbesondere im Umfeld von SAP[17]-Einführungen. Andere Standards, wie z.B. die Aktivitätsdiagramme der Unified Modeling Language (UML) (vgl. Abschnitt 2.1.4), konnten sich in der Praxis nicht für die Geschäftsprozessmodellierung durchsetzen. Ihr Einsatz blieb weitgehend auf den Bereich des objektorientierten Softwareentwurfs beschränkt, wo die UML der akzeptierte Standard ist [2].
Mit der *Business Process Model and Notation (BPMN)* scheint sich nun erstmals ein Standard für die Prozessmodellierung auf breiter Basis durchzusetzen. Diese Notation wurde speziell für die Dokumentation von Geschäftsprozessen entworfen und geht auf Arbeiten beim Unternehmen IBM zurück [2]. Die Weiterentwicklung und Spezifikation wurden an die Object Management Group (OMG) übertragen, damit die Notation eine allgemeine Verbreitung findet. BPMN liegt in der aktuellen Spezifikation 2.0[18] vor, die im Januar 2011 als Release veröffentlicht wurde. Eine gute Einführung in BPMN gibt Allweyer in [2].
In Abbildung 2.11 sind einige Basis-Notationselemente der BPMN dargestellt. Organisationseinheiten wie z.B. Unternehmen oder Abteilungen werden in sogenannten *Pools* und *Lanes* abgebildet, in denen Aktivitäten eingetragen werden, die der jeweiligen Organisationseinheit zugeordnet sind. Ein Prozess wird durch ein *Start-Event* gestartet und führt in der Regel zu einer oder mehreren Aktivitäten. Beendet wird ein Prozess durch ein *End-Event*. Innerhalb von Prozessen werden Verzweigungen durch sogenannte *Gateways* dargestellt. Arbeitsprodukte oder Dokumente werden als *Data-Objects* abgebildet. Erläuterungen können in die Diagramme durch *Text-Annotations* eingetragen werden. Eine ausführliche Dokumentation ist im Internet ebenfalls auf den Seiten der OMG zu finden.

[17] http://www.sap.de
[18] http://www.omg.org/spec/BPMN/2.0/

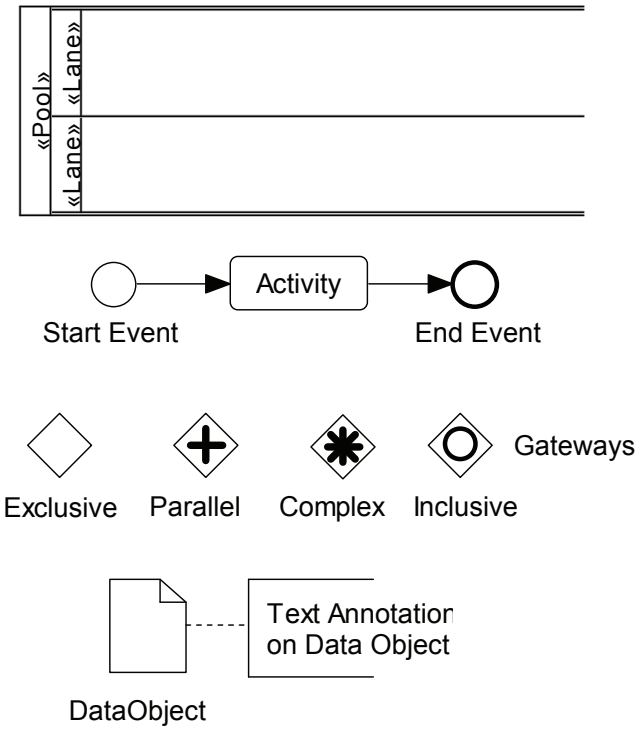

Abbildung 2.11: *Einige Basis-Notationselemente der BPMN*

3 Ansatz für einen Software-Entwicklungsprozess

Basierend auf den in Kapitel 2 vorgestellten Methoden wird in diesem Kapitel der Ansatz für einen an kleine und mittelständische Unternehmen angepassten Entwicklungsprozess dargestellt. Die Entwicklung automobiler Steuergeräte erfolgt in der Regel im Auftrag von Motoren- oder Fahrzeugherstellern. Es handelt sich hierbei also um ein klares Kunde-Lieferanten Verhältnis. Bei vielen Unternehmen ist eine Lieferantenbewertung üblich und ist Teil ihres Produktentstehungsprozesses. Diese Bewertungen erfolgen heute in der Regel nach dem Automotive SPICE-Standard [63], so dass bei der Definition eines Entwicklungsprozesses beim entwickelnden Unternehmen auf Kompatibilität zu diesem Standard geachtet werden sollte. Gleichzeitig soll jedoch der Vorteil der Flexibilität von kleinen Unternehmen beibehalten werden, was für einen „leichtgewichtigen“ Entwicklungsprozess spricht, wie er z.B. bei den agilen Ansätzen verfolgt wird (vgl. Abschnitt 2.7.2). Das zugrunde liegende Vorgehensmodell des Prozesses sollte das in der Automobilindustrie fast ausschließlich verwendete V-Modell aufgreifen. Es sollte sich an den dort verwendeten Begrifflichkeiten orientieren und das Vorgehen prinzipiell übernehmen. Dennoch können die Prozesse der Großunternehmen nicht einfach kopiert werden, da diese Prozesse auch dazu entwickelt wurden um neben der Qualitätssicherung auch die massiv verteilte Entwicklung in großen Teams zu organisieren. Daher wird hier z.B. viel Wert auf eine feingranulare Anforderungs- und Architektur, und Designbeschreibung gelegt, um die Umsetzung auf viele Mitarbeiter zu verteilen. In kleinen Unternehmen ist diese feine Granularität nicht notwendig, da durch die in der Regel kleineren Projekte jeder Mitarbeiter für einen größeren Umfang innerhalb eines Projektes zuständig ist. Somit kann an vielen Stellen auf Formalitäten und feste Strukturen verzichtet werden, die in einem großen Team unbedingt erforderlich sind.
Das in der Automobilindustrie insbesondere bei der Entwicklung automobiler Steuergeräte verwendete prototypische und damit inkrementelle Vorgehen soll unterstützt und erweitert werden. Zum einen passt dies gut zur Vorgehensweise beim Kunden, zum anderen helfen Prototypen den Kundenkontakt zu verbessern, da sehr früh ein funktionierendes System zur Verfügung steht, um die Kundenanforderungen nicht nur zu verfizieren sondern auch zu evaluieren. Weiterhin ermöglichen Prototypen und frühe Muster, die an den Kunden gegeben werden, den Zeitplan im Projekt zu entschärfen, da der Kunde schon früh Muster erhält, um die eigene Entwicklung

voranzutreiben. Das heißt, ein Prozess der die Erstellung von Prototypen fördert, hilft kleinen Unternehmen den Zeitdruck bei Entwicklungsprojekten zu verringern, der bei klassischen Prozessen möglicherweise durch die kleine Teamgröße entsteht.
Bei der Definition des Software-Entwicklungsprozesses sollte zudem eine strukturierte Architekturbewertung einbezogen werden, da durch diese vor allem die nichtfunktionalen Eigenschaften von Systemen stärker beachtet werden. Diese Eigenschaften können im Wesentlichen der Wartbarkeit zugeordnet werden (vgl. Abbildung 2.3). Die Architekturbewertung hilft demzufolge insbesondere auch kleinen Unternehmen die Abhängigkeit von einzelnen Mitarbeitern zu minimieren und zukunftsfähige Software zu erstellen.
Das Projektmanagement in kleinen Unternehmen sollte ebenfalls „leichtgewichtig“ aufgesetzt werden, um die Minimalanforderung der Budget- und Ressourcenplanung sowie der Termin- und Budgetkontrolle zu gewährleisten. Das Vorgehen im Projekt sollte sich nur an wenigen Punkten am Prozess orientieren müssen und vielmehr ergebnis- und risikoorientiert erfolgen, um auch hier den Vorteil der Flexibilität und Reaktionsgeschwindigkeit beizubehalten.
Aus oben genannten Gründen sollten sich kleine und mittelständische Unternehmen bei der Einführung eines standardisierten Entwicklungsprozesses an allgemein anerkannten Prozessen und Prozessbewertungsmethoden orientieren. Ein relativ allgemeiner aber praxisorientierter Entwicklungsprozess wird in der VDI Richtline 2206 [65] *„Entwicklungsmethodik für mechatronische Systeme“* beschrieben.
Insbesondere in Deutschland hat sich aufgrund der Festlegung der Herstellerinitiative Software (HIS) [23] Automotive SPICE (vgl. Abschnitt 2.7.3) als Prozessbewertungsmethode zum de-facto Standard in der Lieferantenbewertung der Automobilindustrie durchgesetzt. Daher sollten sich auch kleine Unternehmen bei Ihrer Prozessdefinition an diesem Bewertungsschema orientieren. Die im Folgenden beschriebenen grundlegenden Prozessbestandteile werden in dieses Bewertungsschema eingeordnet.
Zusammenfassend kann ein entsprechender Entwicklungsprozess wie in Abbildung 3.1 dargestellt werden. Als Notoation wird hier die *Business Process Model and Notation (BPMN)* verwendet, die in Abschnitt 2.7.4 eingeführt wurde. Ein Projekt beginnt zunächst mit dem Anforderungserfassungsprozess, der gemeinsam mit dem Kunden durchgeführt wird. Im Ergebnis wird eine Anforderungsbaseline definiert, die gleichzeitig als Basis für die Konzepterstellung und als Basis für den Integrationstest beim Kunden dient. Im Konzepterstellungsprozess wird die Softwarespezifikation erstellt, die gleichzeitig als Basis für die Softwareerstellung und für den Systemtest dient. Das Konzept als Prozessergebniss wird mit dem Kunden abgestimmt. Wichtige Unterstützungsprozesse sind sind der Review-Prozess, das Projektmanagement und der Wiederverwendungsprozess. Die einzelnen Teilprozesse werden im Folgenden detailliert beschrieben.

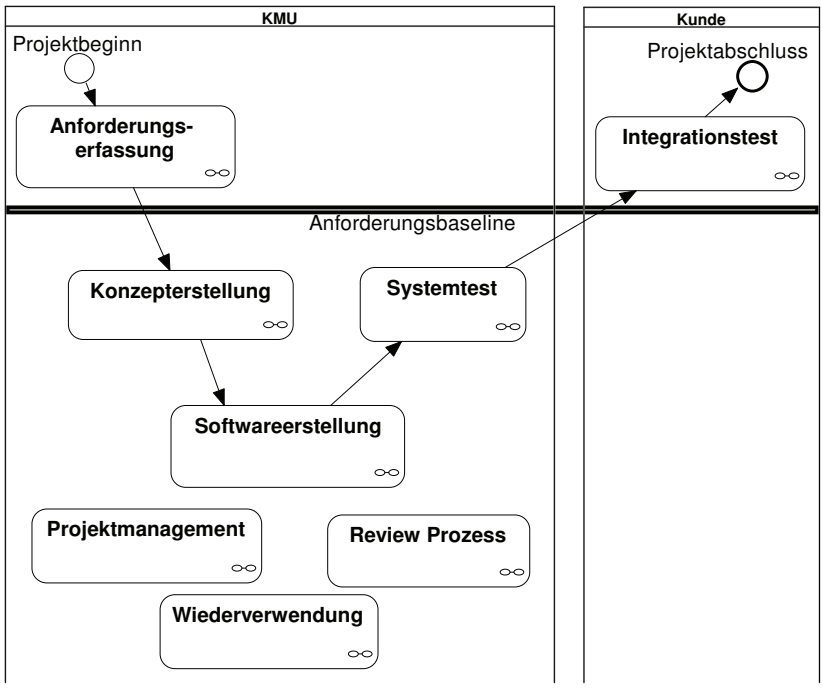

Abbildung 3.1: *Ansatz eines Softwareentwicklungsprozesses für kleine und mittelständische Unternehmen*

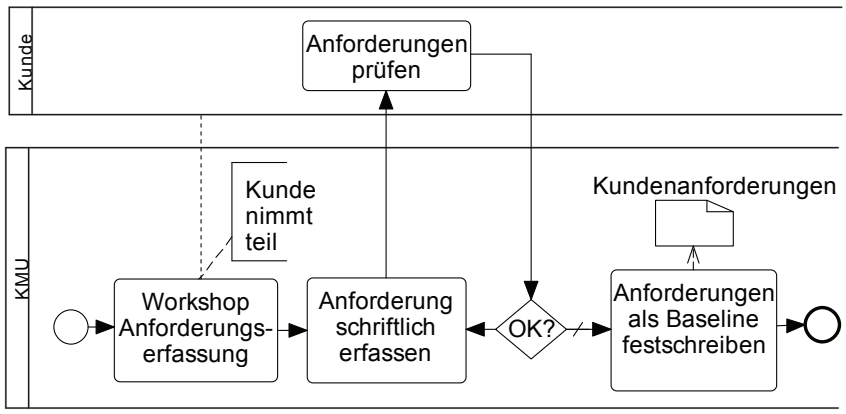

Abbildung 3.2: *Prozess zur Anforderungserfassung für KMUs*

3.1 Anforderungserfassung

Die Herstellerinitiative Software schreibt für die Lieferantenbewertung eine Untermenge an Prozessen (HIS Process Scope, [22]) zwingend vor, die zu bewerten sind. Aus der *Engineering Process Group* sind dabei bis auf ENG.1 (*Requirements elicitation*) alle Prozesse enthalten. Der Zweck dieses Prozesses ENG.1 ist in der Automotive SPICE Spezifikation wie folgt definiert:

> „Der Zweck des Anforderungserhebungs Prozesses besteht darin, die während der Nutzungsdauer eines Produkts und/oder einer Dienstleistung entstehenden Kundenbedürfnisse und -anforderungen zu erheben, zu bearbeiten und nachzuverfolgen, um so eine Anforderungsbaseline zu erstellen, die als Basis für die Definition der benötigten Arbeitsprodukte dient.“ [63]

Es ist es etwas verwunderlich, dass die HIS ausgerechnet diesen Prozess aus Ihrem Bewertungsschema ausklammert, da dieser Punkt insbesondere für das professionelle Kunde-Lieferanten-Verhältnis entscheidend ist. Außerdem bauen die nachfolgenden Prozesse ENG.2-ENG.5, in denen Anforderungen an das zu entwickelnde System definiert werden, auf den Ergebnisprodukten des ersten Prozesses ENG.1 auf. Insbesondere kleine Unternehmen sollten jedoch darauf achten, diesen Prozess zu implementieren, um die starke Kundenorientierung durch eine umfangreiche Einbindung des Kundenwunsches zu unterstützen.
Ein Implementierung des Prozesses Anforderungserfassung kann wie in in Abbildung 3.2 dargestellt aussehen. Für ein erfolgreiches Projekt ist es entscheidend ein gemeinsames Problemverständnis mit dem Kunden aufzubauen. Dies fällt kleinen Unternehmen in der Regel leichter durch die kleinere Teamgröße und der damit

verbundenen direkten Kommunikation der einzelnen Mitarbeiter untereinander. Um diesen Aspekt auch in der Anforderungserfassung zu unterstützen wird als erste Aktion im Prozess ein Workshop gemeinsam mit dem Kunden und den wesentlichen Projektteilnehmern durchgeführt. In diesem Workshop nennt der Kunde seine Anforderung und bekommt eine direkte Rückmeldung der beteiligten Projektteilnehmer. Die Praxis zeigt, dass der Kunde seine Anforderungen oftmals abschwächt, wenn dies zu hohem Projektaufwand führt. Auf der anderen Seite bringen die Projektteilnehmer auch häufig neue Aspekte durch ihren Erfahrungshorizont mit in die Diskussion ein, was häufig zu neuen Anforderungen führt, aber die Qualität des Projektergebnisses deutlich steigert. Im Anschluss an den Workshop werden die Anforderungen schriftlich fixiert und an den Kunden zur Prüfung weitergereicht. Mit den Änderungen nach Kundenwunsch und dessen Freigabe wird das Anforderungsdokument als Baseline im Sinne von Automotive SPICE (ENG.1) festgeschrieben. Dieses so erarbeitete Anforderungsdokument entspricht dem Konzept der Anforderungs Baseline (Requirements Baseline, vgl. Abbildung 2.6) im V-Modell nach Barry Boehm[1]. Diese Trennlinie wurde in den nachfolgenden, aufwändigen, im Auftrag der deutschen Bundesbehörden entwickelten, V-Modellen bis hin zum V-Modell XT nicht mehr explizit benannt (siehe auch Abschnitt 2.7.1). Im Umfeld eines kleinen Unternehmens ist dieser Ansatz jedoch nach wie vor sehr zweckmäßig, zumal er hier auch die Schnittstelle zwischen Kunde und Lieferant markiert.

3.2 Konzepterstellung

In einem weiteren Schritt werden aus den Kundenanforderungen zunächst Systemanforderungen an das zu erstellende System abgeleitet (vgl. Abbildung 3.3). Auf Basis dieser Anforderungen wird dann eine Systemarchitektur festgelegt. Nach der Dekomposition der Systemarchitektur werden sowohl für die Software als auch für die Hardware Anforderungen festgelegt. Auf diesen Anforderungen aufbauend werden anschließend für Software und Hardware eine Architektur und ein Design erstellt. Sowohl die Anforderungen als auch die Architekturbeschriebungen werden in einer Gesamtkonzeptbeschreibung festgehalten, die in einem Review-Prozess gemeinsam mit dem Kunden überprüft wird.

In Abbildung 3.3 ist ein integrierter Prozess für System, Hardware und Software dargestellt. Das wesentliche Ziel dieses Prozesses ist es systematisch und transparent aus den Kundenanforderungen die Teilanforderungen abzuleiten, um auf deren Basis die Architektur und das Design festzulegen. Zur expliziten Verdeutlichung werden in der Gesamtkonzeptbeschreibung Tabellen (Traceability Matrices) angelegt, die aufzeigen auf welche Kundenanforderung ein Element zurückzuführen ist. Je nach Kunde werden die Anforderunge und die entsprechenden Tabellen auch mit Hilfe

[1] Die gilt auch begrifflich.

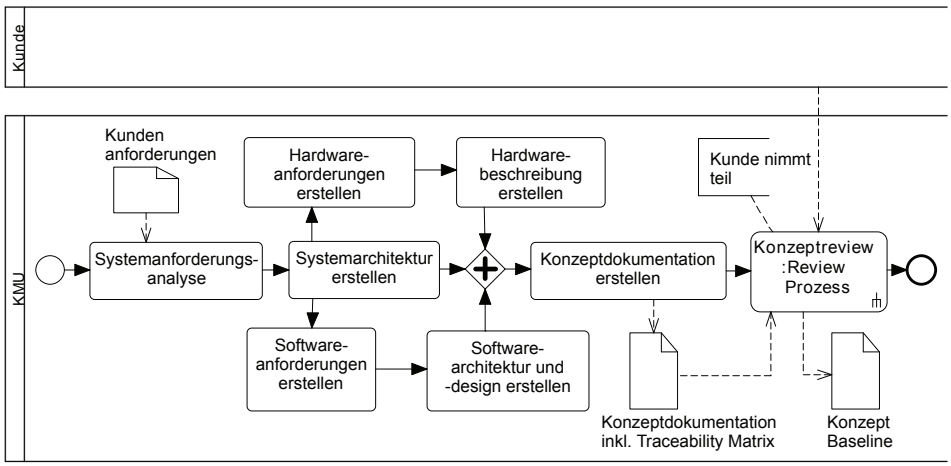

Abbildung 3.3: *Prozess zur Erstellung eines integrierten Konzepts. Dies umfasst für System, Hardware und Software, die Anforderungen sowie die Architektur- und Designbeschreibung*

eines Werkzeugs, wie z.B. IBM DOORS, verwaltet. Die Teilanforderungen für System, Hardware aund Software können z.B. mit Hilfe eines Workshops erarbeitet und dokumentiert werden. Die anschließende Dokumentation der Architektur und des Designs erfolgt idealerweise unter anderem in grafischer Form, indem z.B. UML Diagramme für die Software verwendet werden.
Für die Qualität des erarbeiteten Konzepts ist das Konzeptreview, bei dem auch der Kunde einbezogen wird, von wesentlicher Bedeutung. Hierfür wurde ein eigener Review-Prozess (s.u.) entworfen, der hier auf das Review-Objekt Konzeptbeschreibung angewendet wird. Im Ergebnis wird die Konzeptbeschreibung als Konzept-Baseline festgeschrieben.

3.3 Softwareerstellung

Die Steuergeräte Entwicklung im Automobilbereich erfolgt in der Regel prototypisch anhand von A-, B- und C-Mustern (vgl. Abschnitt 2.6). Dem entsprechend wird auch die Softwareerstellung[2] gestaltet. Projekte oder Teilprojekte, die im Umfang eines kleinen Unternehmens üblicherweise durchgeführt werden, erstellen im Ergebnis in der Regel eines dieser Muster. Der Prozess, der dafür definiert wird, kann universell auf alle Musterstände angewendet werden. Die jeweilige Ausprägung der einzelnen Teilaktionen ist jedoch naturgemäß deutlich unterschiedlich. So steht die Verifikation

[2]Die Erstellung von Hardware und System auf Basis des Konzepts wird in dieser Arbeit nicht weiter beschrieben. Sie erfolgt aber zeitlich parallel und systematisch analog zur Softwareerstellung.

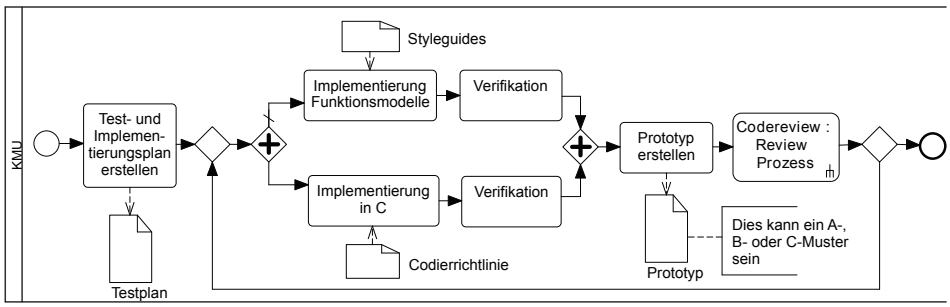

Abbildung 3.4: *Prozess zur Softwareerstellung, wie er für A-, B- und C-Muster durchgeführt wird.*

zum Beispiel bei einem B- oder C-Muster deutlich stärker im Vordergrund als dies bei einem A-Muster der Fall sein wird.
Zudem bietet bietet die modellbasierte Softwareentwicklung insbesondere für kleine Unternehmen ein großes Potenzial, da die Qualität der Software unabhängig von einzelnen Entwicklern sichergestellt werden kann. Reglermodelle können beispielsweise in Modellierungswerkzeugen wie z.B. Matlab/Simulink erstellt werden. Aus diesen Modellen wird mit Hilfe von Code-Generatoren automatisch C-Code erstellt. Die qualitative Reproduzierbarkeit des Codes ist somit gegeben und ist somit unabhängig von einzelnen Entwicklern. Die Modelle sollten entsprechend der prototypischen Entwicklungsweise inkrementell aufgebaut werden, indem Teile zunächst als funktionslose Blöcke angelegt werden und in einem späteren Release mit der gewünschten Funktion hinterlegt werden. Weiterhin können Funktionsteile bereits während der Entwicklung in der Simulation sehr einfach verifiziert werden, was einen White-Box-Test auf der C-Code Ebene in der Regel ersetzt.
Daher ist die modellbasierte Entwicklung integraler Bestandteil des Softwareerstellungsprozesses, wie er in Abbildung 3.4 dargestellt ist. Im ersten Schritt wird auf Basis der Architektur und des Designs ein Testplan für die einzelnen Software- und Modellteile erstellt. Anschließend werden die Software- und Modellteile unter Beachtung von Styleguides für die Modellierung und Codierrichtlichtlinien für die C-Codeerstellung implementiert. Die einzelnen Teile werden zusammengeführt und bilden die Software für einen Prototypen, der ein A-, B- oder C-Muster sein kann. Um eine hohe Codequalität zu garantieren wird der Code in Code-Reviews gesichtet und überarbeitet. Dafür wird der Review Prozess (s.u.) auf das Review Objekt Softwarecode angewendet.

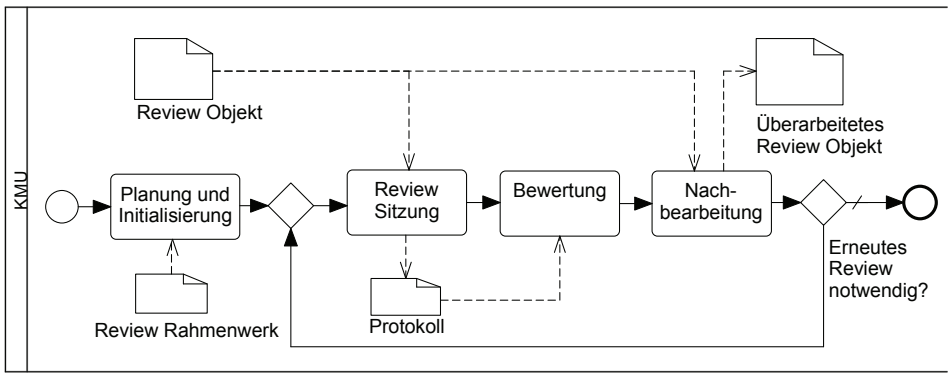

Abbildung 3.5: *Review Prozess, der auf unterschiedliche Review-Objekte angewendet werden kann*

3.4 Reviews

In Abschnitt 2.5 wurde bereits die hohe Software-Qualität erwähnt, die durch Reviews und Architekturbewertung zu erreichen ist. Insbesondere berücksichtigt diese Methode neben der Überprüfung der rein funktionalen Anforderungen auch die nicht-funktionalen Anforderungen. Diese sind in der Regel durch Softwaretests nicht zu überprüfen. Des Weiteren bieten Reviews die Chance, dass Experten ihre jeweilige Erfahrung strukturiert mit einbringen. Dies bietet vor allem kleinen Unternehmen die Möglichkeit, auch mit wenig Personal eine hochqualitative Software zu erstellen. Vor der eigentlichen Reviewsitzung wird der Reviewverlauf zunächst geplant und vorbereitet (vgl. Abbildung 3.5). Dabei wird das allgemeine Review Rahmenwerk auf das Reviewobjekt angepasst indem vor allem die passende Bewertungsmethode ausgewählt wird. Anschließend erfolgt die eigentliche Review Sitzung bei der das Review Objekt inspiziert wird. Die Rollenverteilung ist hierbei unabhängig vom Review-Objekt. Im Anschluss an die Review-Sitzung wird das Review Objekt bewertet und es wird ein Änderungsplan für die Nachbearbeitung beschlossen. Je nach Grad der notwendigen Änderungen wird ein erneutes Review angesetzt oder nicht. Der Ansatz auf Basis des Review Rahmenwerks wurde in [61] erarbeitet und speziell auf kleine und mittelständische Unternehmen zugeschnitten. Dabei sind bestimmte Faktoren zu berücksichtigen, wie z.B. die Erfahrung mit Reviews oder die Anzahl der Experten in diesem Unternehmen. In Folgenden wird davon ausgegangen, das im jeweiligen Unternehmen jeweils einige wenige Personen an einem Review teilnehmen, die aber nach dem Grad ihrer Ausbildung und ihrer Erfahrung als Experten einzustufen sind.
Aufgrund der Unternehmensstruktur wurde ein Ansatz entwickelt, der die Reviewtechnik „leichtgewichtig“ einführt und dabei soziale, sogenannte *weiche* Aspekte

berücksichtigt. Das heißt, es wird nicht versucht eine komplexe Variante des Reviews, wie z.B. die *Design and Code Inspection* von Michael Fagan (IBM) [17], umzusetzen oder einen *ATAM*-Workshop (siehe dazu auch Abschnitt 2.5.3) durchzuführen. Vielmehr geht es darum einen konzeptionellen Rahmen zu schaffen, in den eine kontextabhängige Prüfmethode eingebettet werden kann. Dazu werden generell einzuhaltende Regeln, ein einheitlicher Ablauf und verschiedene Verantwortlichkeiten eingeführt. Ein Augenmerk liegt auch auf den sozialen Auswirkungen des Reviews. Die Einführung eines Reviews stellt eine weitaus größere Veränderung für die Mitarbeiter dar als z.B. die Einführung eines neuen Werkzeugs, da sie die aktuelle Arbeit der Mitarbeiter offenlegt, begutachtet und kritisiert [41]. Daher müssen auch Verhaltensregeln berücksichtigt werden.
In Abbildung 3.6 ist das Prinzip des Review Rahmenwerkes dargestellt. Die Organisation des Reviews stellt den „Rahmen“ dar. Dieser Rahmen ist starr und die organisatorischen Komponenten wie der Ablauf, die Rollen und Regeln sind festgelegt. Abhängig von dem zu prüfenden Dokument wird in diesen Rahmen die entsprechende Prüfmethode integriert. Die Vorteile dieses Rahmenwerkes sind:

- Der generelle Ablauf des Reviews fördert die Vertrautheit.
- Mit jeder Anwendung steigt die Erfahrung und Routine, was in einer Erhöhung der Effizienz und der Qualität des Reviews mündet.
- Die Richtlinien und Regeln werden einheitlich gelebt.
- Das Rahmenwerk ist leicht erweiterbar. Detailliertere Reviewtechniken sind einfach zu integrieren.

Zu einer strukturierten Durchführung, wird das Review in einen Ablauf mit verschiedenen Phasen gegliedert. Für die Durchführung des Reviews ist das Review Team zuständig. Den Mitgliedern des Review Teams werden verschiedene Rollen zugeteilt. Das heißt sie übernehmen in dem Review-Prozess bestimmte Verantwortlichkeiten und Zuständigkeiten.

3.5 Projektmanagement

Das Projektmanagement ist für kleine Unternehmen ebenso wichtig wie für große Entwicklerteams. Aufgrund der Teamgröße kann es jedoch weniger aufwändig gestaltet werden. Automotive SPICE definiert das Projektmanagement im Prozess MAN.3. Neben der Projektplanung und der Kostennachverfolgung während des Projekts ist es insbesondere für kleine Unternehmen entscheidend, Risiken im Projekt frühzeitig aufzudecken und zu bearbeiten, da bei einer Nichtbeachtung oftmals eine große Projektverzögerung entstehen kann, da kleine Unternehmen oftmals nicht die

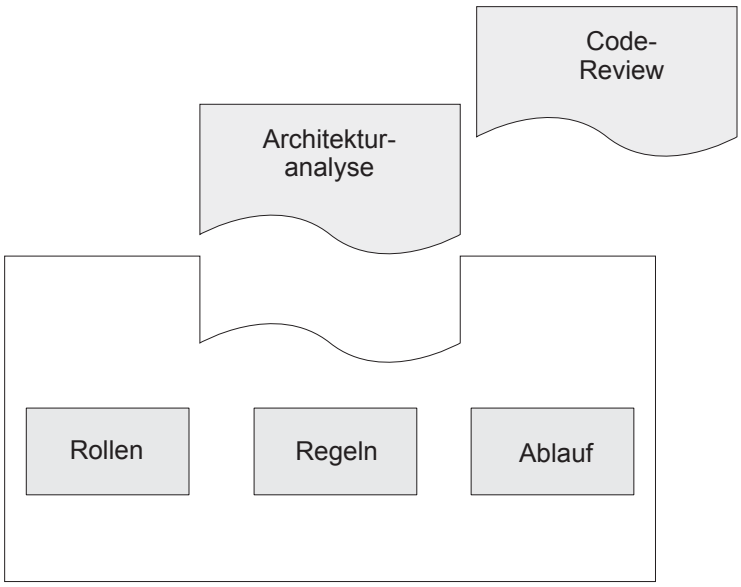

Abbildung 3.6: *Prinzipielle Darstellung des Review-Rahmenwerkes [61]*

Möglichkeit haben, bei Problemen dem Projekt mehr Ressourcen zur Verfügung zu stellen. Unterstützend bietet es sich daher an auch den Prozess MAN.5 Risikomanagement (jedoch wenig formal) umzusetzen.
Weiterhin sollte ein System eingerichtet werden, das Projektaufgaben verwaltet. Dies kann z.B. in Form eines Backlog (vgl. Abschnitt 2.7.2, Scrum) geschehen. In Automotive SPICE wird dies durch den Problemlösungsprozess SUP.9 beschrieben.

3.6 Wiederverwendung

Insbesondere für kleine Unternehmen kann Wiederverwendbarkeit von Code ein entscheidender Marktvorteil sein. Oftmals bestehen in der Architektur viele Freiheiten. Hier bietet es sich an, eine bekannte Architektur wiederzuverwenden und auf bewährte, getestete Softwarekomponenten zurückzugreifen. Die modellbasierte Entwicklung unterstützt diesen Anspruch ebenfalls, da sie Module auf einem relativ hohen Abstraktionsgrad definiert. Für diese kann bei einer sehr bewussten Wahl der Schnittstellen eine gute Wiederverwendbarkeit erreicht werden. Grundvoraussetzung dafür ist aber ein funktionierendes Versions- und Konfigurationsmanagement sowie eine gute Dokumentation. Automotive SPICE definiert das Wiederverwendungsmanagement im Prozess REU.2, der ebenfalls nicht Teil des Scopes der Herstellerinitiative Software (HIS) ist.

4 Ein Werkzeug für die modellbasierte Entwicklung

Software Werkzeuge unterstützen den in Kapitel 3 beschriebenen Ansatz für einen Softwareentwicklungsprozess. Besonders in der Softwareerstellung helfen Werkzeuge die Qualität zu steigern und die Entwicklung effizient durchzuführen. Das im Folgenden beschriebene Entwicklungswerkzeug wird dafür eingesetzt automobile Steuergeräte zu entwickeln. Es integriert die dafür notwendigen externen Softwarewerkzeuge und stellt dem Entwickler gleichzeitig eine Rahmenarchitektur für die zu entwickelnde Steuergerätesoftware zur Verfügung. Diese wird von der verwendeten Ziel-Hardware weitgehend abstrahiert, um eine gute Wiederverwendung in allen Phasen der Entwicklung zu gewährleisten.
Die beschriebene Werkzeugkette eignet sich neben der Verwendung in kleinen und mittelständischen Unternehmen auch für den Einsatz in großen Unternehmen. Der besondere Fokus auf Anpassbarkeit und definierte Software-Schnittstellen eröffnet jedoch insbesondere kleinen Unternehmen die Möglichkeit sich in große Entwicklungsvorhaben zu integrieren

4.1 Anforderungen

Um die zuvor genannten Ziele zu erreichen, können Anforderungen aufgestellt werden, die im Folgenden detailliert dargestellt werden. Dabei lassen sich die einzelnen Anforderungen entsprechend des Qualitätenbaums (vgl. Abschnitt 2.2 und Abbildung 2.3) den Qualitäten Brauchbarkeit und Wartbarkeit zuordnen. Zudem werden sie entsprechend der Qualitäteneinteilung zur besseren Nachverfolgbarkeit nummeriert. Die Zuordnung im Qualitätenbaum ist im Anschluss an die Beschreibung in Tabelle 4.1 dargestellt.

B11 Modellbasierte Entwicklung wird unterstützt
Um insbesondere die Flexibilität und die Reaktionsgeschwindigkeit von kleinen Unternehmen zu sichern und zu fördern, eignet sich eine modellbasierte Softwareentwicklung sehr, da aus den Modellen mit Hilfe von Code-Generatoren automatisch - und damit schnell und reproduzierbar- C-Code erzeugt wird. Gleichzeitig bedeutet dies eine Steigerung der Codequalität, unabhängig vom Entwickler, wenn der Code automatisch durch einen qualifizierten Code-Generator erzeugt wird. Die

gegenüber der direkten C-Code Programmierung eingeschränkten Möglichkeiten in einem Modellierungswerkzeug verhindern, dass Entwickler unerwünschte Konstrukte verwenden, da lediglich die Funktionen verwendet werden können, die das System anbietet. Modellbasierte Softwareerstellung unterstützt somit die Forderung nach größerer Unabhängigkeit von einzelnen Mitarbeitern.
Weiterhin ist die Erstellung von Prototypen während der Entwicklung erwünscht. Die modellbasierte Entwicklung von Software hilft zügig zu einem funktionierendem Prototyp zu gelangen. Weiterhin sollte die Entwicklung dieses Prototyps jeweils ein Inkrement in dem Entwicklungsprozess darstellen. Das bedeutet, dass der Prototyp kein *Wegwerfprototyp* sein sollte sondern ein *evolutionärer Prototyp*. Dies entspricht dem in der Automobilindustrie allgemein verwendeten Prinzip A-, B- und C-Muster (oder Prototypen) auf dem Weg zum Seriensteuergerät zu entwerfen. Die für die einzelnen Prototypen erstellten Modelle sollen für die jeweils nächste Stufe weitgehend wiederverwendet werden.

B12 Bibliothek von Sensoren und Aktuatoren
Um Modelle von der Simulationsphase bis zum Seriensystem konsistent zu halten, müssen die Schnittstellen zwischen dem Regler- und dem Streckenmodell gleich gehalten werden. Im realen Fall bilden Sensoren und Aktuatoren die Schnittstelle zwischen der realen Strecke (z.B. einem Verbrennungsmotor) und dem Steuergerät. Daher muss sich auch das Simulationsmodell an dieser Schnittstelle orientieren. Das Entwicklungswerkzeug sollte daher in der Modellierungsumgebung eine Abstraktion für jeden Sensor und Aktuator in Form einer Bibliothek anbieten.
Ein weiterer Aspekt ist es, die Modelle in der Modellierungsumgebung vor überflüssiger Komplexität zu bewahren. Soll beispielsweise der Sensorwert eines Temperatursensors in einem Regelungsmodell verwendet werden, muss z.B. definiert werden, über welchen Kanal des Analog-Digital-Wandlers das entsprechende Signal eingelesen werden soll. Diese Information ist aber hardwareabhängig und für die Simulation vollkommen überflüssig. Daher sollte diese Information nicht im Modellierungswerkzeug parametriert werden, sondern z.B. in einer separaten Konfigurationsdatei. Stellt nun das Entwicklungswerkzeug eine Bibliothek für verwendbare Sensoren und Aktuatoren zur Verfügung, so kann die Konfiguration des jeweiligen Sensors der Bibliothek zugeordnet werden, und das Modell wird von dieser Information „entlastet“.

B13 Offene Schnittstelle für User-Bibliotheken
Um das Werkzeug für eine möglichst große Anzahl von Anwendungen offen zu gestalten, soll eine offene Schnittstelle zur Verfügung gestellt werden, so dass Anwender eigene Bibliotheken erstellen und in das System einbinden können. Dies eröffnet zudem die Möglichkeit von neuen Geschäftsmodellen, indem z.B. von Dritt-Unternehmen Bibliotheken für das System angeboten werden.

B14 Verwendung des Werkzeugs als A-Muster
Die Bedeutung des A-Musters in der prototypisch geprägten Entwicklung automobiler Steuergeräte ist die eines Funktionsmusters. Das bedeutet, die Funktion des Steuergerätes muss vollständig gegeben sein, der mechanische und elektrische Aufbau unterscheidet sich aber noch vom Seriensystem. Der Hauptzweck des A-Musters ist es die Funktion des Gesamtsystems (reale Strecke mit Steuergerät) zu testen.
Wenn ein Entwicklungswerkzeug zur Verfügung steht, das die Funktion eines A-Musters erfüllen kann, so können die Entwicklungskosten für das A-Muster sehr niedrig gehalten werden. Es muss lediglich die Funktionssoftware erstellt werden, da die Hardware des Entwicklungswerkzeugs idealerweise bereits alle notwendigen Ein- und Ausgänge, sowie alle Basissoftwarekomponenten zur Verfügung stellt. Für die nächsten Phasen der Entwicklung können die weiteren Muster bis hin zum Seriensteuergerät aus dem Entwicklungswerkzeug abgeleitet werden. Dies ist insbesondere wirtschaftlich für Steuergeräte, die in relativ kleinen Stückzahlen produziert werden, da hier der Anteil der Entwicklungskosten an den Gesamtkosten hoch ist.

B21 Integration aller Softwarekomponenten in einer Bediensoftware
Um dem Kunden eine möglichst komfortable Entwicklungsumgebung zur Verfügung zu stellen, sollten möglichst alle Komponenten in einer Bediensoftware zusammengefasst werden. In der Bediensoftware sollten die notwendigen Softwareteile für Datenverwaltung, Kompilieren, Dateiexport, Datentransfer zur Hardware und Konfiguration integriert werden.

B31 Unterstützung eines qualifizierbaren Code-Generators
Für sicherheitskritische Applikationen ist die Verifikation von Softwaremodulen zwingend vorgeschrieben. Das Entwicklungswerkzeug sollte daher mindestens einen Code-Generator unterstützen, der qualifizierbaren Code erzeugt.

W11 Wiederverwendung von Architekturteilen
Insbesondere für kleine Unternehmen ist die Wiederverwendbarkeit von Teilen der Architektur und Code entscheidend, um Entwicklungskosten niedrig und Projektlaufzeiten kurz zu halten. Ein möglicher Ansatz ist es bei der Entwicklung eines Steuergerätes jeweils auf die gleiche Hardware- und Softwareplattform aufzusetzen, um so eine hohe Wiederverwendbarkeit zu erreichen. Dieser Ansatz wird in der Automobilindustrie bei mechanischen Komponenten wie z.B. Achsen, Fahrwerk, Motoren schon seit langem verfolgt. Das hier betrachtete Entwicklungswerkzeug sollte also eine Architektur anbieten, die als Plattform für unterschiedliche Projekte genutzt werden kann.

W12 Unabhängigkeit vom Modellierungswerkzeug
Unterschiedliche Modellierungswerkzeuge werden heute für die Modellerstellung und Code Generierung verwendet. Neben dem sicherlich am Weitesten verbreiteten Werkzeug Matlab/Simulink[1] finden auch Werkzeuge wie Ascet[2] oder Targetlink[3] ihre Verwendung. Das hier betrachtete Werkzeug sollte möglichst offen gestaltet sein, so dass neben Matlab/Simulink auch andere Software-Werkzeuge angekoppelt werden können, damit insbesondere kleine Unternehmen ihre Flexibilität erhalten und ausbauen können.

W13 Konfiguration und Softwarevarianten
Da das Entwicklungswerkzeug als A-Muster in vielfältigen Entwicklungsprojekten (vgl. Anforderung B14) dienen soll, müssen unterschiedliche Varianten angeboten werden können. Dies sollte die Architektur möglichst gut unterstützen. Weiterhin sollen die Module der Bibliothek konfigurierbar (vgl. Anforderung B12) sein. Diese Konfigurierbarkeit muss bei Entwicklung der Softwarearchitektur berücksichtigt werden.

W21 Einheitlicher Treiberaufbau
Um die Wartbarkeit zu unterstützen sollen Treiber und andere Bibliotheksfunktionen einen einheitlichen Aufbau besitzen. So ist es leicht neue Mitarbeiter an die vorhandene Struktur heranzuführen, um Änderungen durchzuführen. Des Weiteren verbessert es die Übersichtlichkeit und das Ausscheiden einzelner Mitarbeiter lässt sich leichter kompensieren.

W22 Weitgehende Verwendung von SW-Standards
Die Automobilindustrie arbeitet in vielfältigen Normungsgremien (z.B.: ASAM[4], HIS[5], AUTOSAR[6], OSEK[7]) an einheitlichen Softwarestandards und -schnittstellen. So können Low-Level-Treiber (wie z.B. Digital I/O oder A/D Wandler) beispielsweise nach dem HIS Standard aufgebaut werden. Für die Kommunikation bietet der ASAM Verein Protokolle wie CCP oder XCP [5], die auf dem CAN-Bus implementiert werden können. Neuere Entwicklungen sollten sich am AUTOSAR Standard orientieren, da dieser Standard andere Standards ablösen wird. Für das Betriebssystem gilt jedoch nach wie vor der OSEK Standard.

[1] The MathWorks, Inc.
[2] ETAS
[3] dSPACE
[4] http://www.asam.net
[5] http://www.automotive-his.de
[6] http://www.autosar.org
[7] http://www.osek-vdx.org

W23 Coding Standard für die Programmierung
Auch kleinere Unternehmen sollten einen Coding-Standard definieren, an dem sich die Mitarbeiter bei der Softwareerstellung orientieren müssen. Dieser Standard sollte auch allgemeine Richtlinien wie die MISRA[8] Regeln beachten. Bei der Erstellung von Matlab/Simulink-Modellen sollten Styleguides wie die des *The MathWorks Automotive Advisory Board (MAAB)*[9] beachtet werden.

W31 Unterstützung von Simulation
Die Verwendung von funktionsorientierter modellbasierter Entwicklung impliziert auch die Unterstützung von Simulation. Bei Definition und Implementierung der Softwarearchitektur für das Entwicklungswerkzeugs sollte auf eine möglichst realitätsnahe Simulation in allen Phasen der Entwicklung geachtet werden. Der erste Schritt ist die Model-in-the-loop (MIL) Phase, bei der das Regelungsmodell gegen das Streckenmodell im Simulationswerkzeug getestet wird. Auch hier sollte der Anwender bereits darin unterstützt werden, die Schnittstellen zwischen Regler und Strecke als Sensoren und Aktuatoren (vgl. Anforderung B12) zu definieren. Der Übergang zu den nächsten Stufen Software-in-the-loop (SIL) und Hardware-in-the-loop (HIL) fällt damit umso leichter. Im letzten Schritt wird die reale Strecke über reale Sensoren und Aktuatoren an das Steuergerät angebunden.

Die genannten Anforderungen lassen die sich anhand des Qualitätenbaums (vgl. Abschnitt 2.2 und Abbildung 2.3) einordnen. Sie sind in Tabelle 4.1 den unterschiedlichen Qualitäten zugeordnet dargestellt.

4.2 Architektur

Im Folgenden wird die Architektur des Entwicklungswerkzeugs beschrieben, das mit VeRa bezeichnet wird. VeRa ist ein Produkt der Fa. VEMAC, das in wesentlichen Teilen durch diese Arbeit entstanden ist.

4.2.1 Übersicht

Ausgehend von den Anforderungen an das Rapid-Control-Prototyping-System (vgl. 4.1) wurde ein System entwickelt, das sowohl von Kunden als auch von VEMAC genutzt werden kann, um Projekte durchzuführen. Abbildung 4.1 zeigt das grundlegende Use Case Diagramm des VeRa Systems. Dabei unterteilt sich das VeRa Gesamtsystem in die Teilkomponenten Hardware, Werkzeugkette (Tool-Chain, VeRa TC) und Externe Software.

[8] http://www.misra.org.uk/
[9] http://www.mathworks.de/automotive/standards/maab.html

Brauchbarkeit	Nützlichkeit	B11	Modellbasierte Entwicklung wird unterstützt
		B12	Bibliothek von Sensoren und Aktuatoren
		B13	Offene Schnittstelle für User-Bibliotheken
		B14	Verwendung des Werkzeugs als A-Muster
	Bedienbarkeit	B21	Integration aller Softwarekomponenten in einer Bediensoftware
	Verfügbarkeit	B31	Unterstützung eines qualifizierbaren Code-Generators
Wartbarkeit	Portabilität	W11	Wiederverwendung von Architekturteilen
		W12	Unabhängigkeit vom Modellierungswerkzeug
		W13	Konfiguration und Softwarevarianten
	Änderbarkeit	W21	Einheitlicher Treiberaufbau
		W22	Weitgehende Verwendung von SW-Standards
		W23	Coding Standard für die Programmierung
	Prüfbarkeit	W31	Unterstützung von Simulation

Tabelle 4.1: *Übersicht der Anforderungen an das Entwicklungswerkzeug eingeordnet in den Qualitätenbaum*

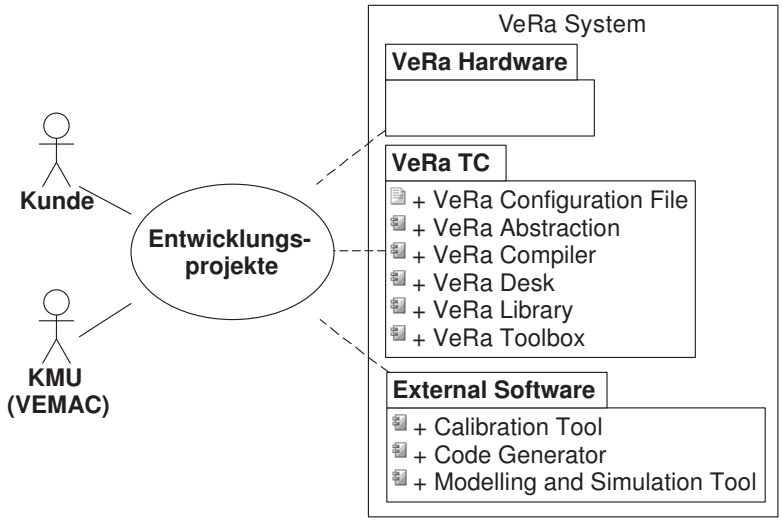

Abbildung 4.1: *Grundlegendes Use Case Diagramm*

Der Benutzer ist mit Hilfe des VeRa Gesamtsystems in der Lage alle Teilaufgaben zu erledigen, die notwendig sind um die digitale Steuerung und Regelung einer realen Strecke zu realisieren. Abbildung 4.2 zeigt das entsprechende Use Case Diagramm. Zunächst wird ein neues Reglermodell mit Hilfe von VeRa TC und dem Simulations- und Modellierungswerkzeug erstellt. Mit diesem Modell werden Simulationen durchgeführt, um die Funktionalität des Reglers nachzuweisen. Anschließend wird das Modell durch den Codegenerator in eine textbasierte Programmiersprache (Ansi C) übersetzt. Der so erzeugte Code wird kompiliert und in den physikalischen Flash-Speicher des Steuergerätes (Hardware) übertragen. Somit ist der Benutzer in der Lage das System zu nutzen um die reale Strecke zu regeln bzw. zu steuern. Mit Hilfe des Kalibrierwerkzeuges kann der Regler zur Laufzeit überwacht und angepasst werden. Dabei können Parameter und Kennfelder, nicht aber die Struktur des Reglers, verändert werden. Abbildung 4.3 zeigt ein Foto des Entwicklungssystems VeRa.

4.2.2 Hardware

Ausgehend von den Anforderungen an das Rapid-Control-Prototyping-System (vgl. 4.1) wurde die Hardware als seriennahes System entwickelt. Dies ermöglicht es, aus dem Prototypensystem mit geringem Aufwand ein Steuergerät abzuleiten, das auch in größeren Stückzahlen wirtschaftlich produziert werden kann (vgl.

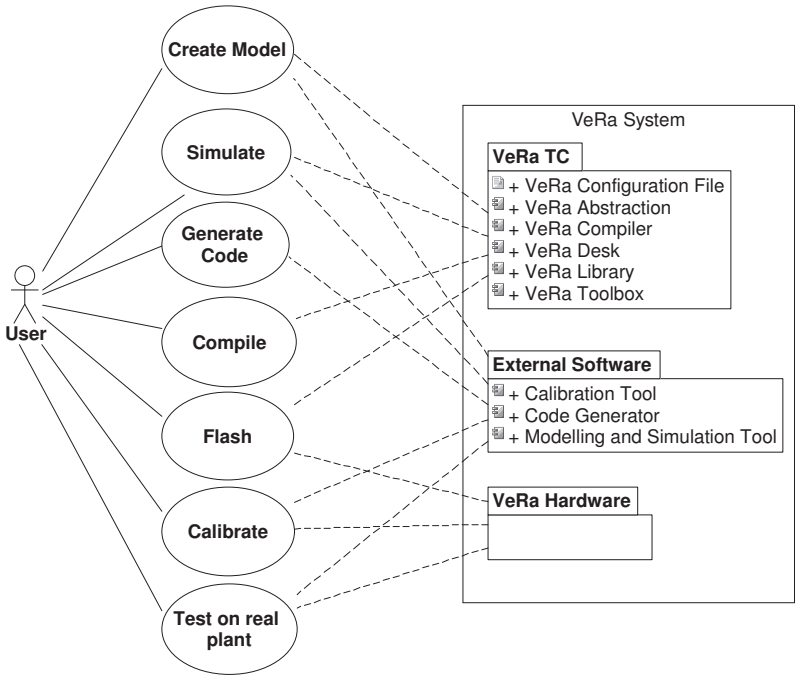

Abbildung 4.2: *Detailliertes Use Case Diagramm für die Projektarbeit, in der Steuerungen und Regelungen für reale Strecken realisiert werden*

Abbildung 4.3: *Entwicklungssystem VeRa bestehend aus Hardware und Software Toolchain*

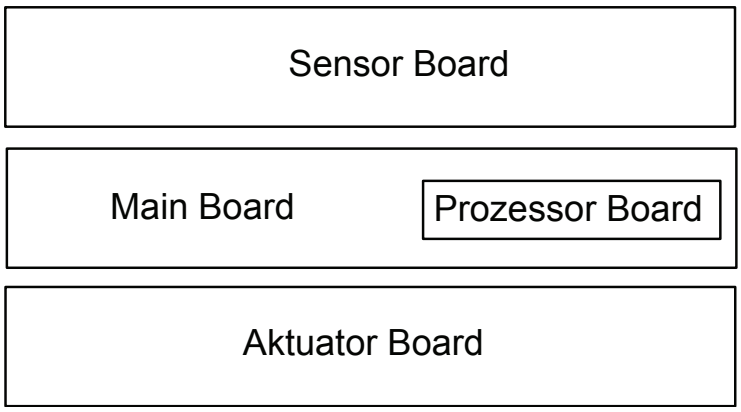

Abbildung 4.4: *VeRa Hardwareaufbau*

Anforderung B14). Folgende Merkmale kennzeichnen den seriennahen Aufbau:

- serientauglicher Prozessor
- serientaugliche Signalauswertung durch Bausteine, die auch heute schon in ähnlicher Anwendung in großer Serie verwendet werden
- serientaugliche Endstufen durch Bausteine, die auch heute schon in ähnlicher Anwendung in großer Serie verwendet werden
- serientaugliche Schaltungstopologie

Um aber gleichzeitig auch der geforderten Flexibilität Rechnung zu tragen, wurde ein modularer Aufbau gewählt. Dabei wurde das System in die drei Komponenten Aktuator Board, Main Board und Sensor Board unterteilt. Als vierte Komponente trägt das Main Board das Prozessormodul (vgl. Abbildung 4.4).

4.2.3 VeRa Tool-Chain

Das Deployment Diagramm des VeRa Systems zeigt die Aufteilung und die Verbindungen der unterschiedlichen Softwarekomponenten auf unterschiedlichen Plattformen (vgl. Abbildung 4.5). Auf dem Entwicklungs PC ist sowohl das Modellierungs- und Simulationswerkzeug als auch die grafische Benutzeroberfläche Vera Desk installiert. Als Kommunikationskanal zur VeRa Hardware-Plattform wird der CAN-Bus verwendet. Ebenfalls über den CAN-Bus ist das Kalibrierwerkzeug angekoppelt, das in der Regel auf einem separaten Rechner installiert ist. Diese Topologie trägt der Tatsache Rechnung, dass sich in der Automobilindustrie eine Trennung von

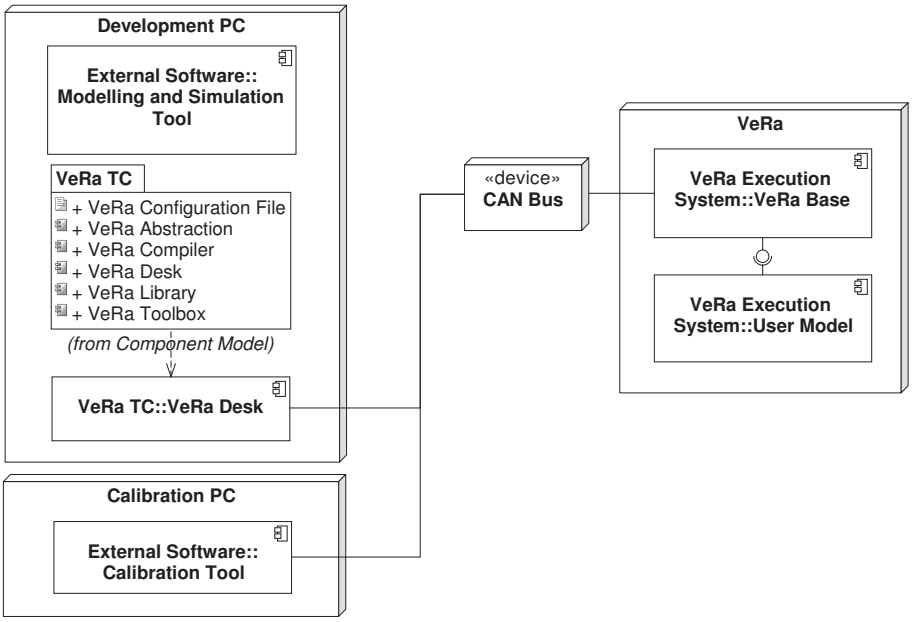

Abbildung 4.5: *Die unterschiedlichen Softwarekomponenten des VeRa Gesamtsystems sind auf mehrere Plattformen verteilt. Hier im Deployment Diagramm dargestellt*

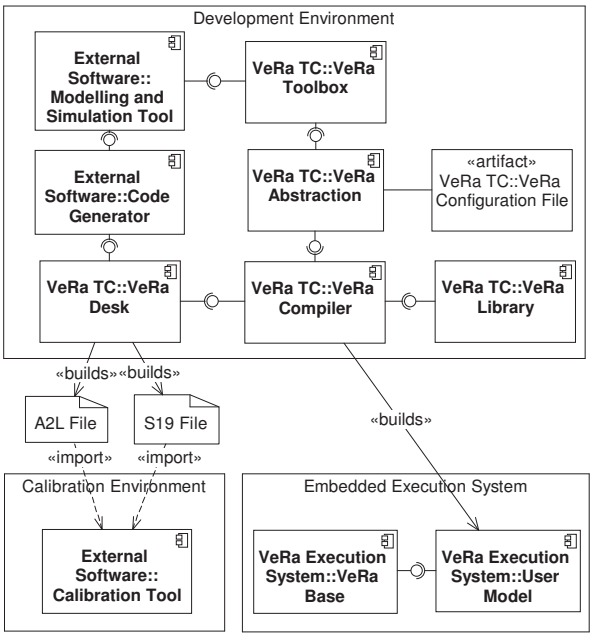

Abbildung 4.6: *Das VeRa System im Komponentendiagramm*

Funktionsentwicklung und Kalibrierung[10] (bzw. Applikation) durchgesetzt hat. Die Nutzung von standardisierten Schnittstellen und standardisierten Protokollen erlaubt dabei eine Kompatibilität der Produkte und Prozesse untereinander.

Das Komponentenmodell des VeRa Systems unterteilt sich in die Bereiche Entwicklungsumgebung, eingebettetes Ausführungssystem und Kalibrierumgebung (vgl. Abbildung 4.6). In der Entwicklungsumgebung stehen die Teile der VeRa Werkzeugkette (VeRa TC), das Modellierungs- und Simulationswerkzeug sowie der Code Generator in Beziehung zueinander. Im Modellierungswerkzeug steht dem Benutzer der Zugriff auf die Sensoren und Aktuatoren des realen Systems in Form einer Toolbox zur Verfügung. Die Elemente der Toolbox können in die Modelle übernommen und somit zunächst simuliert und anschließend auf dem Zielsystem verwendet werden. In der Simulation greift das System bei der Ausführung des Simulationsmodells auf die in der Abstraktionsschicht hinterlegten Sensor- und Aktuatormodelle zu und bindet diese in die Gesamtsimulation ein. Dabei wird die Systemkonfiguration berücksichtigt, indem die Beschreibung dieser Konfiguration in der Konfigurationsdatei im XML Format hinterlegt ist. Nachdem die Simulation erfolgreich abgeschlossen ist, wird mit Hilfe des Codegenerators C-Code erzeugt. Der erzeugte Code wird

[10] Hiermit wird die nachträgliche Veränderung von Modellparametern auf dem Zielsystem bezeichnet.

VeRa Hardware / VeRa Software Toolbox	GP	S6PI	S6DI	P6DI	individual
GP	X	X	X	X	X
Engine		X	X	X	(X)
Individual	(X)	(X)	(X)	(X)	(X)

Tabelle 4.2: *VeRa Varianten*

von VeRa Desk an den Compiler übergeben, der diesen zunächst kompiliert und anschließend mit Hilfe des Linkers zusammen mit den vorkompilierten Treiberfunktionen aus der VeRa Library zu einer ausführbaren Datei bindet. Die Konfiguration des Gesamtsystems wird bei diesem Vorgang ebenfalls berücksichtigt, indem aus der XML Konfigurationsdatei ein C-Modul generiert wird, das die notwendigen Informationen enthält.
Als Ergebnis des Kompiliervorganges wird die vom Benutzer generierte Funktionssoftware als Maschinencode erzeugt. Zusammen mit der VeRa Basissoftware bildet sie das eingebettete Ausführungssystems auf der Zielplattform. Zusätzlich zum Ausführungssystem werden aus dem generierten Code eine A2L[11]-Beschreibungsdatei und eine Binärdatei im S19[12]-Format erzeugt. Diese Dateien können von nahezu allen gängigen Kalibrierwerkzeugen importiert und weiterverarbeitet werden.

4.2.4 Varianten und Konfiguration

Die Anforderungen an das Rapid-Control-Prototyping-System (vgl. Kapitel 4.1, Anforderung W13) machen deutlich, das mehrere Varianten des Systems dargestellt werden müssen, um auf alle Kundenbedürfnisse individuell eingehen zu können. Diese Varianz wird zum einen durch unterschiedliche Hardware-Varianten und zum anderen durch Software-Varianten erreicht. Dabei ist es aus kaufmännischer Sichtweise wichtig, Standard-Varianten aber auch kundenindividuelle Varianten zu definieren (vgl. Tabelle 4.2). Aus technischer Sichtweise ist dies unerheblich. Bei der Hardware werden im Wesentlichen unterschiedliche Ein- und Ausgangskonfigurationen abgebildet. Diese werden auch durch den modularen Hardwareaufbau (vgl. 4.2.2) unterstützt. Auf der Softwareseite werden vor allem unterschiedliche Feature-Konfigurationen realisiert, die durch unterschiedliche Toolboxes im Modellierungswerkzeug zur Verfügung gestellt werden. Als Standard-Varianten wurden bei der Hardware neben einer allgemeinen Variante (VeRa GP (=General Purpose)) vor allem Varianten für den Anwendungsfall Motorsteuerung (VeRa S6PI (= Solenoid,

[11] Das Format dieser Beschreibungsdatei wurde durch die ASAM Vereinigung (www.asam.net) in der ASAM MCD-2 Norm festgelegt
[12] Das Format dieser Datei entspricht dem von Motorola in den 1970er Jahren eingeführten SREC-Format

6-Zylinder, port injection), VeRa S6DI (= Solenoid, 6-Zylinder, direct injection), VeRa P6DI(= Piezo, 6-Zylinder, direct injection)) definiert. Dem gegenüber stehen auf der Softwareseite Toolboxes für allgemeine Anwendungen (GP (= General Purpose)) und auch für den Anwendungsfall Motorsteuerung (Engine). Individuelle Hardware- und Software-Varianten wurden für verschiedene Kundenprojekte bereits ebenfalls realisiert.
Um mit diesen Varianten komfortabel umgehen zu können werden alle änderbaren Parameter in einer XML Konfigurationsdatei hinterlegt. Dies hat den Vorteil, dass bei einem Variantenwechsel lediglich die Konfigurationsdatei ausgetauscht bzw. angepasset werden muss. Der Aufbau der XML Dateien sieht wie folgt aus:

Beispiel: *XML Konfigurationsdatei eines Motordrehzahlsensors*

```
<Blocks>
 <Block Name ="Engine_Speed" Type="EngineSpeed">
  <TPL_ID>d7870dbb-06a1-40a8-887e-d099c4197c64</TPL_ID>
  <ID>98418b90-be62-4bb4bf71-91197b9d834e</ID>
  <FixName>false</FixName>
  <Subtype>Standard</Subtype>
  <Parameter Name="Subtype">
  <Value>Standard</Value>
  </Parameter>
  <Parameter Name="Sampletime" Datatype="INT16">
  <Value>0</Value>
  </Parameter>
  <Parameter Name="Stepsize" Datatype="UINT8">
  <Value>100</Value>
  </Parameter>
  <Parameter Name="Min_Value" Datatype="FLOAT32">
  <Value>0</Value>
  </Parameter>
  <Parameter Name="Max_Value" Datatype="FLOAT32">
  <Value>7000</Value>
  </Parameter>
  <Parameter Name="AngleToggle_Channel" Datatype="UINT8">
  <Value>2</Value>
  </Parameter>
  <Parameter Name="Spm_Channel" Datatype="UINT8">
  <Value>3</Value>
  </Parameter>
  <Parameter Name="TeethPerCycle" Datatype="UINT16">
```

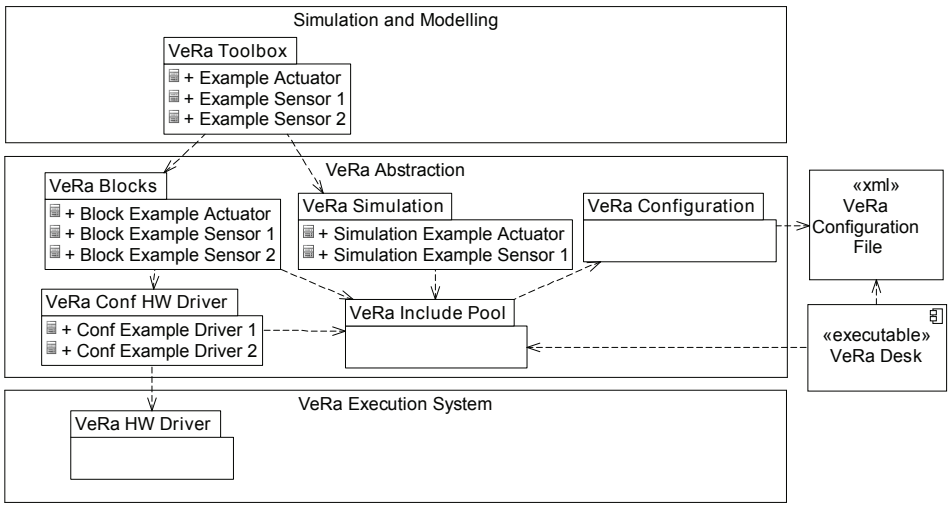

Abbildung 4.7: *Klassendiagramm der VeRa Abstraktionsschicht*

```
  <Value>120</Value>
  </Parameter>
  <Parameter Name="TPU" Datatype="UINT32">
  <Value>1</Value>
  </Parameter>
 </Block>
</Blocks>
    <!--ID==F1FF97E65C8712AE3B9C35D5CD52BE6-->
```

In Abbildung 4.7 ist das Klassendiagramm der VeRa Abstraktionsschicht dargestellt. Über den VeRa Include Pool und das VeRa Configuration Modul können alle anderen Module auf die XML-Konfigurationsdatei zugreifen. In ihr sind die Parametrierungen der Simulationsmodelle der Sensoren und Aktuatoren hinterlegt. Zusätzlich enthält die Konfigurationsdatei Informationen zur Hardwarekonfiguration für die an das reale Steuergerät angeschlossenen Sensoren und Aktuatoren.
Der Aufbau der XML-Konfigurationsdatei wurde oben am Beispiel Motordrehzahlsensor (`Engine_Speed`) gezeigt. Im Beispiel sind die Parameter `Sampletime` und `Stepsize` Beispielparameter, die für die Simulation benötigt werden. `AngleToggle_Channel` und `Spm_Channel` hingegen sind Beispiele für Hardwarekonfigurationsparameter. Für die Ausführung auf dem Zielsystem können die Parameter nicht direkt aus der XML-Datei gelesen werden, da hier in der Regel kein Dateisystem vorhanden ist. Daher werden von dem Anwenderprogramm VeRa Desk aus der XML-Datei automatisch eine Code- und eine Header-Datei erzeugt. Diese werden im Kompiliervorgang übersetzt und zum Anwendungsmodell hinzugebunden. Die

dem oben gezeigten Beispiel entsprechenden Dateien sehen wie folgt aus:

Beispiel: *vera_configuration.h*

```
/**
 * @brief generated vera_configuration Header-File
 *
 * @author <name>
 *
 * @file vera_configuration.h
 */
////////////////////////////////////////////////////////////////////
// DEFINITIONS AND MACROS //////////////////////////////////////////
////////////////////////////////////////////////////////////////////
#ifndef VERA_CONFIGURATION_H_INCLUDED
#define VERA_CONFIGURATION_H_INCLUDED
////////////////////////////////////////////////////////////////////
// TYPEDEFS AND STRUCTURES /////////////////////////////////////////
////////////////////////////////////////////////////////////////////
typedef struct
{
  INT16 Sampletime;
  UINT8 Stepsize;
  FLOAT32 Min_Value;
  FLOAT32 Max_Value;
  UINT8 AngleToggle_Channel;
  UINT8 Spm_Channel;
  UINT16 TeethPerCycle;
  UINT32 TPU;
}EngineSpeedType;
```

Beispiel: *vera_configuration.c*

```
/**
 * @brief generated vera_configuration C-File
 *
 * @author <name>
 *
 * @file vera_configuration.c
 */
////////////////////////////////////////////////////////////////////
// MODULES USED ////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////
#include <vera_configuration.h>
////////////////////////////////////////////////////////////////
// GLOBAL VARIABLES ////////////////////////////////////////////
////////////////////////////////////////////////////////////////
EngineSpeedType Engine_Speed = {
1,
100,
0,
7000,
2,
3,
120,
1
};
```

4.3 Modellierung und Simulation mit VeRa

Im Implementierungsteil des Entwicklungsprozesses steht der Modellentwurf mit Hilfe von Simulation an erster Stelle. Hier wird zunächst ein Modell der Strecke erstellt, die mit Hilfe des modellierten Reglers geregelt werden soll. Anschließend erfolgt der Reglerentwurf und der Regelkreis wird im Modellierungs- und Simulationswerkzeug geschlossen. Somit kann der geschlossene Regelkreis simuliert und die Reglerstruktur entwickelt werden, bevor das Steuergerät oder sogar die Strecke physikalisch vorliegt. Dieses Vorgehen wird in der Literatur mit *Systemsimulation* [1] oder *Model-In-the-Loop (MIL)* [63] bezeichnet.

4.3.1 Übersicht

Wesentlich für einen durchgängigen, modellbasierten Entwicklungsprozess ist eine eindeutige Schnittstelle zwischen Strecke und Regler. Diese muss sich notwendigerweise an der realen, physikalischen Schnittstelle orientieren, die durch Sensoren und Aktuatoren gebildet wird (vgl. Abschnitt 2.1.2). Somit müssen auch schon in der Modellierungsphase die Eingangswerte des Reglers durch simulierte Sensoren erzeugt und die Ausgangswerte des Reglers an simulierte Aktuatoren weitergeleitet werden. Diese scheinbar offensichtliche Schnittstelle wird in der Praxis oft nicht eingehalten, da während der Modellierungsphase die Möglichkeit besteht jede Zwischengröße aus der Streckenberechnung auch für die Regelung zu verwenden. Beim Einsatz an der realen Strecke stehen diese Größen jedoch nicht zur Verfügung und würden eine anschließende Umgestaltung des Reglers erfordern.
Um das reale Verhalten des Regelkreises bereits in der Simulationsphase möglichst

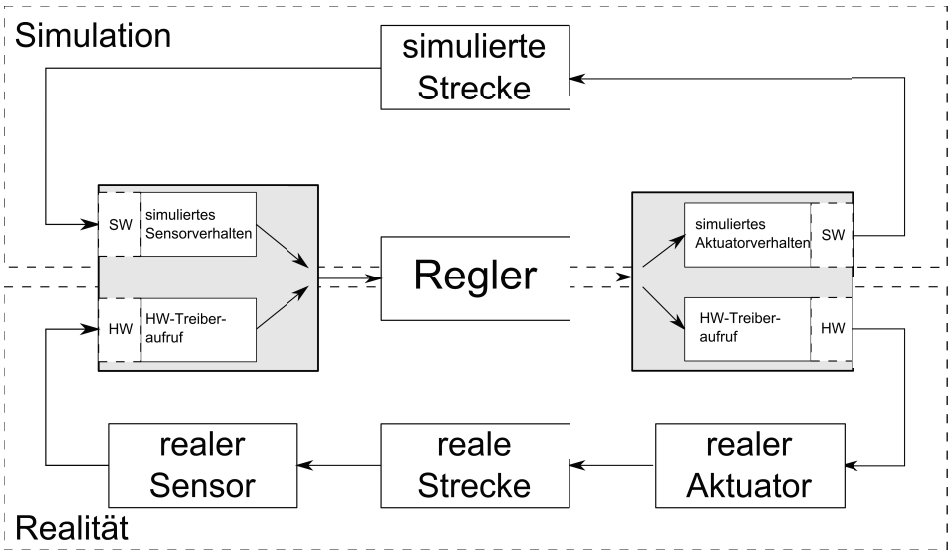

Abbildung 4.8: *Die Modellierungselemente Sensor und Aktuator enthalten für den Anwendungsfall der Simulation ein simuliertes Verhalten. Für den Fall der Codegenerierung werden über Hardware-Treiber die realen Sensorwerte eingelesen und die Aktuatorendstufen angesteuert.*

gut beurteilen zu können, wird bei der Simulation der Sensoren und Aktuatoren auch deren physikalisches Verhalten, wie z.B. Auflösung, zeitliche Verzögerung, Grenzen, berücksichtigt. Das Streckenmodell muss in der Simulation dafür dem Modell eines Sensors den Eingangswert in der physikalischen Einheit zur Verfügung stellen, die der Sensor misst. Der Ausgangswert eines Aktuators wird dem Streckenmodell in der Einheit zurückgeliefert, die er bei der Verstellung verwendet (vgl. Abbildung 4.8). Um den Regelkreis mit Hilfe der Codegenerierung später an die reale Strecke anzukoppeln, lesen die Modellierungselemente für die Sensoren über die Hardware-Treiber die realen Sensorwerte. Die Modellierungselemente für die Aktuatoren steuern ebenfalls über entsprechende Hardware-Treiber die Aktuatorendstufen an, um die realen Aktuatoren zu verstellen (vgl. Abbildung 4.8).

4.3.2 Realisierung in VeRa

Die VeRa Toolbox stellt im Modellierungswerkzeug die Benutzerschnittstelle des VeRa Systems dar. Die Toolbox enthält für jeden Sensor und für jeden Aktuator, die an die Hardware angeschlossen werden können, je einen Block, die in Anwendermodelle übernommen werden können. Gleichzeitig muss bei der Modellbildung darauf geachtet werden, dass mit Hilfe dieser Blöcke die Schnittstellen zwischen dem Reglermodell

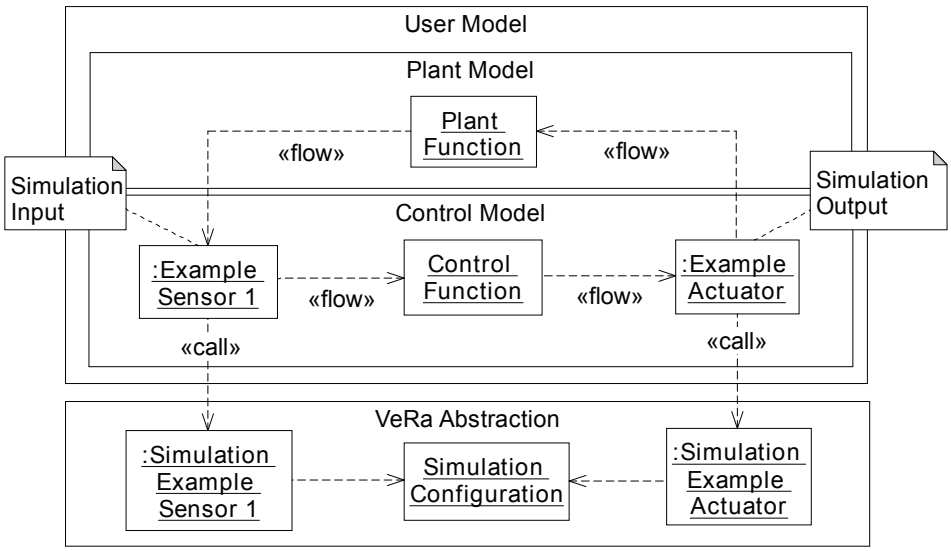

Abbildung 4.9: *Objektdiagramm für den Anwendungsfall Simulation*

und dem Streckenmodell gebildet werden (vgl. Abschnitt 4.3.1). Die Modellbausteine für Sensoren enthalten dafür spezielle Eingänge, über die das Streckenmodell den aktuellen simulierten Wert der vom Sensor zu messenden physikalischen Größe übergibt. Weiterhin enthalten die Modellbausteine für die Aktuatoren spezielle Ausgänge, die den aktuellen simulierten Wert der vom Aktuator zu stellenden physikalischen Größe an das Streckenmodell übergeben (vgl. Abbildung 4.9). Sobald einer der Sensor- oder Aktuatorblöcke während der Simulation des Anwendermodells ausgeführt wird, wird das Simulationsmodell des Sensors bzw. Aktuators als DLL Funktion aufgerufen. Dabei wird auch die jeweilige Konfiguration der Komponente beachtet indem die notwendigen Informationen aus der XML Konfigurationsdatei abgerufen werden. Dieser Aufruf ist in Abbildung 4.10 am Beispiel eines Sensors dargestellt. Bei der Initialisierung des Modells wird die Konfiguration der Komponente aus der XML Datei eingelesen und das Berechnungsmodell wird initialisiert. Bei Ausführung eines Berechnungsschrittes wird das Berechnungsmodell ausgeführt. Als Eingangswert verwendet es dafür den physikalischen Wert, der von dem realen Sensor gemessen wird und in der Simulation vom Streckenmodell an den Sensorblock übergeben wird. Als Ausgangswert liefert das Berechnungsmodell den Sensorwert in der gleichen Einheit wie der reale Sensor (z.B. analoge Spannung).

Um in der Simulation auch Nebenläufigkeitsprobleme erkennen und Fehler vermeiden zu können, wird die Aufteilung der Modellberechnung auf unterschiedliche Tasks auch in der Modellierungsumgebung definiert. Dafür steht eine spezielle Toolbox zur Verfügung, über die Betriebsmittel des OSEK Betriebssystems definiert und

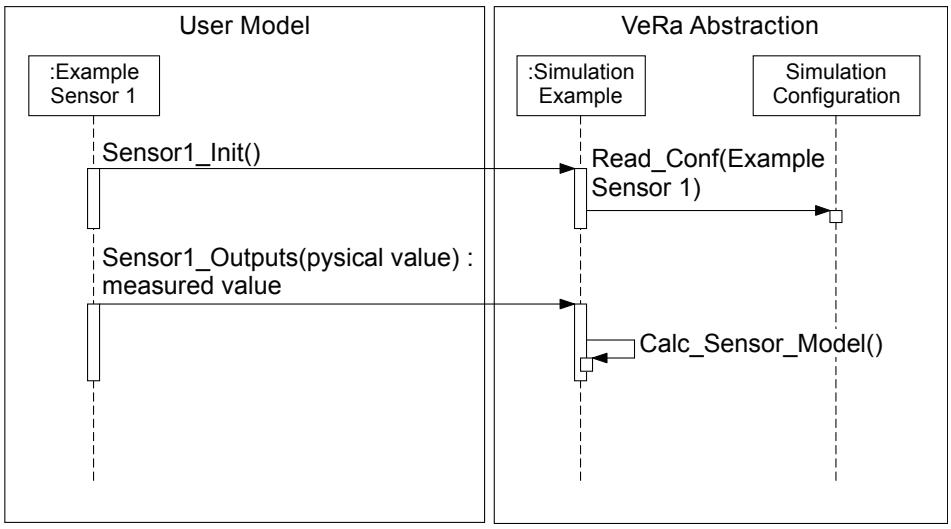

Abbildung 4.10: *Sensoraufruf im Anwendungsfall Simulation dargstellt im Sequenzdiagramm*

konfiguriert werden können. Es stehen Module für den Scheduler, den Taskaufruf, die Synchronisation über Events und Alarme sowie die Inter-Task-Kommunikation zur Verfügung.
Im Anwendungsfall der Simulation wird bei Verwendung der OSEK Toolbox auch das OSEK Betriebssystem simuliert. Dafür muss die zeitliche Schrittweite des Simulationsmodells kleiner als die niedrigste zeitliche Periode gesetzt werden. Bei jedem Berechnungsschritt wird nun der Scheduler aufgerufen und es wird genau wie auf dem Zielsystem der jeweils aktive Task nach der Scheduling Strategie ausgeführt. Um auch winkelbasierte Tasks berechnen zu können, wird dem Scheduler-Block der aktuelle Kurbelwinkel des Motors übergeben, um diese bei der Berechnung des aktiven Tasks zu berücksichtigen. Bei der Verwendung von Synchronisationsblöcken wird die Berechnung eines Tasks abgebrochen, wenn auf ein Betriebsmittel gewartet werden muss. Sobald das Betriebsmittel in einer der nächsten Berchnungsschritte wieder freigegeben wird, wird die Ausführung an der gleichen Stelle fortgesetzt, an der die Ausführung zuvor gestoppt wurde. Bei Verwendung der Kommunikationsblöcke werden die Inhalte der übertragenen Nachrichten und Werte bei der Berechnung der betroffenen Modellteile ebenfalls berücksichtigt.

4.4 Codegenerierung mit VeRa

Mit Hilfe eines Codegenerators wird aus dem in der Modellierungs- und Simulationsphase erstellten Modell Code für die Zielhardware erzeugt. Der Vorteil der automatischen Codegenerierung gegenüber der Codierung von Hand ist neben der Zeitersparnis vor allem in dem reproduzierbaren und qualitativ hochwertigen Code zu sehen. Die besondere Schwierigkeit einen Codegenerator für einen Anwendungsfall nutzbar zu machen, liegt in der Ankopplung an die Zielhardware. Mit dem VeRa System ist es gelungen den Aufwand für diese Ankopplung bei dem Austausch des Codegenerators zu minimieren. Dies wird vor allem durch die Separierung der VeRa-Toolbox vom Modellierungswerkzeug und durch Kapselung in eigenständige Module innerhalb der Abstraktionsschicht erreicht.

4.4.1 Übersicht

Nachdem in der Simulationsphase ein geeignetes Modell für den gesamten Regelkreis gefunden wurde, wird aus dem Reglerteil des Modells Code für die Zielhardware generiert. Dabei werden mit den verwendeten Sensor- und Aktuatorblöcken die Schnittstellen zwischen Regler und Strecke ebenfalls für die Codegenerierung berücksichtigt. Die Konfiguration des gesamten Systems ist in der XML-Konfiguration hinterlegt, so dass auch ein Austausch von Sensoren zu keiner Änderung des Modells führt.

4.4.2 Realisierung in VeRa

Für jeden Sensor- bzw. Aktuatoraufruf im Anwendermodell wird bei der Code Generierung der Aufruf eines entsprechenden Blockes der VeRa Abstraktionsschicht im Code erzeugt. Die entsprechenden Blöcke müssen auf die notwendigen Hardwareschnittstellen zugreifen um den Sensorwert auszulesen bzw. den Aktuatorsollwert zu übergeben. Komplexe Sensormodelle müssen dafür unter Umständen auf mehrere Hardwareschnittstellen zugreifen, indem die entsprechenden Treiber aufgerufen werden. Weiterhin ist es jedoch auch notwendig, dass mehrere Sensormodelle auf einen Treiber zugreifen. Beispielsweise wird der Treiber für den Analog/Digital-Wandler von vielen unterschiedlichen Sensoren verwendet (vgl. Abbildung 4.11). Um auf dem Zielsystem eine eindeutige Zuordnung der Treiberinformationen zu erreichen, wird die Kommunikation der Sensor- und Aktuatormodelle über ein Konfigurationsmodul gesteuert, das für jeden verwendeten Treiber vorhanden ist. Hier wird beispielsweise für den Analog/Digital-Wandler erfasst, welche Kanäle verwendet werden und welche nicht. Der Treiber kann so unter anderem kollisionsfrei verwendet werden. Weiterhin muss die Konfiguration der im Anwendermodell verwendeten Sensoren und Aktuatoren auch im Anwendungsfall der Code Generierung berücksichtigt

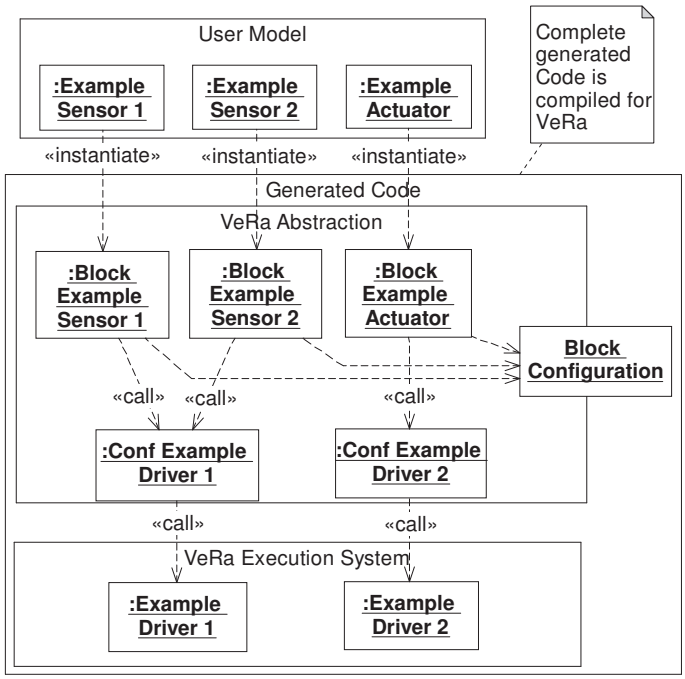

Abbildung 4.11: *Objektdiagramm für den Anwendungsfall Code Generierung*

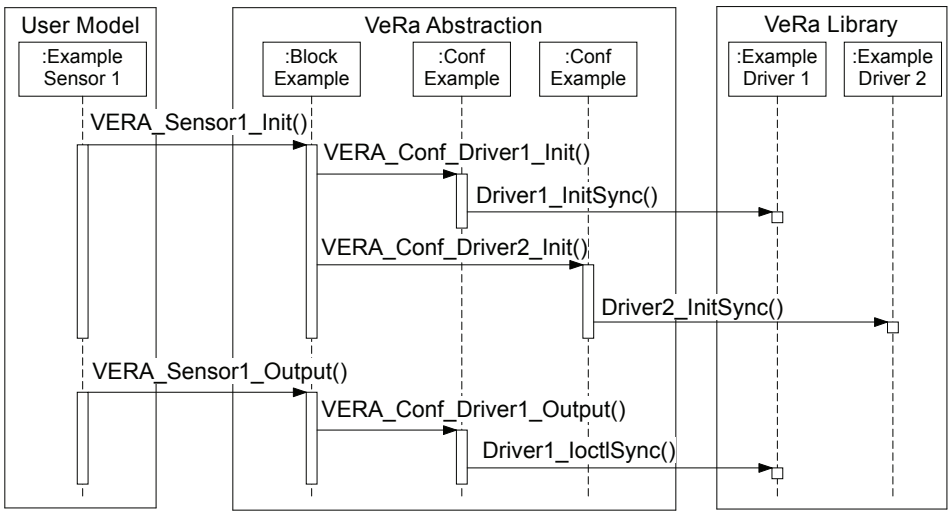

Abbildung 4.12: *Sensoraufruf im Anwendungsfall Code Generierung dargstellt im Sequenzdiagramm*

werden. Dafür wird aus der XML Konfigurationsdatei eine C Modul generiert, das die notwendigen Informationen enthält (vgl. Abschnitt 4.2.4) . Die Schnittstelle zum VeRa Ausführungssystem bilden somit die Hardwaretreiber, die vom Compiler und Linker mit eingebunden und von den Treiberkonfigurationsmodulen aufgerufen werden. Abbildung 4.12 zeigt das Beispiel eines Sensoraufrufs im Anwendungsfall Code Generierung. In dem gezeigten Beispiel müssen für einen Sensoraufruf zwei unterschiedliche Treiber aufgerufen werden. Diese werden zunächst initialisiert, wobei die zuvor aus der XML Datei generierte Konfiguration berücksichtigt wird. Bei jedem Ausführungsschritt des Modells wird der Sensorwert über einen Treiberaufruf ausgelesen. Im Beispiel ist dafür nur noch ein Treiberaufruf notwendig.

4.5 Ausführungssystem und Kalibrierarbeit mit VeRa

Nach der Reglerentwicklung mit Simulation und Codegenerierung für das Zielsystem wird die Reglersoftware an der realen Strecke zur Ausführung gebracht. Hierbei sind nachträgliche Parametrierungen des Reglermodells wesentlich, da es nahezu unmöglich ist bereits in der Simulationsphase alle Betriebszustände und Sonderfälle zu berücksichtigen. Die Möglichkeit einer komfortablen Reglerabstimmung auf dem Prüfstand und im Fahrzeug ist daher eine wichtige Eigenschaft für ein modellbasiertes Entwicklungssystem.

Weiterhin werden insbesondere bei einer Anwendung als Motorsteuerung die unterschiedlichen Reglerfunktionen in mehrere nebenläufige Prozesse aufgeteilt, die

mit unterschiedlicher Periode berechnet werden. So können langsame Prozesse mit niedriegerer Berechnungsperiode abgebildet werden als schnelle Prozesse, um die Prozessorauslastung dadurch gering zu halten. Zudem müssen einige Berechnungen synchron zur Motorumdrehung ausgeführt werden.

4.5.1 Übersicht

Für den Kalibrierzugriff haben sich in der Industrie unterschiedliche Kalibrierwerkzeuge etabliert. Um die Möglichkeit zu haben viele unterschiedliche Steuergeräte mit einem Kalibrierwerkzeug zu bedienen wurde mit der ASAM-MCD-2 [4] Schnittstelle eine Standardschnittstelle zwischen Steuergerät und Kalibrierwerkzeug geschaffen. Das VeRa System unterstützt diese Schnitstelle, indem während des Übersetzungsvorgangs Beschreibungsdateien erzeugt werden, die von den Kalibrierwerkzeugen eingelesen werden können. Diese Dateien enthalten Informationen zu den zu verstellenden Parametern, sowie Messwerten, die von dem Kalibrierwerkzeug verstellt und eingelesen werden können. Zudem wurde das Ausführungssystem mit der XCP [5] Schnittstelle mit einer Standardschnittstelle ausgestattet, die von den Kalibrierwerkzeugen ebenfalls unterstützt wird.

Für die Modellierung von Berechnungsschleifen mit unterschiedlicher Periodendauer beinhaltet VeRa ein Betriebssystem nach dem OSEK Standard. Das Betriebssystem erlaubt es durch die Definition von mehreren Tasks die nebenläufigen Prozesse abzubilden. Da OSEK weiterhin die Definition mehrerer Taktgeber zulässt, wurde zudem neben der Zeit auch der Kurbelwinkel eines Motors als Taktbasis implementiert. Somit können auch Tasks definiert werden, die synchron zur Motordrehzahl ausgeführt werden. Abbildung 4.13 zeigt das Beispiel eines Taskmodells mit zwei zeitbasierten Tasks und einem winkelbasierten Task, wie es in der Praxis häufig vorkommt. Niederpriore Tasks können von höher prioren Tasks preemptiv unterbrochen werden. Im Beispiel in Abbildung 4.13 besitzt Task 2 die niedrigste Priorität.

4.5.2 Realisierung in VeRa

Das eingebettete Ausführungssystem unterteilt sich in die Basissoftware und das Anwendermodell (vgl. Abbildung 4.14). Die Basissoftware enthält die Module für den Prozessorstart, den Aufruf des Anwendermodells sowie einer Instanz der XCP Schnittstelle für die Kommunikation. Das Anwendermodell setzt sich zusammen aus dem Betriebssystem OSEK, einem Aufrufsystem für den automatisch generierten Code und dem generierten Code selbst. Die Basissoftware ist im prozessorinternen Flashbereich untergebracht, das Anwendermodell hingegen befindet sich im externen Flashspeicher. Während die Basissoftware lediglich einmal bei der Auslieferung des Gerätes geflasht wird, muss das Anwendermodell nach jeder Änderung im Modell

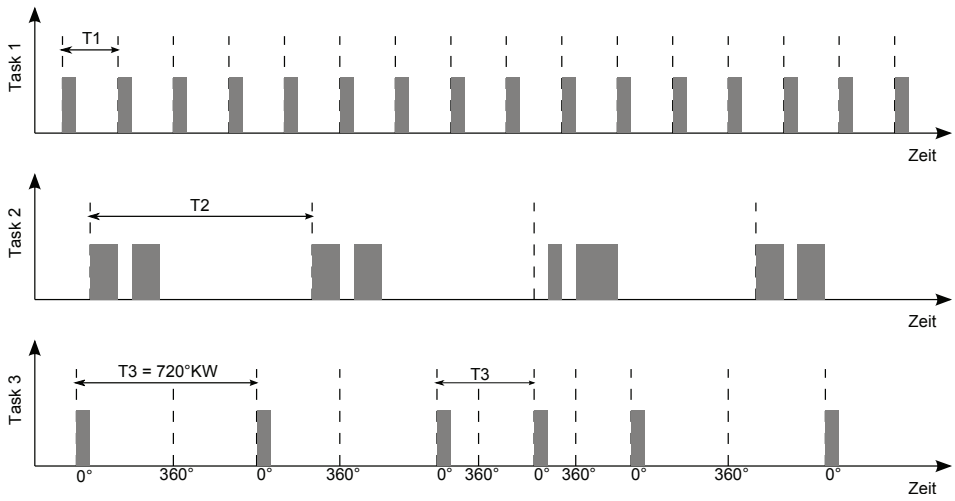

Abbildung 4.13: *Beispiel eines Taskmodells,*
Task 1 ist zeitbasiert mit kleiner Periodendauer (T1),
Task 2 ist zeitbasiert mit großer Periodendauer (T2),
Task 3 ist winkelbasiert mit einer Periodendauer (T3)

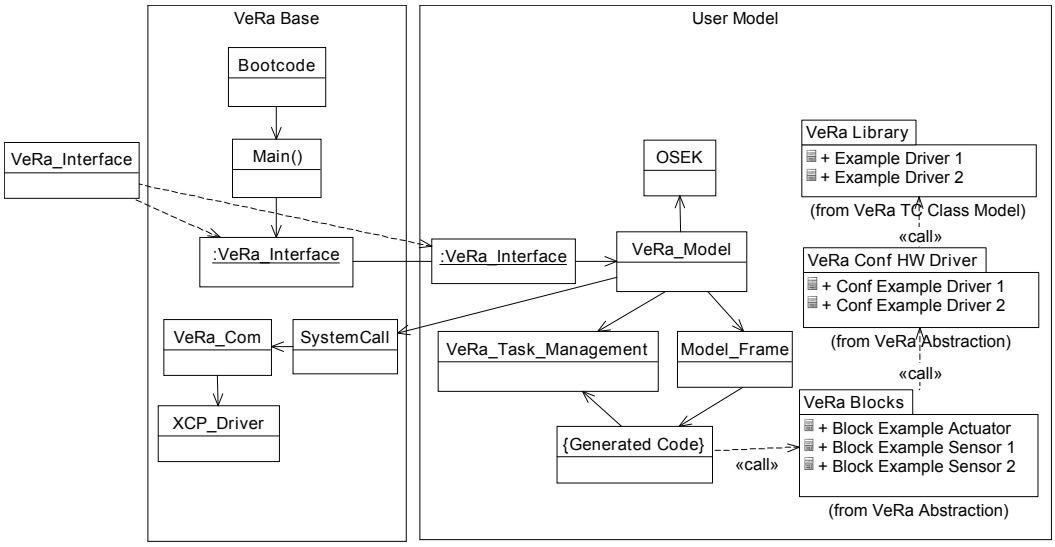

Abbildung 4.14: *Das VeRa Ausführungssystem im Klassendiagramm*

neu geflasht werden können. Durch diese Aufteilung des Speicherbereichs und die unterschiedlichen Zeitpunkte des Kompilierens ergibt sich die Problematik der Datenübergabe und des Programmsprungs von der Basissoftware zum Anwendermodell. Dies wird durch das `VeRa_Interface` Modul gelöst. Durch den Linker Prozess wird je eine Instanz dieses Moduls im gleichen Speicherbereich platziert. Das Interface Modul stellt die Funktionsaufrufe für das Anwendermodell zur Verfügung. Zur Kompilierzeit der Basissoftware zeigen die Funktionszeiger noch auf ein Dummymodell. Zur Kompilierzeit des Anwendermodells werden diese Funktionszeiger jedoch auf das Anwendermodell selbst umgelegt. Die Größe und damit die Adresse der einzelnen Funktionen ändert sich dadurch nicht. Der Inhalt, also die Funktionsaufrufadresse hingegen ist unterschiedlich. Während des Flashvorgangs für das Anwendermodell wird die Instanz des Interface Moduls durch eine neue Version mit aktualisierten Adressen überschrieben.

Über das Interface Modul wird das VeRa Modell einschließlich des OSEK Betriebssystems aufgerufen und gestartet. Dies geschieht durch das Modul `VeRa_Model`. Über das Modul `VeRa_Frame`, das individuell für jedes Modellierungswerkzeug erstellt wurde, wird der generierte Code des eigentlichen Anwendermodells aufgerufen. Über den generierten Code werden die Bibliotheksfunktionen des VeRa Systems aufgerufen (vgl. Abschnitt 4.4).

Ein Betriebssystem nach dem OSEK Standard ist ein statisch gebundenes Betriebssystem. Das heißt, alle Betriebsmittel, wie Tasks, Alarme, usw., sind bereits zur Kompilierzeit bekannt und können während der Laufzeit nicht verändert werden. Da die Betriebsmittel-Konfiguration dynamisch durch das Modell festgelegt wird, wird daraus die Konfiguration mit Hilfe des Codegenerator erzeugt. Zusätzlich zu den Anwender-Tasks wird noch ein weiterer Task für die Kommunikation mit dem Kalibrierwerkzeug über die XCP-Schnittstelle angelegt. Er besitzt die niedrigste Priorität und wird somit immer dann ausgeführt, wenn keine Modellteile gerechnet werden müssen.

Die Startsequenz des VeRa Ausführungssystems ist in Abbildung 4.15 dargestellt[13]. Nach dem Start des Prozessors wird das Modul `Main()` ausgeführt, das zunächst einige Initialisierungen durchführt. Anschließend wird die Speicheradresse des Kommunikationstasks `VeRaCom` dem Modul `VeRa_Model` des Anwendermodells mitgeteilt, da diese zur Kompilierzeit des Anwendermodells unbekannt ist. Der entsprechende Funktionsaufruf erfolgt über das zentrale Austauschmodul zwischen Basis Software und Anwendermodell `VeRa_Interface`. Anschließend wird der Start des OSEK Betriebssystem über den Funktionsaufruf `VeRa_StartOS()` vorbereitet. Dabei werden die Tasks `Idle`, `VeRaCom` und `Model`, die in jedem Fall benötigt

[13] Die Initialisierung der übrigen Betriebsmittel des Betriebssystem wie Alarme und Events wurden in dieser Darstellung aus Gründen der Übersichtlichkeit weggelassen. Sie erfolgt jedoch im Prinzip analog zur Definition und Initialisierung der Tasks.

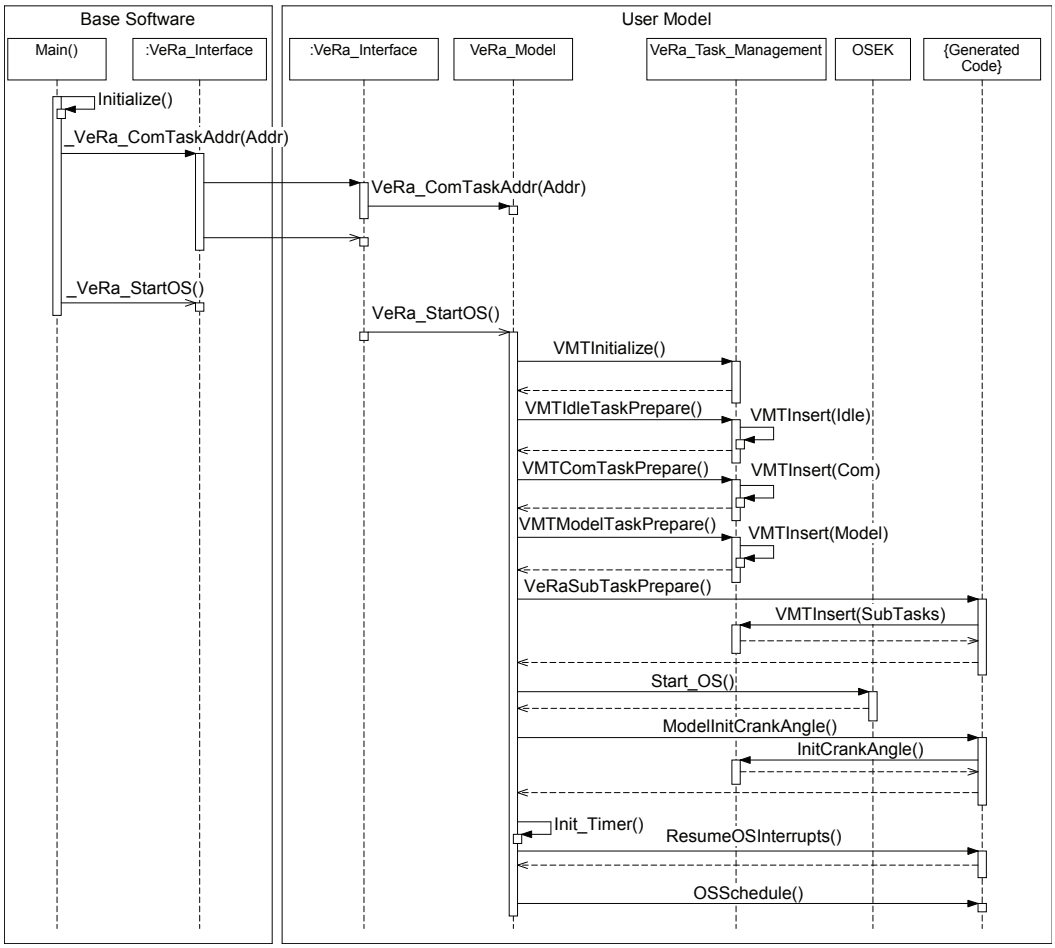

Abbildung 4.15: *Startsequenz des VeRa Ausführungssystems*

werden, initialisiert und durch das `VeRa_Task_Management` am Betriebssystem angemeldet. Die Tasks, die der Anwender in seinem Modell definiert hat, werden als Sub-Tasks des Modell-Task ebenfalls dem Betriebssystem mitgeteilt. Dies geschieht über den Aufruf der Funktion `VeRaSubTaskPrepare()`. Eine Funktion dieses Namens wird vom Code Generator bei jedem Generiervorgang individuell mit den jeweils vom Anwender über die XML Konfigurationsdatei festgelegten Parametern erzeugt. In einem weitere Schritt wird nun das Modul OSEK initialisiert, da zu diesem Zeitpunkt alle Betriebsmittel feststehen. Anschließend werden die Zeit- und Winkelbasis für das Betriebssystem durch Aufruf der Funktionen `init_timer()` und `InitCrankAngle()`gestartet. Im letzten Schritt wird der Scheduler des Betriebssystems durch die Funktion `OSSchedule()` aufgerufen. Danach wird nach der Schedulingstrategie der jeweils aktive Task bestimmt und ausgeführt.

4.6 Zuordnungstabelle

Im Folgenden sind Zuordnungstabellen (Traceability Matrices) dargestellt, die die einzelnen Architekturelementen den Anforderungen aus Kapitel 4.1 zuordnen. Dabei werden in Tabelle 4.3 die Anforderungen hinsichtlich Brauchbarkeit und in Tabelle 4.4 die Anforderungen hinsichtlich Wartbarkeit zugeordnet. Die in Kapitel 4.1 dargestellten Anforderungen werden vollständig abgedeckt.

#	Anforderung	Architekturelement
B11	Modellbasierte Entwicklung wird unterstützt	VeRa Toolbox, VeRa Abstraction
Die modellbasierte Entwicklung wird unterstützt, indem externe Modellierungswerkzeuge über diese Module angekoppelt werden.		
B12	Bibliothek von Sensoren und Aktuatoren	VeRa Toolbox
Die Bibliothek enthält ausschließlich Repräsentationen für Sensoren und Aktuatoren.		
B13	Offene Schnittstelle für User-Bibliotheken	VeRa Toolbox (Template)
Die Architektur der VeRa Toolbox kann Dritt-Unternehmen zugänglich gemacht werden, so dass eigene Bibliotheken erstellt werden können.		
B14	Verwendung des Werkzeugs als A-Muster	VeRa Hardware
Die Hardware wurde seriennah ausgelegt, d.h. insbesondere der Prozessor muss beim Übergang zur Serie nicht getauscht werden.		
B21	Integration aller Softwarekomponenten in einer Bediensoftware	VeRa Desk
Die Bediensoftware integriert alle Tool-Chain Komponenten in einer Oberfläche.		
B31	Unterstützung eines qualifizierbaren Code-Generators	VeRa Abstraction
Durch Kapselung aller notwendigen Funktionen in Modulen der VeRa Abstraktionsschicht, können unterschiedliche Code Generatoren sehr leicht angekoppelt werden.		

Tabelle 4.3: *Zuordnungstabelle von Architekturelemente zu den Brauchbarkeitsanforderungen*

#	**Anforderung**	**Architekturelement**
W11	Wiederverwendung von Architekturteilen	VeRa Base, VeRa Abstraction
Die Komponenten der Basis Software und die Bibliothekselemente werden bei allen Projekten wiederverwendet.		
W12	Unabhängigkeit vom Modellierungswerkzeug	VeRa Abstraction
Durch Kapselung aller notwendigen Funktionen in Modulen der VeRa Abstraktionsschicht, können unterschiedliche Modellierungswerkzeuge sehr leicht angekoppelt werden.		
W13	Konfiguration und Softwarevarianten	VeRa Configuration file
Durch Anpassung der XML Konfigurationsdatei können komfortabel Varianten erzeugt werden.		
W21	Einheitlicher Treiberaufbau	VeRa Library
Alle Treiber der VeRa Library wurden entsprechend des HIS Standards implementiert.		
W22	Weitgehende Verwendung von SW-Standards	VeRa Library, VeRa Base, ASAM MCD2
VeRa Library setzt den HIS Standard um, VeRa Base nutzt ein OSEK Betriebssystem und für die Ankopplung des Kalibrierwerkzeugs wird die ASAM MCD2 Schnittstelle verwendet.		
W23	Coding Standard für die Programmierung	–
Ist nicht Teil der Architektur, wird aber als organisatorische Maßnahme umgesetzt.		
W31	Unterstützung von Simulation	VeRa Toolbox, VeRa Abstraction
Durch Ankopplung von Matlab/Simulink wird Simulation nach dem Model-In-the-Loop (MIL) Prinzip ermöglicht. Die Verwendung von XCP als Kommunikationsschnittstelle erlaubt zusätzliche eine komfortable Hardware-In-the-Loop (HIL) Simulation.		

Tabelle 4.4: *Zuordnungstabelle von Architekturelemente zu den Wartbarkeitsanforderungen*

5 Evaluierung des KMU-orientierten, modellbasierten Entwicklungsprozesses

5.1 Einführung

In diesem Abschnitt wird erläutert, wie der in Kapitel 3 beschriebene Entwicklungsprozess implementiert werden kann und im Unternehmen VEMAC bereits zum großen Teil umgesetzt wird. Dazu werden einzelne Prozessimplementierungen betrachtet und die erreichten Prozessergebnisse diskutiert. Es ist sicherlich niemals möglich einen für alle Unternehmen allgemeingültigen Entwicklungsprozess zu definieren, aber dennoch können die hier durchgeführten Untersuchungen wichtige Hinweise zur Umsetzung eigener Prozesses bieten. Die Bewertung der einzelnen Prozesse erfolgt konzeptionell anhand des Automotive SPICE [63] Bewertungsschemas für den ersten SPICE Reifegrad. Um einen höheren Reifegrad zu erlangen ist die Implementierung von generellen Praktiken zusätzlich zu den Basispraktiken und deren Bewertung notwendig. Die dafür erforderlichen Maßnahmen in kleinen Unternehmen unterscheiden sich jedoch nur unwesentlich von denen für große Unternehmen. Eine Betrachtung dieser Prozesse würde daher den Rahmen dieser Arbeit übersteigen. Eine gute Übersicht zur Erlangung eines höheren Reifegrades für ein Beispielprojekt im Unternehmen Continental geben Höhn, et al. in [24].
Die beschriebenen Bewertungen wurden vom Autor selbst vorgenommen und stellen somit eine subjektive, zunächst nicht zertifizierbare Beurteilung dar.

5.1.1 Bewertungsmetrik

Die Voraussetzung für die Erreichung eines SPICE Level von größer oder gleich eins (1) ist das weitgehende oder vollständige Vorhandensein der Prozessausführung (Prozessattribut PA1.1) [63]. Dieses Prozessattribut ist ein Maß dafür, inwieweit der Zweck des Prozesses erreicht wird. Als Ergebnis einer vollständigen Erfüllung des Attributs gilt, wenn der Prozess seine definierten Ergebnisse erzielt.
Die Erreichung des Prozesszwecks wird demnach daran gemessen, inwieweit der Prozess seine Ergebnisse erreicht. Diese Beurteilung stützt sich auf den Indikator GP1.1.1 (Erziele die Prozessergebnisse) und dieser wiederum auf die Indikatoren zur Umsetzung der Basispraktiken und Arbeitsprodukte/ -ergebnisse. Die Basispraktiken und Arbeitsprodukte sind Anzeichen dafür, inwieweit der Prozesszweck erfüllt ist und

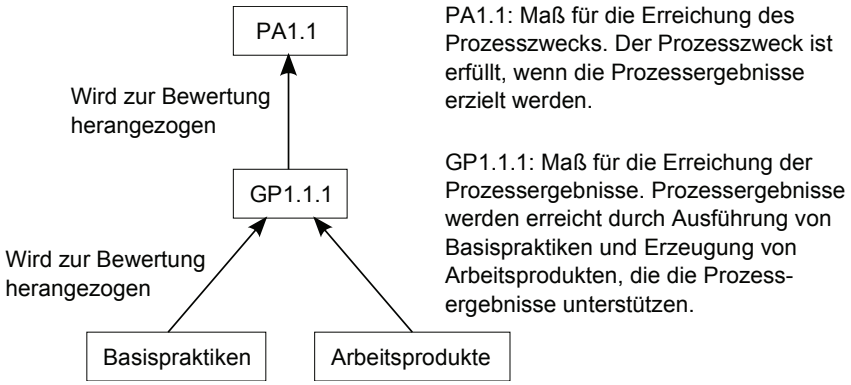

Abbildung 5.1: *Beurteilung von PA1.1 nach [45]*

die Prozessergebnisse erreicht werden [45]. Die Zusammenhänge sind in Abbildung 5.1 grafisch dargestellt. Für die Beurteilung der Umsetzung der Prozessergebnisse sollen folgende Kriterien gelten:

- Die Basispraktiken werden ausgeführt und befolgt und sind zweckmäßig implementiert, d.h. sie sind so gestaltet, dass sie dem jeweiligen Prozesszweck dienen und zu den Prozessergebnissen beitragen
- Die Arbeitsprodukte sind adäquat implementiert in Anlehnung an die Anforderungen des Automotive SPICE Assessmentmodells (siehe Annex B, [63]).

Die Beurteilung der Umsetzung der Basispraktiken (BP), die Beurteilung der Erreichung der Prozessergbnisse (GP1.1.1) und die Beurteilung zur Erfüllung des Prozesszwecks (PA1.1) erfolgt auf Basis einer vierstufigen Bewertungsskala (siehe Tabelle 5.1). Diese Bewertungsskala ist für SPICE allgemein in der ISO/IEC 15504-2 [32] festgelegt.

Die Beurteilung der einzelnen Prozesse erfolgt tabellarisch in folgender Aufteilung

- **Basispraktiken (BP.n)**: Bewertung nach vierstufiger Bewertungsskala inwieweit die jeweilige Basispraktik umgesetzt wird, stichwortartige Kommentierung
- **Prozessergebnisse (GP1.1.1)**: Bewertung nach vierstufiger Bewertungsskala inwieweit die jeweiligen Prozessergebnisse erreicht werden, stichwortartige Kommentierung, Mittelwert aller Bewertungen der Basispraktiken bildet die Grundlage
- **Prozessattribut (PA1.1)**: Bewertung nach vierstufiger Bewertungsskala inwieweit der jeweilige Prozesszweck erfüllt wird, stichwortartige Kommentierung, Bewertung der Prozessergebnisse bildet die Grundlage

Akronym	Bedeutung	Erfüllungsgrad	Wert für Mittelwertbildung	Beschreibung
N	Not achieved bzw. nicht erfüllt	0-15%	0%	Es gibt keine oder geringe Anzeichen der Erfüllung des definierten Attributs bei dem assessierten Prozess.
P	Partially achieved bzw. teilweise erfüllt	>15-50%	33,3%	Es gibt einige Anzeichen für eine Vorgehensweise und eine teilweise Erfüllung des definierten Attributs in dem assessierten Prozess. Einige Aspekte der Erfüllung des Attributs können unvorhersagbar sein.
L	Largely achieved bzw. überwiegend erfüllt	>50-85%	66,6%	Es gibt Anzeichen für eine systematische Vorgehensweise und eine signifikante Erfüllung des definierten Attributs in dem assessierten Prozess. Im assessierten Prozess können einige Schwächen bezüglich des Attributs existieren.
F	Fully achieved bzw. vollständig erfüllt	>85-100%	100%	Es gibt Anzeichen für eine vollständige und systematische Vorgehensweise und eine volle Erfüllung des definierten Attributs in dem assessierten Prozess. Im assessierten Prozess gibt es keine siginifikanten Schwächen bezüglich des Attributs.

Tabelle 5.1: *SPICE Bewertungsskala nach ISO/IEC 15504-2 [32]*

5.2 Anforderungserfassung

5.2.1 Beschreibung

Bereits in Kapitel 3 wurde die große Bedeutung insbesondere für kleine Unternehmen hervorgehoben die Kundenanforderungen möglichst präzise zu verstehen. Daher wird die Anforderungserfassung strukturiert umgesetzt. Zu Beginn eines Projekts werden die Anforderungen in Workshops in Form von Szenarien erhoben. Diese Szenarien werden anschließend, ebenfalls im Workshop, hinsichtlich ihrer Relevanz und ihres Umsetzungsaufwands bewertet. Dabei sind neben dem Auftraggeber (oder auch dem Kunden) der Projektverantwortliche sowie wichtige Spezialisten beteiligt, die das Projekt durchführen. Im Anschluss an den Workshop werden die Szenarien bewertet und in einen Anforderungskatalog in Form einer Liste übertragen. Die Liste wird mit dem Kunden bzw. dem Auftraggeber abgestimmt und im gegenseitigen Einverständnis als Baseline in der Version 1.0 festgeschrieben. Nachträgliche Änderungen müssen immer in Abstimmung mit Kunde und Auftraggeber erfolgen. Dabei wird die Versionsnummer erhöht.
Ein Nachteil dieser Art der Dokumentation ist jedoch, dass die im Klartext verfassten Anforderungen, maschinell kaum weiterverarbeitet werden können, da die Anforderungen nicht formal beschrieben sind. Durch eine automatisierte Weiterverarbeitung, kann z.B. die Konsistenz von Anforderungen überprüft werden. Des Weiteren sind die Beziehungen der Anforderungen untereinander durch Querverweise nur schwer nachvollziehbar. Diese Aspekte legen es nahe auch die Anforderungen modellbasiert zu beschreiben.
Ein Modell bietet die Möglichkeit Elemente und Beziehungen zwischen den Elementen zu definieren. Das Modell ist eine formale Beschreibung, die auch maschinell weiterverarbeitet werden kann. Die zuvor textuell formulierten Anforderungen, lassen sich durch Meta-Informationen im Modell zuordnen, wenn keine geeigneten Elemente zur Verfügung stehen. Und schließlich bietet ein Modell eine grafische Darstellung, die in der Regel eine anschaulichere Dokumentation darstellt. Wie bereits oben erwähnt, nimmt die detaillierte Anforderungsdokumentation in kleinen Unternehmen eine weniger wichtige Position ein, da ein gemeinsames Verständnis des Entwicklungsziel in einem kleinen Team leichter herzustellen ist. Zudem ist ein gemeinsames Verständnis des Entwicklungsziels zweckdienlicher, da die Anforderungen so durch jeden Mitarbeiter kritisch hinterfragt werden. Eine modellbasierte Darstellung der Anforderungen ist daher insbesondere für kleine Unternehmen eine interessante Option, da sie die Anforderungen anschaulicher beschreibt, und so das gemeinsame Problemverständnis unterstützt.
Angewendet auf das hier betrachtete Anwendungsfeld der Entwicklung automobiler Steuergeräte, sind die Anforderungen vielfältig. Es müssen sowohl funktionale, als auch nicht-funktionale Anforderungen berücksicht werden, die aus verschiedenen

Disziplinen wie der Softwaretechnik, der Elektronik und der Regelungtechnik entstammen. Daher wurde die modellbasierte Anforderungserfassung mit i* [68] (vgl. Abschnitt 2.1.4) im Rahmen des ZAMOMO Projekts (vgl. Abschnitt 2.1.2) auf die Anwendbarkeit in dieser Anwendungsdomäne untersucht. Die entsprechenden Arbeiten hierfür wurden unter Mitarbeit des Autors vor allem durch das Fraunhofer Institut für angewandte Informationstechnologie (FIT) durchgeführt.

5.2.2 Erfahrungen

Je nach Komplexität wird die Nachverfolgbarkeit der Anforderungen und Architekturelemente anhand von Listen und Zuordnungstabellen schwierig. Die Gefahr besteht darin, die Konsistenz zu verlieren. Daher kann es sinnvoll sein an dieser Stelle ein zusätzliche Unterstützung durch ein Werkzeug wie IBM/DOORS einzusetzen, zumal dessen Einsatz ohnehin von einigen Kunden verlangt wird. Für kleine Unternehmen sind diese Listen im Vergleich zu großen Entwicklerteams jedoch von geringerer Bedeutung. In großen Teams werden die sehr formal spezifizierten Anforderungen als Pflichtenheft für einzelne Teilentwicklungen durch einzelne Entwickler benötigt, da es nicht wirtschaftlich ist, wenn jeder Projektteilnehmer das gesamte Projekt überblickt. Im kleinen Team ist es jedoch zielführend das Projekt und die Kundenanforderungen initial allen Projektbeteiligten ausführlich zu erläutern. Viele Projektbeteiligte werden ohnehin an der Erarbeitung der Anforderungsbaseline mitgearbeitet haben. Das Konzept wird idealerweise unter Beteiligung aller Projektteilnehmer erarbeitet. Damit dient die Konzeptdokumentation intern lediglich noch als Gedankenstütze und extern der Abstimmung mit dem Kunden. Im kleinen Team ist es zudem ratsam durch regelmäßige Projektmeetings (vgl. SCRUM, Kapitel 2.7.2) den aktuellen Projektstatus bei allen Beteiligten auf dem neusten Stand zu halten. Der Kunde bzw. Auftraggeber wird in diese Meetings miteinbezogen, bzw. es werden regelmäßige Telefonkonferenzen abgehalten. Bei VEMAC erfolgen die regelmäßigen Projektmeetings durch eine wöchentliche Jour fixe.
Ergänzend zu dem so implementierten Prozess der Anforderungserfassung wurde der Einsatz einer modellbasierte Anforderungserfassung mit i* (vgl. Abschnitt 2.1.4) evaluiert. Um das i* Rahmenwerk für das Anwendungsfeld der Regelung eines Verbrennungsmotors zu nutzen, wurde ein allgemeines Domänenmodell erstellt [56], um das domänenspezifische Wissen zu verallgemeinern und abzubilden. Das gemeinsam von Regelungstechnikern und Softwaretechnikern entworfene Modell ist in Abbildung 5.2 dargestellt. Auf der obersten Ebene ist im *Strategic Dependency* Diagramm zentral die Motorsteuerung (electronic control unit) als das zu entwickelnde System dargestellt. Das um die Motorsteuerung angeordnete Umgebungsmodell besteht aus Verbrennungsmotor-spezifischen Komponenten, wie Zylinder oder Turbolader, und anderen Akteuren, wie dem Kunden. Dabei werden die Komponenten und Funktionen der Akteure (wie z.B. dem Motorblock oder der Motorsteuerung) jeweils

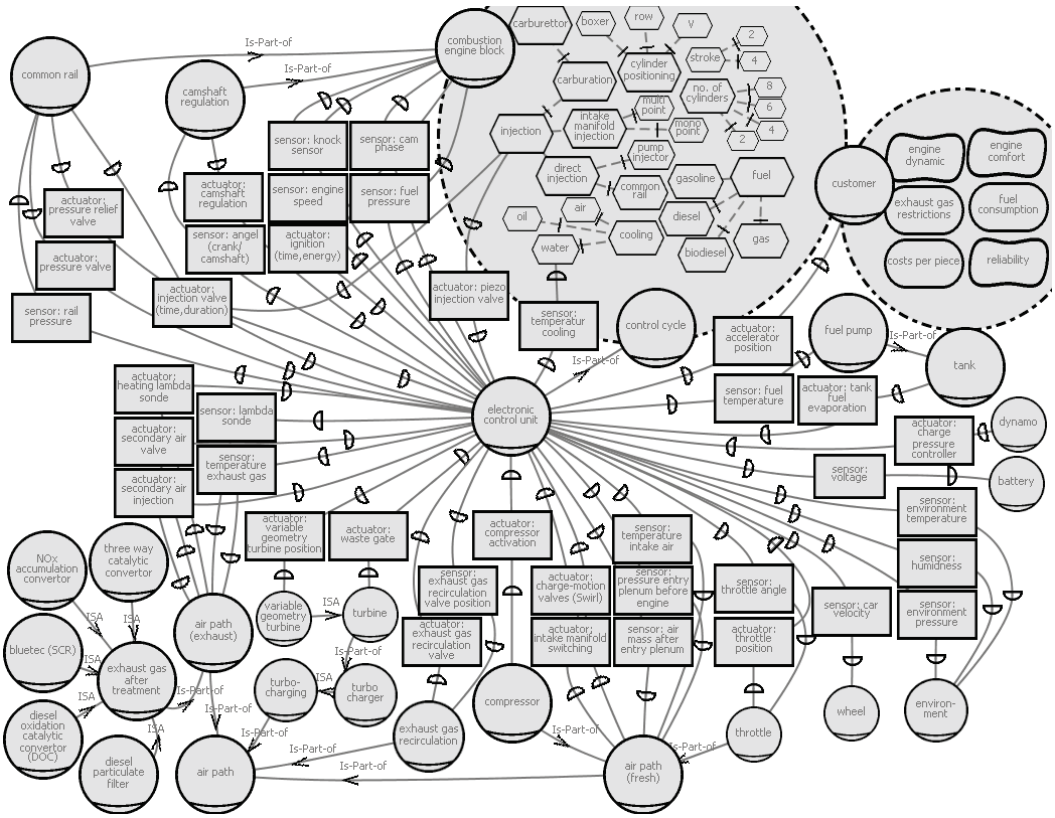

Abbildung 5.2: *Generisches Domänenmodell für Motorsteuerungssysteme [56]*

in zugeordneten Strategic Rationale Diagrammen abgebildet (große graue Kreise). Die Verbindungen zwischen den Motorkomponenten und der Steuerungseinheit sind auch auf dieser Ebene ausschließlich durch Repräsentationen von Sensoren und Aktuatoren realisiert.

Bei Anwendung des Modells für ein neues Projekt wird eine Instanz des allgemeinen Domänenmodells gebildet und es werden, die in dem betrachteten Projekt nicht vorhandenen Komponenten weggestrichen. So können aus dem allgemeinen Modell z.B. Modelle für Diesel- oder auch Ottomotoren abgeleitet werden. Auf diese Weise kann ein Anforderungsmodell sehr leicht für ein neues Projekt erstellt werden. Dies geschieht unter Verwendung des speziellen Editors OME [42] auf Eclipse Basis.

Der Editor wurde im Rahmen des ZAMOMO Projekts so erweitert, dass die Modelle an die deduktive, objektorientierte Datenbank ConceptBase[36] übertragen werden können. Durch Anfragen an diese Datenbank können dann z.B. Ähnlichkeitsanalysen mit vorhandenen Modellen durchgeführt werden. Auf diese Weise ist es möglich bereits abgeschlossene Projekte zu finden, die ähnlich gelagert waren wie das jetzt zu

betrachtende neue Projekt. Die Arbeitsergebnisse der gefundenen Projekte können mit der Problemstellung im neuen Projekt verglichen und besser wiederverwendet werden.

5.2.3 Bewertung

Die Bewertung des Anforderungserhebungsprozesses ENG.1 nach oben beschriebener Methode ist in Tabelle 5.2 dargestellt. Der Prozesszweck des Prozesses ENG.1 ist wie folgt definiert [63]:

> „Der Zweck des Anforderungserhebungs-Prozesses besteht darin, die während der Nutzungsdauer eines Produkts und/oder einer Dienstleistung entstehenden Kundenbedürfnisse und -anforderungen zu erheben, zu bearbeiten und nachzuverfolgen, um so eine Anforderungsbaseline zu erstellen, die als Basis für die Definition der benötigten Arbeitsprodukte dient."

Mit dem beschriebenen Prozess wird dieses Prozessziel und damit das Prozessattribut PA1.1 voll erfüllt, d.h. ein Reifegrad von eins oder höher ist für diesen Prozess erreichbar.

Das wichtigste Prozessergebnis ist die Festlegung der Kundenanforderungen als Baseline, die dem Konzept der Anforderungs Baseline (Requirements Baseline, vgl. Abbildung 2.6) im V-Modell nach Barry Boehm entspricht[1]. Diese Trennlinie wurde in den nachfolgenden, aufwändigen, im Auftrag der deutschen Bundesbehörden entwickelten, V-Modellen bis hin zum V-Modell XT aufgehoben bzw. aufgeteilt (siehe auch Abschnitt 2.7.1). Im Umfeld eines kleinen Unternehmens ist dieser Ansatz jedoch nach wie vor sehr zweckmäßig.

Für die Erreichung der Ziele nach Automotive SPICE ist die modellbasierte Anforderungserfassung nicht notwendig. Dennoch ergänzt der beschriebene Ansatz mit Hilfe der i* Modellierung den Prozess der Anforderungserfassung sinnvoll. Durch Verwendung des Domänenmodells mit Hilfe der i* Modellierung gelingt es Anforderungen zu formalisieren und in einer Datenbank abzulegen. Bei einer großen Anzahl ähnlich gelagerter Projekte ist dies ein gutes Hilfsmittel um die Wiederverwendung von Modellen, Anforderungen und Wissen allgemein zu steigern. Allerdings muss das Domänenmodell bei unterschiedlich gelagerten Projekten häufig ergänzt und erweitert werden. Erweiterungen sind jedoch problematisch, da zum einen die Komplexität des Domänenmodells immer weiter gesteigert wird. Dadurch geht die Übersichtlichkeit des ohnehin schon großen Modells (vgl. Abbildung 5.2) verloren. Zum anderen können die Erweiterungen nur die neu eingepflegten Projekte berücksichtigen, so dass die Vollständigkeit der Datenbank insgesamt abnimmt. Insbesondere kleine

[1] Die gilt auch begrifflich.

ENG.1 Basispraktiken	
BP1: Eingang von Kundenanforderungen und -anfragen	F
Die Kundenanforderungen werden initial in einem Workshop erfasst und anschließend als Baseline festgeschrieben. Eine regelmäßige Telefonkonferenz wird eingerichtet, bei der Änderungen in den Anforderungen vom Kunden eingebracht werden können.	
BP2: Kundenerwartungen verstehen	F
Die Kundenanforderungen werden als Baseline festgelegt. Der Initiale Workshop gemeinsam mit dem Kunden beugt Missverständnissen vor.	
BP3: Einverständnis bezüglich Anforderungen	F
Baseline wird im gegenseitigen Einverständnis festgeschrieben	
BP4: Festlegung der Kundenanforderungs Baseline	F
Baseline wird formal als Liste geführt. Ergänzend kann ein Modell der Anforderungen erstellt werden.	
BP5: Management der Änderungen der Kundenanforderungen	L
Bei Änderungen in den Kundenanforderungen wird eine neue Version der Kundenanforderungen erstellt. Änderungen erfolgen in Übereinkunft mit dem Projektteam, es erfolgt jedoch keine formale Auswirkungsanalyse.	
BP6: Einführung eines Mechanismus für die Kunden Lieferanten Anfragenkommunikation	L
Änderungen in den Anforderungen werden mit dem Kunden abgestimmt und protokolliert.	
GP1.1.1: Prozessergebnisse	F
Alle Prozessergebnisse werden durch die umgesetzten Basispraktiken erreicht.	
PA1.1: Prozessattribut	F
Das Projektziel wird in vollem Umfang erreicht. Durch Art und Weise des Prozesses wird ein intensiver, aber nicht überformaler Kundenkontakt gewährleistet.	

Tabelle 5.2: *Bewertung der Anforderungserhebung nach ENG.1*

und mittelständische Unternehmen leben jedoch von ihrer großen Flexibilität, so dass sich ein Domänenmodell besonders hier häufig ändert. Dies hat durch die Evaluierung einen allgemeinen Zielkonflikt in den Vordergrund gerückt:
Die domänenspezifische Anforderungsmodellierung bedeutet immer auch eine Verallgemeinerung der Anforderungen, indem diese einem allgemeinen Domänenwissen (z.B. in Form eines Modells) zugeordnet werden. Dies widerspricht jedoch der hohen Flexibilität und Variabilität in Projekten wie sie bei kleinen Unternehmen typisch ist.
Dieser Widerspruch wurde in [48] in Bezug auf die Arbeiten von Sutcliffe and Maiden in [60] dargestellt. Diese argumentierten bereits 1993, dass es in Bezug auf die Anwendung von domänenspezifischen Wissen zweifelhaft sei, ob Werkzeuge mit integriertem Domänenwissen jemals in der Lage sein werden mit der Entwicklung von neuen Systemen in sich stark ändernden Domänen mithalten zu können. Bezogen auf den hier untersuchten Ansatz bedeutet dies, dass das Domänenmodell entweder nur einen kleinen allgemeingültigen Teil enthalten kann oder, dass es aufgrund der fehlenden „Domänenstabilität“ zu einer Einschränkung der Flexibilität führt, was jedoch der starken Kundenorientierung entgegen stehen würde. Daher wurden in den Arbeiten [48] und [47] Ansätze vorgestellt, wie mit diesem hohen Maß an Änderungen im Domänenwissen umgegangen werden kann.
Insgesamt konnte gezeigt werden, dass die Modellierung von Anforderungen ein erstrebenswertes Ziel darstellt, um die Anforderungserfassung und -verwaltung darin zu unterstützen eine gemeinsame Sichtweise aller Beteiligten auf das zu entwickelnde System zu erlangen. In den durchgeführten Arbeiten wurde die grundsätzliche Eignung von i* als Modellierungssprache gezeigt und mit dem Domänenmodell ein erster Ansatz zur Wissensrepräsentation entwickelt. Neben dem erkannten theoretischen Weiterentwicklungsbedarf stehen dem praktischen Einsatz sicherlich auch noch einige praktische Aspekte wie folgt entgegen.

- Für die Verwendung der i* Modellierung zur Anforderungserfassung müssen Möglichkeiten erarbeitet werden, um die Verwendung von Anforderungen im Sinne eines Lastenhefts für das Kunde-Lieferanten-Verhältnis zu gewährleisten. Dazu müssen Möglichkeiten in die Modellierung eingebracht werden, um messbare Werte abbilden zu können.

- Der Sprachgebrauch der i* Modellierung ist mit Begriffen wie *Agenten*, *Zielorientierung* oder *Strategic Rationale Diagramm* sowohl für Softwaretechniker als auch für Regelungstechniker eher ungewöhnlich und kann zu Missverständnissen und damit Akzeptanzproblemen führen. Die Symbolik der grafischen Repräsentation ist zudem wenig anschaulich und besteht aus vielen unterschiedlichen Elementen, was zunächst zu Kommunikationsproblemen führen kann. Dieser Aspekt konnte auch im Zamomo Projekt beobachtet werden.

- Wie in dieser Arbeit dargestellt wurde, ist die Bewertung von Entwicklungsprozessen (z.B. nach Automotive SPICE) auch für kleine Unternehmen ein wichtiger Aspekt im Geschäftsalltag. Daher müssen Möglichkeiten geschaffen werden, um die Anforderungsmodellierung kompatibel zu den entsprechenden Basispraktiken zu gestalten.

- Die derzeitigen Bemühungen der OMG die Anforderungen zu modellieren sollten bei einer Einführung von i* berücksichtigt werden. Insbesondere da i* in der Basis auf die Wissensrepräsentationssprache TELOS aufbaut, bietet i* sicherlich einen großen Vorteil gegenüber den bisher lediglich als UML Erweiterung spezifizierten Sprachelementen der OMG.

5.3 Konzepterstellung

5.3.1 Beschreibung

Aufbauend auf der im Anforderungserfassungsprozess festgelegten Anforderungs-Baseline wird ein Konzept erarbeitet, in dem das zu entwickelnde System und im Speziellen die zu entwickelnde Software beschrieben werden. Dabei werden auf Basis der Kundenanforderungen zunächst Systemanforderungen abgeleitet, aus denen eine Systemarchitektur abgeleitet und beschrieben wird. Darauf aufbauend werden Softwareanforderungen beschrieben, um daraus die Sofwarearchitektur abzuleiten. Wenn das System lediglich aus Softwarekomponenten besteht, können die Schritte Systemanforderungen und Systemarchitektur entfallen. Die Dokumentation der Architektur erfolgt unter Zuhilfenahme der UML Notation. Zusätzlich werden die Anforderungen und die Architekturelemente nummeriert und in Listen gepflegt. Die Zuordnung der System- und Softwareanforderungen sowie der Architekturelemente zu den Kundenanforderungen erfolgt in einer oder, falls notwendig, mehreren Zuordnungstabellen *(Traceability Matrices)*. Darüberhinaus enthält das Konzept auch Angaben über die Erstellung einer Testumgebung bzw. eines Umgebungsmodells.
Der Prozess zur Erstellung des Konzepts integriert somit die in Automotive SPICE definierten Prozesse Systemanforderunegn (ENG.2), Systemarchitektur (ENG.3), Softwareanforderungen (ENG.4) und Softwarearchitektur (ENG.5). Im Ergebnis wird ein Dokument erstellt, welches das zu erstellende System beschreibt und in Beziehung zu den Kundenanforderungen setzt. Das so erstellte Konzept wird in einem Konzept-Review mit dem Kunden bzw. dem Auftraggeber begutachtet. Auf Basis des Reviews wird das Konzept geändert und in Abstimmung mit dem Kunden als Konzept-Baseline in der Version 1.0 festgeschrieben.

5.3.2 Erfahrungen

Die Erarbeitung des Konzepts in einem integrierten Prozess unterstützt die enge und interdisziplinäre Zusammenarbeit. Dies entspricht im hohem Maße der Struktur von kleinen und mittelständischen Unternehmen, die durch den so gestalteten Prozess gestärkt wird. Die Trennung der in Automotive SPICE definierten Prozesse zwingt den Anwender zu einer separierten Dokumentation und getrennten Beschreibung. Dies ist für die Arbeit in einem großen Team notwendig, da Aufgaben so parallel durch Entwickler abgearbeitet werden können, auch ohne voneinander zu wissen. Dies ist jedoch in einer klein- und mittelständischen Unternehmensstruktur nicht notwendig und somit kann darauf verzichtet werden.
Das erarbeitete Konzept wird in einem Review mit dem Kunden abgestimmt. Dadurch wird der Kunde in die Entwicklung mit einbezogen und es kann auf zusätzliche Wünsche oder Kritik frühzeitig eingegangen werden. Die bei kleinen und mittelständischen Unternehmen ausgeprägte Kundenbindung wird somit durch den Prozess unterstützt. Bei der Besetzung des Review Teams ist es im Sinne der Softwarequalität zudem wichtig neben dem Kunden auch technisch versierte Mitarbeiter des Auftragnehmers einzubeziehen. Durch den Kunden wird vor allem die Gebrauchsqualität (vgl. Abschnitt 2.2) sichergestellt, da ein in allen Belangen zufriedenstellend funktionierendes System dessen oberstes Ziel ist. Die Wartungsqualität hingegen wird vor allem durch Entwickler und Entscheider des Auftragnehmers sichergestellt, da sie ein besonderes Interesse daran haben diese Qualitäten zu stärken, um die Wiederverwendung von Software- und Architekturteilen auch in anderen Projekten zu ermöglichen.

5.3.3 Bewertung

Aus den Kundenanforderungen wird ein integriertes Konzept dadurch erarbeitet, dass aus den Kundenanforderungen zunächst System- bzw. Softwareanforderungen abgeleitet werden. Aus diesen Anforderungen wiederum wird eine System- bzw. Softwarearchitektur abgeleitet. Die Prozesszwecke für die System- bzw. Softwareanforderungen (ENG.2 und ENG.4 nach Automotive SPICE) sind wie folgt definiert [63]:

> „Der Zweck des SystemanforderungsanalyseProzesses (ENG.2) besteht darin, definierte Kundenanforderungen in systemtechnische Sollanforderungen zu überführen, die die Grundlagen des Systemdesigns bilden.
> Der Zweck des SoftwareanforderungsanalyseProzesses (ENG.4) besteht darin, die Softwareanforderungen der Softwareelemente des Systems zu ermitteln.“

Aus diesen Anforderungen wiederum werden anschließend die System- und Softwarearchitekturen (ENG.3 und ENG.5) abgeleitet. Die Zwecke der entsprechenden

ENG.2 und ENG.4 Basispraktiken	
BP1: Ermittlung der System- und Softwareanforderungen	P
Aus den Kundenanforderungen werden System- und Softwareanforderungen abgeleitet und Listen verwaltet. In aller Regel sind die Listen aber nicht vollständig im Sinne der Anforderungsspezifikationen (Annex B, [63]).	
BP2: Untersuchung der System- und Softwareanforderungen	L
Die Untersuchung soll vor allem im Hinblick auf Realisierbarkeit, Verifizierbarkeit und Messbarkeit der ermittelten System- bzw. Softwareanforderungen erfolgen. Dies erfolgt im beschreibenden Text des Konzepts, sowie unter Angabe der Messkriterien in der Anforderungsliste.	
BP3: Bestimmung der Auswirkungen auf die Betriebsumgebung	L
Die Schnittstellen des Systems- bzw. der Software sind im Konzept beschrieben. Zusätzlich werden notwendige Anforderungen an die Umgebung (Hardwarefähigkeiten, externe Software, usw.) hier beschrieben.	
BP4: Priorisierung und Kategorisierung der System- und Softwareanforderungen	L
Es erfolgt eine Einordnung der Umsetzung der System- und Softwareanforderungen in die Musterstufen. Gegebenenfalls werden noch weitere Zwischenstände definiert und den Anforderungen zugewiesen. Das Konzeptpapier ordnet zudem Prioritäten zu, falls notwendig.	
BP5: Bewertung und Aktualisierung der System- und Softwareanforderungen.	L
In Absprache mit dem Kunden erfolgt die Anforderungsspezifikation im Konzeptreview. Aufbauend darauf wird eine Anforderungsbaseline im Konzeptpapier und in der Anforderungsliste als Version 1.0 festgelegt. Änderungen werden anhand von Versionen gepflegt	
BP6: Sicherstellung von Konsistenz und Traceability in beide Richtungen zwischen den Kunden- und den Systemanforderungen, bzw. zwischen den System- und den Softwareanforderungen	L
Zu den Anforderungen wird eine Traceability Matrix angelegt. Dabei wird darauf geachtet, dass die Zuordnung konsistent ist.	
BP7: Kommunikation der System- und Softwareanforderungen	L
Teil der Konzeptbeschreibung ist die Spezifikation der Anforderungen für System und Software.	
GP1.1.1: Prozessergebnisse	L
Alle Prozessergebnisse werden durch die umgesetzten Basispraktiken weitgehend erreicht. Die Anforderungen werden nicht so detailliert beschrieben, wie in Automotive SPICE gefordert.	
PA1.1: Prozessattribut	L
Das Projektziel wird weitgehend erfüllt. Die Kundenanforderungen werden systematisch in System- bzw. Softwareanforderungen überführt.	

Tabelle 5.3: *Bewertung der System- bzw. Softwareanforderungsanalyse nach ENG.2 bzw. ENG.4*

ENG.3 und ENG.5 Basispraktiken	
BP1: Definition der System- und Softwarearchitektur	F
Die Definition der System- und Softwarearchitektur erfolgt modellbasiert anhand von UML Diagrammen und textueller Beschreibung. Die Anforderungen werden hierbei berücksichtigt und den einzelnen Qualitäten zugeordnet. Die Qualitäten des Qualitätsbaumes dienen als Checkliste für die Vollständigkiet des Entwurfs.	
BP2: Zuweisung von System- und Softwareanforderungen	F
Teil des Konzepts ist eine Tabelle von Zuweisungen, die die Anforderungen den Architekturelementen zuordnet.	
BP3: Definition der Schnittstellen	P
Die Schnittstellen werden durch UML Diagramme dargestellt und kommentiert, wo dies notwendig ist. Eine detaillierte Beschreibung der Schnittstellen erfolgt nur, wo externe Komponenten angebunden werden, intern verlässt man sich auf die Kommunikation unter den Projektteilnehmern.	
BP4: Beschreibung des dynamischen Verhaltens	L
Das dynamische Verhalten wird anhand von UML Sequenzdiagrammen beschrieben.	
BP5: Definition von Zielen hinsichtlich der Ressourcennutzung	L
Die Ziele werden im Konzept beschrieben, wenn dies erforderlich ist. Sie werden vor allem für die parallele Entwicklung von Hardware und Software benötigt.	
BP6: Entwicklung eines Softwarefeindesigns	P
Das Softwarefeindesign wird ebenfalls mit UML Diagrammen im Konzept beschrieben, wo dies sinnvoll erscheint. Die Beschreibung ist in der Regel nicht vollständig.	
BP7: Entwicklung von Verifkationskriterien	P
Verifikationskriterien werden vor allem für externe Komponenten festeglegt. Bei internen Komponenten werden diese Kriterien für Module definiert, die wiederverwendet werden sollen.	
BP8: Verifikation des System- und Softwaredesigns	F
Die Verifikation des System und Softwaredesigns erfolgt anhand des Konzeptreviews. Das Review wird entsprechend dem Reviewprozess nach SUP.4 durchgeführt.	
BP9: Sicherstellung von Konsistenz und Traceability in beide Richtungen zwischen den System- bzw. Softwareanforderungen und der System- bzw. Softwarearchitektur	L
Es wird eine Traceability Matrix erstellt, die im Review überprüft und ergänzt wird.	
BP10: Sicherstellung von Konsistenz und Traceability in beide Richtungen zwischen der Softwarearchitektur und dem Softwarefeindesign	L
Es wird eine Traceability Matrix erstellt, die im Review überprüft und ergänzt wird.	
GP1.1.1: Prozessergebnisse	L
Die Prozessergebnisse werden weitgehend erfüllt. Das Softwarefeindesign wird in einem kleinen Team nicht so ausführlich beschrieben, wie von der Norm gefordert.	
PA1.1: Prozessattribut	L
Der Prozesszweck wird weitgehend erfüllt. Eine System- und Softwarearchitektur wird erstellt. Das Softwarefeindesign wird nur an den Stellen detailliert ausgearbeitet, an denen externe Schnittstellen bestehen. In einem kleinen Team ist das zulässig, da die Entwicklung weniger verteilt erfolgt.	

Tabelle 5.4: *Bewertung der System- und Softwarearchitekturentwicklung nach ENG.3 und ENG.5*

Prozesse sind wie folgt definiert [63]:

> „Der Zweck des Prozesses Entwurf der Systemarchitektur (ENG.3) besteht darin, festzustellen, welche Systemanforderungen welchen Elementen des Systems zugewiesen werden.
> Der Zweck des Prozesses Entwurf des Softwaredesigns (ENG.5) besteht darin, ein Design für die Software bereitzustellen, mit dem die Softwareanforderungen implementiert werden können und der gegenüber den Softwareanforderungen verifiziert werden kann."

Obwohl die Prozessschritte in kleinen Unternehmen wie oben beschrieben integriert durchgeführt werden, erfolgt hier eine Bewertung separat nach System- und Softwareanforderungen (ENG.2 und ENG.4, siehe Tabelle 5.3) sowie System- und Softwarearchitektur (ENG.3 und ENG.5, siehe Tabelle 5.4). Diese Zusammenlegung ist auch im Sinne von Automotive SPICE zulässig (Anmerkungen zu ENG.4, [63]). Wie den Tabellen 5.3 und 5.4 zu entnehmen ist, werden mit dem oben beschriebenen Vorgehen der Konzepterstellung Anforderungen erhoben und eine Architektur abgeleitet. Die Vorgehensweise entspricht somit weitgehend den Kriterien für Automotive SPICE Prozesse ENG.2-5. Durch die Zusammenlegung der Prozesse und durch das wenig ausdetaillierte Softwarefeindesign wird deutlich weniger Aufwand erzeugt, als dies bei großen Unternehmen der Fall sein muss. Der Schwerpunkt wird bei dem beschriebenen Vorgehen auf ein hohes Kunden- bzw. Auftraggebereinverständnis gelegt, indem ein ausführliches Review durchgeführt wird. Der Dokumentationsaufwand hingegen wird verglichen mit separiert durchgeführten Prozessen gering gehalten.

5.4 Softwareerstellung

5.4.1 Beschreibung

Ein Teil des erstellten Konzepts (vgl. Abschnitt 5.3) ist die Softwarearchitektur. Die einzelnen Module der Architektur werden in der Phase der Softwareerstellung implementiert und getestet. Der Einsatz von modellbasierter Software-Entwicklung erlaubt hierbei eine integrierte Implementierung bei gleichzeitiger Simulations- und Testmöglichkeit vom einzelnen Modul bis zum System, da für die Implementierung und für die Simulation das gleiche Modell verwendet wird. Dies stellt bis zu einem gewissen Grad eine Verlagerung der Tätigkeiten bezogen auf das Vorgehen im V-Modell dar. War bei der klassischen Softwareentwicklung ein Systemtest erst in einer späten Phase auf dem aufsteigenden Ast des V-Modell möglich, so können bestimmte Funktionen durch die modellbasierte Softwareentwicklung bereits während der Implementierung getestet werden. Durch dieses Vorgehen können Fehler in der Konsistenz

der Spezifikation bereits in der Implementierungsphase gefunden werden, wenn eine Testumgebung bzw. ein Streckenmodell mit hohem Detailgrad zur Verfügung steht. Bei konsequentem Einsatz von modellbasierter Implementierung und Codegenerierung kann der Modultest auf C-Code Ebene sogar entfallen [24]. Dies gilt allerdings nur bei Software für nicht sicherheitskritische Systeme. Darüber hinaus werden die nachfolgenden Tests mit großer Wahrscheinlichkeit ohne die Notwendigkeit der Nacharbeit bestanden. Insgesamt ergibt sich so eine effizientere Entwicklung, da Fehler früher aufgedeckt werden und eine Nacharbeit weniger kostenintensiv ist. Bei der Wahl des Code Generators ist darauf zu achten, ob dieser in der Lage ist Code in der für das jeweilige Projekt geforderten Qualität zu erzeugen.
Trotz vieler Vereinfachungen durch die modellbasierte Entwicklung muss auch bei der Erstellung von Modellen eine Qualitätssicherung angewendet werden, indem Styleguides wie z.B. MAAB[2] für Matlab/Simulink Modelle als eine organisatorische Maßnahme beachtet werden. Darüberhinaus werden auch für die erstellten Modelle Reviews als analytische Maßnahme durchgeführt, um die Qualität sicherzustellen. Dabei wird besonders darauf geachtet, dass die Architekturteile im Modell so kommentiert sind, dass eine Nachverfolgbarkeit zum Softwaredesign und den Softwareanforderungen gegeben ist.
Für die Codeteile, die konventionell in C oder Assembler erstellt werden, wurde eine Codierrichtlinie erstellt, die auch die MISRA[3] Regeln beinhaltet. Darüberhinaus werden auch hier Codereviews als analytische Methode verwendet, um eine hohe Codequalität zu erreichen. Auch hier wird besonderen Wert auf die Kommentierung gelegt, um Bezug zu den spezifizierten Softwaredesignelementen und den Softwareanforderungen zu gewährleisten.
Das Ziel der Softwareerstellung ist dann erreicht, wenn alle Elemente des Konzepts erstellt und im Sinne des Konzepts getestet wurden. Dazu wird zu Beginn des Softwareerstellungsprozesses ein Testplan erstellt, in dem festgelegt wird wie welche Module getestet werden sollen (z.B. Review, White-Box-Test, oder ähnliches). Weiterhin werden im Testplan Abnahmekriterien für das Bestehen der jeweiligen Tests festgelegt.

5.4.2 Erfahrungen

Kleine Unternehmen müssen sich in ihren Auftraggebern, die in der Regel große Unternehmen sind, anpassen. Dies gilt insbesondere für die Softwareerstellung und diese kann durch eine konsequente modellbasierte Softwareerstellung unterstützt werden. Da die Entwicklung von automobilen Steuergeräten überwiegend prototypisch erfolgt, ist die Übertragbarkeit von Ergebnissen während des gesamten Entwick-

[2] http://www.mathworks.de/automotive/standards/maab.html
[3] http://www.misra.org.uk/

lungsprozesses von entscheidender Bedeutung. Durch den Einsatz von Modellen für die Funktionssoftware können Ergebnisse zwischen unterschiedlichen Phasen der Entwicklung ausgetauscht werden. Auf diese Weise können Modelle der Regelungssoftware erstellt und zunächst im Rechner mit Hilfe von Simulation validiert werden. Aus den Modellen kann automatisch C-Code generiert werden, der zunächst für Prototypensysteme und später für das Zielsystem kompiliert wird. Auf diese Weise ist die Übertragbarkeit von Testergebnissen von der Simulationsphase bis zum realen System gegeben.
Kleine Unternehmen übernehmen häufig Teilprojekte, etwa während der Simulation, der Prototypenphase oder der A-Muster-Erstellung. Die intensive Nutzung von modellbasierter Softwareentwicklung bietet hierfür die Voraussetzung, da Modelle mit dem Kunden entweder als Arbeitsergebnis oder als Beistellung genutzt werden können.
Diese Erfahrungen können insbesondere durch den Einsatz von speziellen Entwicklungswerkzeugen, wie das in dieser Arbeit vorgestellte (vgl. Kapitel 4) unterstützt werden. Sie wurden in die Anforderungen für das Werkzeug eingearbeitet und anschließend umgesetzt. Die Erfahrungen im Umgang mit dem Werkzeug und damit auch die Erfahrungen mit der modellbasierten Softwareerstellung werden in Kapitel 6 nach unterschiedlichen Aspekten diskutiert.

5.4.3 Bewertung

Die Softwareerstellung baut nach Automotive SPICE auf die Ergebnisse des Software Designs auf, das im hier vorgestellten Entwicklungsprozess in der Konzepterstellung umgesetzt wird. Die im Design geplanten Module werden hier implementiert und im Sinne des Designs verifiziert. Der Zweck des Software Erstellungsprozesses (ENG.6) ist dementsprechend wie folgt definiert [63]:

> „Der Zweck des Softwareerstellungs Prozesses besteht darin, verifizierte Softwareeinheiten zu erstellen, die das Softwaredesign richtig wiedergeben.“

Die Bewertung dieses Vorgehens nach der Automotive SPICE Prozessbewertung ENG.6 ist in Tabelle 5.5 dargestellt. Der Prozesszweck wird im Sinne von Automotive SPICE weitgehend erreicht.

5.5 Review Prozess

5.5.1 Beschreibung

Wie bereits in Abschnitt 3.4 dargelegt, sind Bewertungsverfahren in Form von Reviews anerkannte Maßnahmen zur Steigerung der Qualität des begutachteten

ENG.6 Basispraktiken	
BP1: Definition einer Strategie für die Verifikation von Softwareeinheiten	L
Vor dem Beginn der Implementierung wird ein Testplan aufgestellt. Er enthält eine Definition, zu welchen Modulen Code- oder Modellreviews abgehalten werden. Der Testplan beschreibt die Methoden, mit denen einzelne Module getestet werden sollen.	
BP2: Entwicklung von Kriterien für die Verifikation der Softwareeinheiten	L
Die Verifikationskriterien sind ebenfalls im Testplan enthalten.	
BP3: Entwicklung von Softwareeinheiten	F
Softwareeinheiten werden entsprechend des Konzepts als Modell oder C-Code erstellt.	
BP4: Verifikation von Softwareeinheiten	L
Die Softwareeinheiten werden entsprechend des Konzepts verifiziert. Die dazu durchgeführten Verifikationen decken sich zum Teil mit dem Testplan.	
BP5: Protokollieren und Ablegen der Ergebnisse der Verifikation von Softwareeinheiten	L
Die Testergebnisse werden im Sinne des Testplans dokumentiert.	
BP6: Sicherstellung von Konsistenz und Traceability in beide Richtungen zwischen den Softwareanforderungen und den Softwareeinheiten	P
Die Konsistenz ist durch Kommentare in Code und Modell gegeben. Es wird eine Traceability Matrix angelegt.	
BP7: Sicherstellung von Konsistenz und Traceability in beide Richtungen zwischen dem Softwarefeindesign und den Softwareeinheiten	P
Die Konsistenz ist durch Kommentare in Code und Modell gegeben. Es wird eine Traceability Matrix angelegt.	
BP8: Sicherstellung von Konsistenz und Traceability in beide Richtungen zwischen den Softwareeinheiten und den Testspezifikationen der Softwareeinheiten	F
Als Teil des Testplans wird eine entsprechende Traceability Matrix angelegt.	
GP1.1.1: Prozessergebnisse	L
Die Prozessergebnisse werden weitgehend erfüllt. Aufgrund der nicht so detaillierten Spezifikation des Softwaredesigns ist auch die Testspezifikation etwas gröber gefasst. Dennoch wird insbesondere durch die modellbasierte Entwicklung eine hohe Testabdeckung erreicht.	
PA1.1: Prozessattribut	L
Der Prozesszweck wird weitgehend erfüllt.	

Tabelle 5.5: *Bewertung des Softwareerstellungsprozesses nach ENG.6*

Arbeitsprodukts. Dabei kann das begutachtete Arbeitsprodukt ein Konzept, eine Architektur oder auch C-Code sein. Speziell für die Durchführung in kleinen Unternehmen wurde, wie bereits in Kapitel 3.4 beschrieben, ein Rahmenwerk entwickelt, dass eine Vorlage für die Durchführung von Reviews liefert. Neben vielen gleichen Komponenten für die Durchführung eines Reviews muss jedoch eine spezielle Bewertungsmethode für unterschiedliche Arbeitsprodukte verwendet werden. Somit kann das Rahmenwerk an das jeweils durchzuführende Review angepasst werden. Zur Evaluierung wurden ein Architekturreview und ein Code-Review durchgeführt. Zur Einführung werden hier zunächst die allgemeine Teile des Rahmenwerks (Rollen, Ablauf und Regeln) nach [61] beschrieben.

Rollen

Die Rollenverteilung des Reviewverfahrens orientiert sich an den Varianten *Structured Walkthrough* und *Technisches Review*. Es ist gegenüber dem technischen Review vereinfacht und kann mit geringerem Aufwand durchgeführt werden, dafür ist allerdings ihr Nutzen auch etwas geringer [41]. Außerdem sind Elemente des *Structured Walkthrough* enthalten, jedoch mit einer detaillierteren Organisation. Das Review Team besteht aus einer kleinen Gruppe von Mitarbeitern. Die tatsächliche Anzahl von Personen und die Personenauswahl variiert in Abhängigkeit des zu prüfenden Dokuments und anhand des vorhandenen Fachwissens. Bei der Prüfung von technischen Dokumenten, wie beispielsweise bei einem Code Review, ist das Team kleiner, da dort die Breite an benötigten Expertenwissen auch schmaler ist. Drei Personen sind jedoch mindestens erforderlich, deren Rollen nach [61] wie folgt eingeführt werden:

- „Der *Manager* ist in der Regel der Projektleiter. Er trägt die Verantwortung für die Freigabe des Prüflings. Der Manager nimmt aber nicht am Review teil. Damit wird verhindert, dass der Vorgesetzte die Kritik eines Prüflings auf die Beurteilung des Mitarbeiters überträgt [41]“.

- „Der *Autor* ist der Urheber des zu prüfenden Arbeitsergebnisses. Wurde das Arbeitsergebnis in einem Team erstellt, ist er der Repräsentant des Teams. Er moderiert und leitet die Review-Sitzung. Dazu stellt er das Arbeitsergebnis schrittweise vor.“

- „Der *Gutachter* ist ein Kollege, der aufgrund seines Fachwissen das entwickelte Dokument beurteilen kann. In erster Linie sind das Personen die in dem Software-Entwicklungsprozess beteiligt sind, aber auch Personen die ein begründetes Interesse an dem Produkt haben wie z.B. Benutzer des Systems im Falle einer Überprüfung der Anforderungsspezifikation. Autoren des zu prüfenden Dokuments dürfen jedoch nicht die Rolle des Gutachters übernehmen.“

- „Der *Protokollant* erstellt in der Review-Sitzung das Protokoll. Er protokolliert die gefundenen Fehler in einer Fehlerliste. Die Fehlerliste enthält den Fundort, eine kurze Beschreibung sowie eine Klassifizierung des Fehlers. Außerdem protokolliert er weitere wichtige Informationen in einer separaten Liste. Dazu gehören z.B. Fehler, die in einem Referenzdokument entdeckt worden sind."

Das Team kann je nach bedarf um die Rolle eines *Moderators* erweitert werden, wie sie aus dem technischen Review bekannt ist. Diese Rolle kann nach [41] sehr anspruchsvoll sein, wäre aber sinnvoll, wenn ausreichend Ressourcen und genügend Erfahrung mit Reviews bestehen.

Der Ablauf

Der Ablauf des Review wird, wie in [61] detailliert beschrieben, in folgende Phasen aufgeteilt:

- Planung
- Initialisierung
- Einführungssitzung (optional)
- Review-Sitzung
- Bewertung
- Nachbearbeitung
- Anschluss-Sitzung

Die Phasen erlauben eine strukturierte Durchführung des Reviews. Der immer gleiche Aufbau erlaubt es den beteiligten Mitarbeitern die Situation wiederzuerkennen. Dadurch wird die Effektivität im Laufe der Zeit gesteigert.

Regeln

Generelle Regeln sollen die effektive und effiziente Durchführung des Reviews unterstützen. Diese betreffen auch soziale Aspkete. Besonders die Autoren fühlen sich in den ersten Reviews unwohl. Sie müssen sich erst daran gewöhnen, dass ihre Arbeit von anderen Kollegen berurteilt wird. Eine Aufklärung über die Prinzipien und Ziele eines Review können die Probleme reduzieren [41]. Das Prinzip des Reviews setzt immer Teamarbeit voraus. Daraus lassen sich neben der Aufdeckung von Mängeln weitere wichtige Vorteile eines Reviews ableiten. Durch routinemäßige Reviews wird die Zusammenarbeit der Entwickler vertrauter und damit letztendlich auch effizienter [41]. Die folgenden Regeln nach [46] und [41] sollten daher unbedingt

beachtet werden:

- Das Dokument und nicht der Autor wird beurteilt.
- Direkte Vorgesetzte nehmen an der Review Sitzung nicht teil.
- Während der Review Sitzung werden keine Lösungen diskutiert. Für Diskussionen steht im Anschluss Zeit zur Verfügung.
- Die Rollen dürfen in einer Sitzung nicht vertauscht werden.
- Der Fokus des Reviews ist auf die technischen Aspekten des Prüflings gerichtet. Es werden beispielsweise keine Management Entscheidungen diskutiert, die der Prüfling reflektiert.
- Das gemeinsame Ziel des Reviews muss immer im Mittelpunkt stehen. Persönliche Interessen dürfen das Review nicht beeinflussen.
- Persönliche Auseinandersetzungen müssen von den Teammitgliedern unterbunden werden.
- Reviews müssen in der Projektplanung mit den erforderlichen Aufwänden berücksichtigt werden.
- Die Planungsdokumente dürfen nach der Initialisierung des Reviews nicht mehr verändert werden.
- Alle Teilnehmer der Review-Sitzung unterschreiben das Protokoll.

5.5.2 Erfahrungen

Um Erfahrung in der Anwendung des erstellten Review Prozesses zu sammeln wurden zur Evaluierungen zwei Reviews nach oben beschriebenen Vorgehen durchgeführt. Im ersten Review wurde die Architektur der Abstraktionsschicht als Teil der Gesamtarchitektur des erstellten Entwicklungswerkzeugs bewertet (vgl. [61]). Im zweiten Review wurde der Code der Implementierung eines Teils der Abstraktionsschicht untersucht und bewertet (vgl. [18]). Die Ergebnisse der durchgeführten Reviews, sowie eine Bewertung des vorgestellten Prozesses nach den Kriterien des Automotive SPICE Prozesses SUP.4 werden anschließend dargestellt.

Architektur Review

Für das Architektur-Review wurde *SAAM* als Prüfmethode ausgewählt (vgl. Kapitel 2.5.3). Zunächst werden die Beteiligten des Reviews und ihre Zuständigkeiten vorgestellt. Sie bilden ein Gruppe von sechs Personen. Es gibt einen Autor, einen Protokollant und vier Gutachter. Die Rolle des Managers fällt in dem Kontext dieser Arbeit weg, da hier der Autor auch gleichzeitig die Rolle des Projektleiters einnimmt.

- **Gutachter**
  - 2 Mitarbeiter aus der Abteilung Software-Entwicklung, die aufgrund ihrer Ausbildung und Erfahrung Experten bezüglich des VeRa-Systems sind.
  - 1 Mitarbeiter aus der Abteilung Elektronik-Entwicklung. Der Mitarbeiter ist Experte bezüglich der VeRa-Hardware und nutzt Matlab/ Simulink Modelle, um die Elektronik zu testen. Somit kann er als Anwender der VeRa-Abstraktionsschicht angesehen werden.
  - 1 leitender Angestellter. Er ist mit den Anforderungen und Zielen des Produktes *VeRa* bestens vertraut.
- **Autor**
  - Der Autor hat die zu bewertende VeRa-Architektur weiterentwickelt.
- **Protokollant**
  - 1 Mitarbeiter aus dem Vertrieb.

Bei der Zusammenstellung der Gutachter wurde darauf geachtet, dass Experten verschiedener Fachbereiche diese Rolle einnehmen. Dadurch soll erreicht werden, dass Szenarien erstellt werden, die möglichst viele verschiedene Aspekte des Systems berücksichtigen. Durch die Heterogenität der Szenarien wird das System auf einer breiteren Ebene geprüft.

Teinehmer der Phase Review-Sitzung sind der Autor, die Gutachter und der Protokollant. Der Autor stellt in dieser Sitzung zunächst die Architektur des Systems vor. Anschließend erstellen die Gutachter die *Anwendungs-* bzw. *Änderungsszenarien*. Das heißt, die Szenarien beschreiben Funktionen die das System unterstützen muss (*Anwendungsszenarien*) oder eventuell in Zukunft unterstützen sollte (*Änderungsszenarien*). Im letzten Schritt werden die Szenarien von den Gutachtern priorisiert. Bei Fragen bezüglich der Architektur steht der Autor zur Verfügung. Der Protokollant hat in dieser Phase die Aufgabe, die verschiedenen Szenarien und ihre Priorisierung zu protokollieren. Dazu verwendet er ein vorgefertigtes Formular, in dem die Szenarien eingetragen werden. Um die Ermittlung der Szenarien zu unterstützen wird ein Qualitätenbaum mit kurzer Erklärung der einzelnen Qualitäten an die Beteiligten

ausgegeben. Dieser Qualitätenbaum kann in diesem Zusammenhang als Checkliste genutzt werden. Zur Priorisierung der Szenarien werden die drei verschiedenen Stufen *low*, *medium* und *high* verwendet. Das heißt Szenarien mit höchster Priorität werden mit der Stufe *high* gekennzeichnet.
Die Architektur wurde von den Beteiligten schnell verstanden. Somit konnte früh mit der Erstellung der Szenarien begonnen werden. Viele Beteiligte notierten sich bereits während der Vorstellung der Architektur Ideen für die Szenarien. Dementsprechend wurden direkt im Anschluss der Architekturbeschreibung verschiedene Szenarien von den Beteiligten genannt und konstruktiv diskutiert. Es wurden insgesamt fünfzehn Szenarien erstellt. Dabei wurden sechs Szenarien als *high*, fünf als *medium* und vier als *low* eingestuft.
In der Bewertungs Phase findet anschließend die eigentliche Bewertung der Architektur statt. Dazu müssen die Gutachter, anders als in der Review-Sitzung, Erfahrungen in der Software-Entwicklung besitzen. Aus diesem Grund nehmen in dieser Phase lediglich die zwei Gutachter aus der Abteilung Software-Entwicklung teil. Die weiteren Teilnehmer sind der Autor und der Protokollant. Die Bewertung geschieht mittels eines iterativen Prozesses bestehend aus drei Schritten. Im ersten Schritt wird das Szenario mit der höchsten Priorität gewählt. Danach stellt der Autor in dem zweiten Schritt den Architekturansatz bezüglich des Szenarios vor. In dem dritten Schritt wird der Architekturansatz bewertet. Im Anschluss an diesen iterativen Prozess wird auf Basis der bewertenden Architekturansätzen eine Empfehlung gegen bzw. für die Freigabe der Architektur ausgegeben.
Die Skalierung der Bewertung folgt dem Prinzip der Schulnoten. Bei der Bewertung mittels *Anwendungsszenarien* wird geprüft, wie die Architektur die bestimmte Anwendung erfüllt. Dementsprechend vergeben die Gutachter die Noten. Zur Beurteilung der Architektur mittels *Änderungsszenarien*, wird das Prinzip der Kopplung und Kohäsion zur Notenvergabe herangezogen. Zur Ermittlung der Kopplung werden die zu ändernden Module identifiziert. Die Zahl der betroffenen Module entspricht dem Kopplungsgrad. Eine hohe Kopplungsgrad bedeutet, dass keine ausreichende Trennung von Zuständigkeiten in der Architektur berücksichtigt wurde. Zur Ermittlung des Kohäsionsgrades wird notiert, welche Szenarien dasselbe Modul betreffen. Deckt ein Modul mehrere Szenarien ab, übernimmt dieses Modul zu viele Funktionen. Das bedeutet eine geringe Kohäsion des Moduls [41].
Die Protokollierung dieser Phase wird von dem Protokollant übernommen und wird durch zwei vorgefertigte Formulare unterstützt. In ein Formular werden die Schulnoten eingetragen. Das andere dient zur Dokumentation des Kohäsion- bzw. Kopplungsgrades. Auf Basis der ermittelten Noten wird im Anschluss des iterativen Bewertungsprozesses eine Empfehlung ausgegeben. Die Empfehlung beinhaltet entweder Punkte die nachzuarbeiten sind oder eine Befürwortung zur Freigabe des Dokumentes ohne Nachbearbeitung. Zur Formulierung einer Empfehlung wurden die „Ausreißer“ der Noten betrachtet (siehe dazu Tabelle 5.6). In der Abbildung

#Szenario	Priorität	Noten					
		1	2	3	4	5	6
01	high	X					
02	high		X				
03	high		X				
04	high	X					
05	high						
06	high			X			
07	high			X			
08	high			X			
12	high	X					
14	high	X					
15	high		X				
09	medium	X					
13	medium	X					
10	low	X					
11	low	X					

Tabelle 5.6: *Ergebnis der Architekturbewertung [61]*

sind die Szenarien entsprechend ihrer Priorität aufgelistet. Jedem Szenario wird eine Schulnote zugeordnet. Die weiß hinterlegten Szenarien wurden ohne Befund akzeptiert. Eine dunkelgraue Markierung der Szenarien bedeutet ein Befund, welcher nachbearbeitet werden muss. Das hellgrau hinterlegte Szenario wird nicht berücksichtigt, mit der Begründung, dass es außerhalb des Architekturbereichs liegt. Zusätzlich zu der Notenvergabe erfasst der Protokollant die Begründungen der Notenfindung.

Das erfolgreiche Review zeigte sich unter anderem in der konstruktiven Beteiligung aller Stakeholder in den einzelnen Phasen. So wurde in der Review Sitzung eine große Anzahl relevanter Szenarien in einem kurzen Zeitraum erstellt. Unterstützt wurde dies durch eine heterogene Gruppe von Stakeholdern, die aufgrund ihrer verschiedenen Fachbereiche unterschiedliche Sichtweisen beitrugen. In der Bewertungsphase wurden Mängel in der Architektur identifiziert. Dadurch wurde das Ziel des Reviews Fehler *frühzeitig* zu entdecken erfolgreich erreicht. Die Durchführung des Reviews unterstützte zudem die Verwendung eines einheitlichen Sprachgebrauch bezüglich der Software-Architektur. Eine detaillierte Darstellung dieses Architektur-Reviews gibt [61].

Code Review

Das Code-Review wurde neben dem recht aufwändigen Architektur-Review bewusst mit minimalem Aufwand als *Structured Walkthrough* [41] durchgeführt. Dabei wurde der C-Code eines einzigen Moduls mit Hilfe des Code-Viewing Werkzeugs tagSEA[4] vom Autor zwei Gutachtern präsentiert. Die Gutachter sind Experten in der Softwareentwicklung, von denen aber lediglich einer vertraut war mit der zu begutachtenden Architektur.
In der Phase Planung des Review-Rahmenwerks bereitet der Autor das Code-Review vor, indem er eine Präsentation des Codes erstellt. Dies kann durch Folien, Text oder auch ein Online Hilfsmittel geschehen.
In der Phase Review Sitzung wird in einer kleinen Präsentation vom Autor ein kurzer Überblick über den Inhalt des zu begutachtenden Codes gegeben. Anschließend wird der Code sukzessiv und strukturiert gesichtet. Dabei werden folgende Schritte durchgeführt.

- Verifikation Namensgebung
  Es wird geprüft, ob die Namensgebung der Codeelemente die Funktionalität des damit bezeichneten Elements wiedergibt und sie dem Coding Standard entspricht.

- Verifikation der Code Relevanz
  Es wird geprüft, ob der Code relevant im Sinne der Funktionalität des geprüften Elementes ist. Hierdurch wird sichergestellt, dass die Dekomposition im Sinne der Software-Architektur auch in der Implementierung richtig wiedergegeben wurde.

- Verifikation Erweiterbarkeit
  In diesem Schritt wird geprüft, ob der Code sinnvoll im Sinne der Erweiterbarkeit erstellt wurde. Wenn für ein Element die Wiederverwendung in anderen Applikationen durch entsprechende Anforderungen geplant war, so wird hier ebenfalls geprüft, ob dies möglich ist.

- Verifikation der minimalen Code Komplexität
  In diesem Schritt wird geprüft, dass es keine Code Wiederholungen gibt, die durch eine andere Dekomposition in Unterfunktionen vermieden werden können. Des Weiteren wird geprüft, ob der Code vereinfacht werden kann und das Schleifen nur verwendet werden, wo dies erforderlich ist.

- Verifikation der algorithmischen Komplexität
  Es wird geprüft, ob die Anzahl der Ausführungspfade minimal ist und nur die Pfade enthalten sind, die für die Anwendung auch ausgeführt werden.

[4] http://tagsea.sourceforge.net/

Paramter	Gutachter 1	Gutachter 2
Aufgewendete Zeit	1 Stunde	1 Stunde
Architekturkennnis	Nein	Ja
Anzahl gefundener Fehler	1	3

Tabelle 5.7: *Ergebnis des Code Reviews*

Während der Sitzung wird ein Protokoll erstellt, in dem die zu verändernden Punkte aufgeführt werden. Die Gutachter geben gleichzeitig eine Handlungsempfehlung an den Autor ab.
In der Phase Nachbearbeitung beseitigt der Autor die gefundenen Fehler und pflegt die gewünschten Änderungen ein. Bei großen Änderungen wird bei Bedarf ein weiteres Review angesetzt, um die Änderungen von den Gutachtern nochmals überprüfen zu lassen.
In dem hier dargestellten Code-Review wurde ein Softwaremodul mit insgesamt 406 Codezeilen (LOC) untersucht. Dafür wurden insgesamt zwei Mannstunden der Gutachter und etwa vier Mannstunden des Entwicklers aufgewendet. Es fällt auf, dass der Gutachter mit Kenntnis der Architektur wesentlich effektiver gearbeitet hat (vgl. Tabelle 5.7), da er sich wirklich auf den Code konzentrieren konnte. Der zweite Gutachter hatte Schwierigkeiten den Code zu verstehen, da er mit der Architektur nicht vertraut war. Das Ergebnis überrascht deswegen insofern ein wenig, als dass die Bewertungskriterien für den Code unabhängig von der Architektur sind. Aber insbesondere die Untersuchung von externen Schnittstellen erfordert Architekturkenntnis. Eine detaillierte Auswertung des Code Reviews ist in [18] dargestellt.

5.5.3 Bewertung

Automotive SPICE sieht für die Durchführung von Reviews einen eigenen Unterstützungsprozess (SUP.4) vor. Ein Review kann zu unterschiedlichen Zeiten im Projekt und zu Bewertung von unterschiedlichen Dokumenten oder Arbeitsprodukten eingesetzt werden. Der Prozesszweck ist wie folgt definiert [63]:

> „Der Zweck des Prozesses bezüglich gemeinsamer Reviews besteht darin, ein gemeinsames Verständnis mit den Stakeholdern bezüglich der Fortschritte im Vergleich zu den vereinbarten Zielsetzungen zu wahren, und abzustimmen, was getan werden sollte, um sicherzustellen, dass ein Produkt entwickelt wird, das zur Zufriedenheit der Stakeholder ist. Gemeinsame Reviews gibt es sowohl auf der Projektmanagementebene als auch auf der technischen Ebene und werden im Laufe der gesamten Projektdauer durchgeführt.“

SUP.4 Basispraktiken	
BP1: Definition von Review-Objekten	F
Durch das Review Rahmenwerk und die unterschiedliche Ausprägung ist eine vollständige Definition gegeben. Die jeweilige Organisation erfolgt durch das Projektmanagement.	
BP2: Einrichtung eines Verfahrens zur Handhabung der Review Ergebnisse	F
Im Review Rahmenwerk wird die Rolle des Protokollanten festgelegt, der bei allen Reviews zwingend zu besetzen ist. Die Review Ergebnisse werden so systematisch festgehalten. Die Informationsweitergabe innerhalb des Projekts erfolgt durch den Projektmanager.	
BP3: Vorbereitung gemeinsamer Reviews	F
Durch die Phasen Planung und Initialisierung ist die Vorbereitung des Reviews festgelegt.	
BP4: Durchführung gemeinsamer Reviews	F
Erfüllt durch die Phasen Review-Sitzung und Bewertung des Review Rahmenwerks.	
BP5: Kommunikation der Ergebnisse	L
Das Protokoll wird durch den Projektmanager verteilt.	
BP6: Bestimmung von Maßnahmen aufgrund der Review Ergebnisse	F
In der Phase Bewertung wird diese Praktik erfüllt, indem Handlungsempfehlungen durch die Gutachter gegeben werden.	
BP7: Verfolgung der Maßnahmen bezüglich der Review Ergebnisse	L
Gegebenenfalls wird bei großem Änderungsbedarf ein weiteres Review vorgesehen. Die sonstige Verfolgung der Umsetzung der Maßnahmen erfolgt durch den Projektmanager.	
BP8: Identifikation und Aufzeichnung von Problemen	F
In der Review-Sitzung werden Probleme erkannt und in einem Protokoll festgehalten.	
GP1.1.1: Prozessergebnisse	F
Bei Beachtung des Review Rahmenwerkes werden alle Ergebnisse erreicht.	
PA1.1: Prozessattribut	F
Der Prozesszweck wird durch das vorgestellte Review-Rahmenwerk und seinen unterschiedlichen Ausprägungen in vollem Umfang erfüllt.	

Tabelle 5.8: *Bewertung der gemeinsamen Reviews nach SUP.4*

Wie in Tabelle 5.8 dargestellt wird der Prozesszweck durch das vorgestellte Rahmenwerk in vollem Umfang erfüllt und damit ist ein Reifegrad von sicherlich größer als eins (1) für diesen Prozess erreichbar. Entscheidend für die Wirtschaftlichkeit von Reviews ist die Adaption des Rahmenwerkes unter Berücksichtigung des zu begutachtenden Objekts. Solange das Rahmenwerk beachtet wird, bleibt der Reifegrad des Prozesses unberührt.

5.6 Weitere Prozesse

Weiterer Prozesse insbesondere zum Projektmanagement und zur Wiederverwendung sind für kleine Unternehmen entscheidend, um Projekte erfolgreich durchzuführen. Dennoch liegen eine detailliertere Beschreibung über Kapitel 3 hinaus und eine Bewertung dieser Prozesse nicht mehr im Fokus dieser Arbeit.

6 Evaluierung der modellbasierten Werkzeugkette

Die Evaluierung der modellbasierten Werkezugkette wurde an unterschiedlichen Fallbeispielen durchgeführt. Dabei werden in den einzelnen Fallbeispielen jeweils unterschiedliche Aspekte der Werkzeugkette untersucht, um eine möglichst vielfältige Erfahrungsbasis zu erhalten.
Darüberhinaus wurde das Entwicklungswerkzeug mit der hier beschriebenen erweiterten Architektur auch bereits in Kundenprojekten bei VEMAC eingesetzt. Die Erfahrungen, die hieraus resultieren, sind in den Text zu den jeweiligen Fallbeispielen mit eingeflossen und gekennzeichnet.

6.1 Anwendungsfeld Motorsteuerung

Eines der zentralen Einsatzfelder des Steuerungssystem VeRa ist der Einsatz als Steuerung für Verbrennungsmotoren. Das zeigt sich auch in den verfügbaren Hardware-Derivaten (vgl. Tabelle 4.2). Drei der vier Standardvarianten sind für diesen Anwendungsfall ausgelegt.
Die hier dargestellte Evaluierung erfolgte im Rahmen von [61].

6.1.1 Versuchsaufbau

Für die Evaluierung wurde das Modell einer Motorsteuerung unter Matlab/Simulink implementiert. Als Umgebungsmodell wurde das Modell eines Verbrennungsmotors entwickelt, welches das stationäre und das dynamische Verhalten eines Verbrennungsmotors abbildet. Das Steuerungsmodell wurde im geschlossenen Regelkreis in der Simulation getestet. Anschließend wurde aus dem Steuerungsmodell mit Hilfe des Entwicklungswerkzeugs C-Code generiert und für die Verwendung auf der VeRa Hardware compiliert. Mit diesem System wurde ein realer Verbrenungsmotor auf dem Prüfstand getestet. Im letzten Schritt wurden die Messergebnisse mit den Simulationsergebnissen verglichen.

Umgebungsmodell

Als Versuchsträger dient ein 2-Zylinder Ottomotor mit Saugrohreinspritzung der Firma Weber Motor. Die Tabelle 6.1 zeigt die wesentlichen technischen Daten des

Zylinder	2
Ventile pro Zylinder	4
Bohrung x Hub	85 mm x 66 mm
Hubraum	750 ccm
max Leistung	55 PS bei 5500 U/min
max Drehmoment	72 Nm bei 4000 U/min
Gewicht	47 kg
Kompression	10 : 1

Tabelle 6.1: *Technische Daten des Versuchsmotors (Quelle: Weber Motor AG)*

Motors in der Ausführung ohne Turbolader.

Für die Modellbildung wird das Gesamtsystem Verbrennungsmotor zunächst in einzelne Teile zerlegt. Diese Teile müssen dann physikalisch beschreibbar und mathematisch formulierbar sein. Abbildung 6.1 gibt einen schematischen Überblick über den zu modellierenden Teil eines Motors und bietet einen ersten Ansatz den Motor in physikalische Teilbereiche zu gliedern.

In der Abbildung 6.1 sind die abstrakten Teilbereiche Drosselklappe, Saugrohr, Einspritzung, Zündung, Brennraum und Reibungen dargestellt. Diese Bereichen lassen sich in Teilmodelle des gesamten Motormodells beschreiben. Daraus ergibt sich eine Modellstruktur, die in Abbildung 6.2 dargestelltist. Die Blöcke repräsentieren die verschiedenen kombinierten Teilmodelle, welche jeweils eigenen physikalischen Effekten zugeordnet sind und miteinander durch berechnete Zwischengrößen verbunden sind. Die dabei verwendeten Teilmodelle sind entweder mathematisch formulierbar oder mit Kennfeldern hinterlegt. Das Verhalten von Drosselklappe und Saugrohr wird in dem sogenannten Luftpfadmodell beschrieben. Dort werden über die Befüll- und Entleermethode die Größen Luftmassenstrom durch die Drosselklappe, der Druck im Saugrohr sowie die in den Zylinder einströmende Luftmasse berechnet. Die eingespritzte Kraftstoffmasse ergibt sich über Kennfelder, in denen die Einspritzzeit hinterlegt ist. Für die Berechnung des Brennverlaufs bzw. des Zylinderdruckverlaufs wird der Ansatz nach Vibe [66] angewandt. Das bietet den Vorteil, dass auch die Einflüsse des Zündwinkels auf den Brennverlauf darstellbar sind. Die Reibungen im Zylinder und an der Kurbelwelle werden nach einem Ansatz von [19] berücksichtigt. Aus dem Antriebsmoment wird letztlich die Motordrehzahl berechnet. Eine detaillierte Beschreibung des Motormodells ist in [61] zu finden.

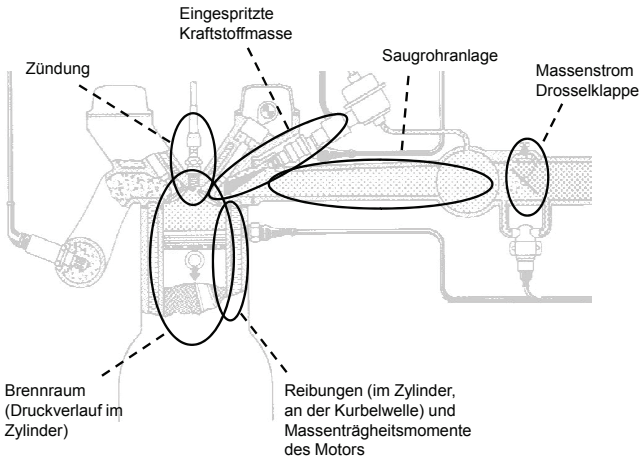

Abbildung 6.1: *Ansatz zur Zerlegung des Motors in Komponenten (nach [39], Hintergrundbild: Bosch)*

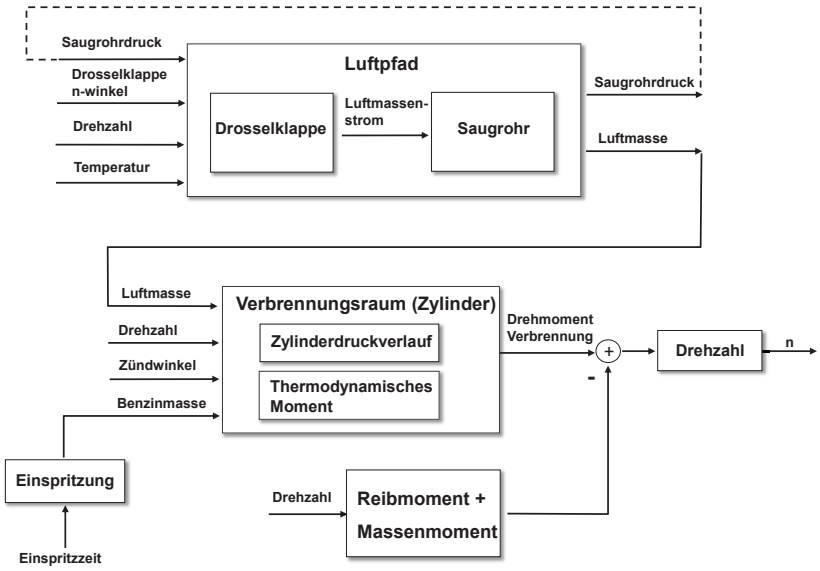

Abbildung 6.2: *Modulare Modellstruktur [61]*

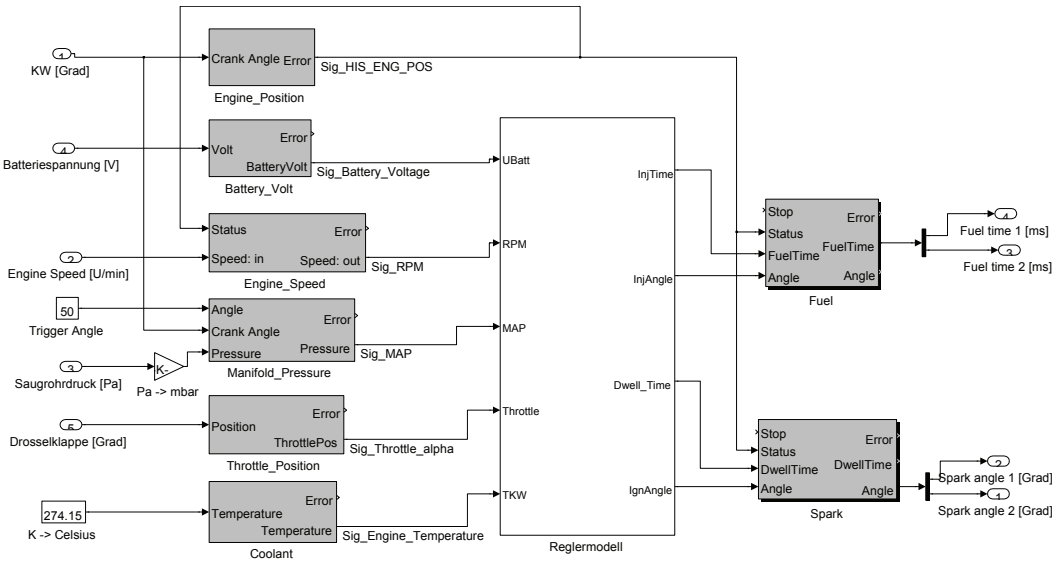

Abbildung 6.3: *Das Modell mit Verwendung der Abstraktionsschicht [61]*

Steuerungsmodell

Das zur Evaluierung verwendete Steuerungsmodell ist in Abbildung 6.3 dargestellt. Die grau gefärbten Blöcke der Modelle markieren Blöcke aus der VeRa-Toolbox, die wie in Kapitel 4.3.1 beschrieben Sensoren und Aktuatoren repräsentieren. Das in einer weißen Box dargestellte Subsystem stellt das eigentliche Steuerungsmodell dar, dessen Elemente für die Evaluierung jedoch nicht relevant sind und somit als *Black-Box* betrachtet werden können.

Sehr deutlich ist in Abbildung 6.3 das Prinzip der Verwendung von Sensoren und Aktuatoren als Schnittstelle zwischen Regler- oder Steuerungsmodell und dem in der Abbildung nicht dargestellten Streckenmodell zu erkennen. Die Aktuatormodelle besitzen jeweils Ausgänge, die dem Streckenmodell als Eingang dienen, während die Sensorblöcke jeweils mit Eingängen ausgestattet sind, die mit den Ausgängen des Streckenmodells verbunden sind. Die Verbindungen zwischen Steuerungsmodell und Streckenmodell werden ausschließlich über die Sensor- und Aktuatorrepräsentationen der VeRa Toolbox geführt. Somit ist sichergestellt, dass das Steuerungsmodell ohne Modifikation sowohl für die Simulation als auch für die Code-Generierung verwendet werden kann. Das dynamische Verhalten von Sensoren und Aktuatoren wird in der Simulation realitätsnah berücksichtigt. Ein weiterer Aspekt ist, dass in der Simulation der gleiche Code durchlaufen wird wie im Anwendungsfall Codegenerierung. Dadurch wird bereits auf der Modellierungsebene der auch später im Steuergerät verwendete Code getestet. Als Beispiel für die Simulation wird das Verhalten des

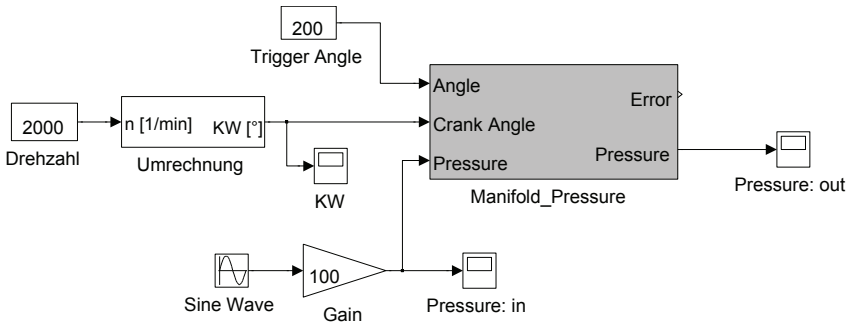

Abbildung 6.4: *Der Block Manifold Pressure in einer geeigneten Testumgebung [61]*

Saugrohrdrucksensors *Manifold Pressure* gezeigt. Der Block berechnet den Druck, der bei einem vorzugebenden Kurbelwinkel am Saugrohr anliegt. Dazu erhält der Block als Eingangswerte den Kurbelwinkel als Laufvariable, einen Winkelwer, bei dem der Druck gemessen wird, und den simulierten Saugrohrdruck. In Abbildung 6.4 ist der Block in einer Testumgebung für den Modultest dargestellt. Der Block *Umrechnung* berechnet aus der Drehzahl einen fortlaufenden Kurbelwinkelwert. Die Blöcke *Sine Wave* und *Gain* generieren den Druckverlauf beispielhaft durch eine modifizierte Sinuskurve. Der Block tastet diesen Druckverlauf über den Eingang *Pressure* an konstant vorgegebenen Weinkelwert von 200°KW ab. Um die Quantisierung des Sensors durch den Analog / Digital-Wandler richtig abzubilden, wird der abgetastete Analogwert des Drucks innerhalb des Manifold-Pressure-Blocks zunächst auf den entsprechenden Digital-Wert umgerechnet. Aus dem so quantisierten Wert wird der Druckwert für die Simulation anschließend äquivalent zum Anwendungsfall Codegenerierung berechnet.
In Abbildung 6.5 sind beispielhaft die Werte aus der oben beschriebenen Testumgebung dargestellt und verdeutlichen so die Zusammenhänge zwischen dem Eingangsdruck, dem Ausgangsdruck und dem Kurbelwinkel. Bildlich gesprochen friert der Block *Manifold Pressure* den Eingangsdruck an einem Kurbelwinkel von 200 Grad ein. Dieser Wert wird dann bis zur nächsten Abtastung als Eingangswert zur Berechnung des Ausgangsdruckes verwendet. Dadurch erhält der Ausgangsdruck einen „treppenförmigen“ Verlauf.

6.1.2 Ergebnisse

Modellerstellung

Die Modellerstellung erfolgte mit Hilfe des Entwicklungswerkzeuges VeRa. Dazu wurde unter VeRa Desk ein neues Projekt angelegt. Dabei wird die verwendete Hardware ausgewählt. In Abhängigkeit von dieser Auswahl werden später in der

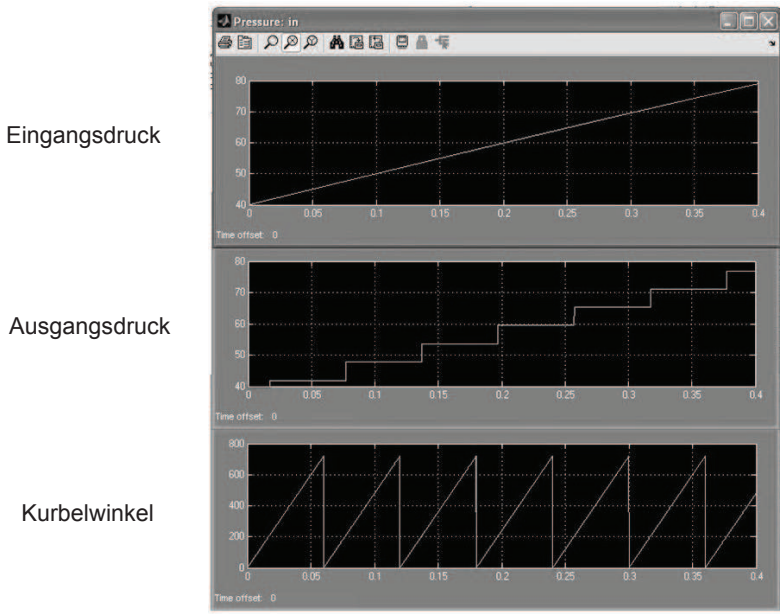

Abbildung 6.5: *Testergebnisse des Block Manifold Pressure [61]*

Simulation und in der Codegenerierung die entsprechenden Konfigurationen geladen und verwendet.
Das Steuerungsmodell und das Streckenmodell werden unter Matlab/Simulink erstellt und über die Sensor- und Aktuatorblöcke aus der VeRa Toolbox verknüpft. Auf dieser Ebene der Modellerstellung muss der Bediener dafür keine Parameter oder Konfigurationen setzen. Das bedeutet, das der Entwickler sich vollständig auf das Modell konzentrieren kann und kein Fachwissen zur verwendeten Hadware haben muss. Das Modell ist auf diese Weise vollständig von der Konfiguration entlastet, so dass die verwendete Hardware oder Hardware Elemente ausgetauscht werden können ohne das Modell zu ändern. In der Simulation werden jedoch Änderungen in der Konfiguration sichtbar, wenn z.B. zwei unterschiedliche Sensoren verwendet werden. Zur Verwaltung der Konfiguration wurde daher ein Konfigurationstool entworfen, das einen komfortablen Zugriff auf die Konfigurationsdatei erlaubt. Durch Sicherheitsmechanismen im Aufbau der Konfigurationsdateien, können Teile der Konfiguration für die Veränderung durch den Benutzer freigegeben werden, andere sind jedoch fest voreingestellt und können nur durch den Werkzeughersteller verändert werden. Dies stellt zum einen eine sehr große Flexibilität sicher, verhindert aber auch gleichzeitig Bedienungsfehler durch Anwender der Software.
Nach zufriedenstellender Simulation wird aus dem Steuerungsmodell automatisch Code erzeugt und in VeRa Desk für den Einsatz auf der Zielhardware kompiliert.

Die dafür notwendige Konfiguration wird ebenfalls aus den Konfigurationsdateien geladen. Nach der Übertragung auf das Steuergerät können Variablen und Parameter, die im Modellierungswerkzeug angelegt wurden, mit Hilfe eines Kalibrierwerkzeugs verstellt und gemessen werden. Die Auswahl des Kalibrierwerkzeugs ist dabei nahezu frei, da für die Ankopplung die standardisierte ASAM-MCD2 Schnittstelle verwendet wird (vgl. Abschnitt 4.5). Parameter, die über die Konfigurationsdatei festgelegt wurden sind nicht kalibrierbar. Das hat den Vorteil, dass Fehlbedienungen vermieden werden, es hat aber auch den Nachteil geringerer Flexibilität. Dies hat sich vor allem in frühen Phasen der Entwicklung als unkomfortabel herausgestellt, so dass in der nächsten Release ein System implementiert werde sollte, das die Verstellung von Konfigurationsparametern erlaubt.
Nachdem der generierte Code auf VeRa übertragen wurde, können mit Hilfe eines Kalibrierwerkzeuges Parameter zur Laufzeit kalibriert werden. Bei der ursprünglichen Implementierung wurden die abzubildenden Sensoren bzw. Aktuatoren über die elementaren Blöcke der Simulink-Bibliothek und die Blöcke der Prozessorschnittstellen modelliert. Dadurch konnte die Konfiguration zur Laufzeit des Sensors manipuliert werden. Dagegen schränkt die Weiterentwicklung die Sichtbarkeit auf die relevanten Parameter der Sensoren zur Laufzeit ein. Das heißt, es besteht keine Möglichkeit mehr die Konfiguration zur Laufzeit zu verändern. Die Vorteile der Weiterentwicklung sind ein Schutz vor Fehlkonfigurationen zur Laufzeit und eine effizientere Anwendung des Applikationswerkzeuges, da nur die für die Laufzeit relevanten Parameter sichtbar sind.

Code Effizienz

Das wesentliche Element der vorgestellten Architektur im Gegensatz zu einer manuellen Ankopplung des Steuerungsmodells an die Peripheriesysteme des Steuergeräts, wie A/D-Wandler, PWM Einheit, usw., ist die Abstraktionsschicht. Daher wurden in [61] die Auswirkungen der Abstraktionsschicht bezüglich der Performance in dem Anwendungsfall Codegenerierung untersucht. Dazu wird aus dem Modell der Motorsteuerung der Code einmal mit und einmal ohne Abstraktionsschicht generiert und anschließend verglichen. Zur Untersuchung der Performance werden statische sowie dynamische Größen betrachtet.
Zunächst wird die Speichernutzung als eine statische Größe betrachtet. Dazu werden die Speicherbereiche des *ROM (Read Only Memory)* und *RAM (Random Access Memory)* getrennt untersucht. Im ROM werden Funktionen, Globale Variablen und Konstanten abgelegt. Der Speicherbereich RAM enthält die initialisierten und die nicht initialisierten globalen Variablen. Die globalen Variablen des ROM dienen als Referenzwerte für die initialisierten globalen Variablen des RAM. Beim Start werden die initialisierten globalen Variablen des RAM vorbelegt, indem der entsprechende Speicherbereich mit den Referenzwerten des ROM in den RAM kopiert wird.

Speicherbereiche	Code Größe ohne Abstraktionsschicht	Code Größe mit Abstraktionsschicht
	belegte Speichergröße (in Byte)	belegte Speichergröße (in Byte)
ROM		
.text	142.368	160.952
.rodata	6.620	10.476
Σ	148.988	171.428
Δ		+22.440
RAM		
.sdata	180	184
.data	24.096	25.176
.sbss	324	360
.bss	18.480	18.636
.COMMON	1.880	1.940
Σ	44.960	46.296
Δ		+1.336

Tabelle 6.2: *Vergleich der Speichernutzung*

Die Auswirkung der Abstraktionsschicht auf die Speichernutzung, wird durch einen Vergleich der durch den Linker erstellten Linker Maps vorgenommen. Die Tabelle 6.2 zeigt die Auswertung der beiden Linker Maps. Dort ist die Speichernutzung der verschiedenen Speichertypen nach den Sektionen dargestellt. Die Sektionen erlauben es den generierten Code und die Variablen zu unterscheiden. Als Compiler wurde der GCC für PowerPC in der Version 2.95.3 verwendet. Zur Kompilierung der verschiedenen Ansätze wurden die gleichen Compiler Einstellungen verwendet.

Die Differenz in der Speichernutzung der verglichenen Modelle verdeutlicht den EInfluss der Abstraktionsschicht. Die Modelle mit der Abstraktionsschicht benötigen in jeweils für ROM und RAM einen größeren Speicherbereich gegenüber dem Modell ohne Abstraktionsschicht. Für den ROM beträgt der Unterschied der Speicherbelegung 22.440 Byte. Dabei wurde das Modell mit Abstraktionsschicht ohne Optimierung kompiliert und dient somit als Abschätzung der oberen Grenze für den Speicherzuwachs durch die Abstraktionsschicht. Der gesamte nutzbare Speicher im ROM beträgt 4096 kB. Das Modell ohne Abstraktionsschicht nutzt davon ca. 145 kB und das Modell mit der Abstraktionsschicht ca. 167 kB. Die größere Speicherbelegung von ca. 22 kB bedingt durch die Abstraktionsschicht bedeutet einen Zuwachs von 15,1 %. Im Verhältnis zu dem nutzbaren Gesamtspeicher macht der Zuwachs einen Teil von 0,5 % des Gesamtspeichers aus.

Analog beträgt der nutzbare Speicher des RAM ebenfalls 4096 kB. Von dem Modell ohne Abstraktionsschicht werden davon ca. 44 kB verwendet und von dem Modell mit der Abstraktionsschicht 45 kB. Das heißt, durch die Abstraktionsschicht werden 2,97 % mehr Speicher im RAM belegt. Im Verhältnis zu dem Gesamtspeicher macht dieser Zuwachs einen Teil von 0,03 % aus.
Der Speicherzuwachs in dem ROM und RAM gegenüber dem Modell ohne Abstraktionsschicht lässt sich durch die in der Abstraktionsschicht implementierten Funktionalität erklären. Durch die Abstraktionsschicht wurden 18 Softwaremodule neu hinzugefügt.

Als eine dynamische Größe wird die CPU-Auslastung zum Vergleich herangezogen. Zur Berechnung wird die Ausführungszeit des Modells gemessen und in das Verhältnis zur Modell-Ausführungsperiode gestellt:

$$CPU - Auslastung\ (in\ Prozent) = \frac{Modell\ Periode}{Modell\ Ausfuehrungszeit} * 100 \qquad (6.1)$$

Bei dem Modell ohne Abstraktionsschicht ist eine maximale Ausführungszeit von 3 ms und eine gemittelte Ausführungszeit von 2,72 ms gemessen worden. Das Modell mit Abstraktionsschicht benötigt maximal 4 ms und im Mittel 3,51 ms zur Ausführung.
Die Modellausführung benötigt also durch die Abstraktionsschicht eine längere Ausführungszeit. Ein Grund dafür ist die Anwendung des Schichten-Architekturmusters. Durch die Abstraktionsschicht kann der generierte Code nicht mehr direkt auf die VeRa-Hardware Treiber zugreifen, sondern muss nun den Zugriff über die Dienste der Schicht vornehmen. Dies ist im Vergleich zum direkten Zugriff mit zusätzlichen Funktionsaufrufen verbunden.

Simulations- und Messergebnisse

Nach der Simulation des geschlossenen Regelkreises entsprechend des oben beschriebenen Modells, wurden die Simulationsergebnisse am Motorprüfstand validiert. In dem hier exemplarisch dargestellten Testfall aus [61] werden statische Betriebspunkte simuliert und mit den Prüfstandswerten verglichen.
Zum Vergleich werden die Prüfstandsmessungen der Betriebszustände mit 4 bar effektivem Mitteldruck (entspricht einem Antriebsmoment von ca. 23,9 Nm) und den Drehzahlen von 1500 bis 6000 U/min in Abständen von 500 U/min betrachtet. Zunächst wird der für die Motorsteuerung wichtige Saugrohrdruck verglichen. Auf Basis dieses Wertes werden Einspritz- und Zündwinkel sowie die Einspritzzeit ermittelt. Der Saugrohrdruck wird als Absolutdruck gemessen. In Abbildung 6.6 sind die gemessenen und die simulierten Saugrohrdrücke in mbar bei einem effektiven Mitteldruck von 4 bar in Abhängigkeit von der Drehzahl dargestellt. Die beiden

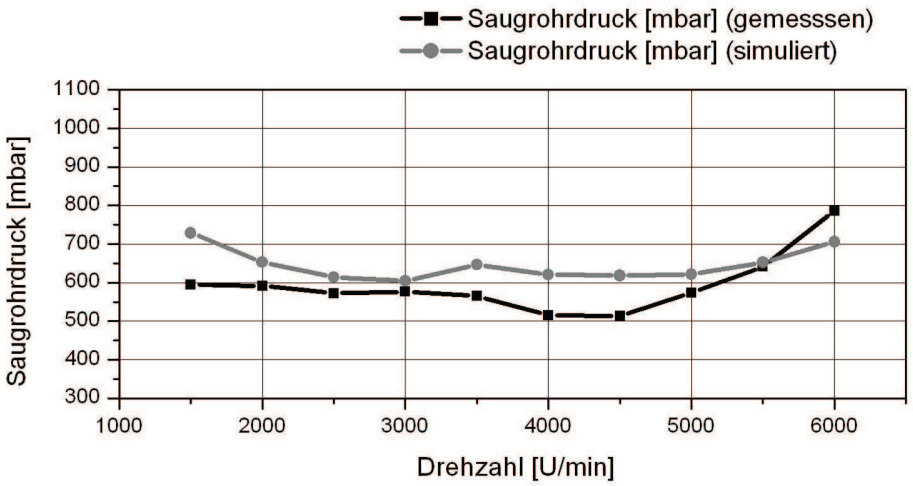

Abbildung 6.6: *Vergleich der gemessenen mit den simulierten Saugrohrdrücken bei einem effektiven Mitteldruck von 4 bar [61]*

Kurven sind im Verlauf ähnlich. Die maximale Abweichung von dem gemessenen Wert beträgt ca. 22 % bei einer Drehzahl von etwa 1500 U/min. In dem Punkt 5500 U/min liegt die minimale Abweichung bei etwa 2 %. Der Mittelwert der Abweichungen beträgt ca. 12 %.

In Abbildung 6.7 sind die gemessen und simulierten Luftmassenströme in kg/h dargestellt. Zusätzlich sind noch die dazugehörigen Drosselklappenstellungen in Prozent angegeben. Die Verläufe der simulierten Luftmassenströme und Drosselklappenstellungen folgen prinzipiell den gemessenen Werten. Bei den Luftmassenströmen lässt sich eine maximale Abweichung von den gemessenen Werten von ungefähr 41 % bei 1500 U/min beobachten. Die minimale Abweichung beträgt ca. 2 % bei einer Drehzahl von 2000 U/min. Der Mittelwert der Abweichungen ist ungefähr 24 %. Die Drosselklappenstellung beeinflusst maßgeblich den im Saugrohr maximal möglichen Luftmassenstrom. Der Kurvenverlauf der simulierten Luftmassenströme ist bezüglich des simulierten Öffnungsquerschnittes der Drosselklappe plausibel.

Die dargestellten Abweichungen in der Simulation gegenüber den Prüfstandsmessungen sind auf die Unzulänglichkeiten des Modells zurückzuführen und entstehen nicht aufgrund der hier vorgestellten Entwicklungswerkzeugs. Detaillierte Messergebnisse und eine intensive Diskussion der Messwertabweichungen sind in [61] dargestellt.

Es ist somit gezeigt, dass das hier vorgestellte Entwicklungswerkzeug für die Entwicklung von Motorsteuergeräten geeignet ist. Der hier dargestellte Entwicklungsstand kann als A-Muster interpretiert werden, bei dem die geforderte Funktionalität vollumfänglich gegeben ist. In weiteren Projektstufen können von diesem Entwicklungsstand

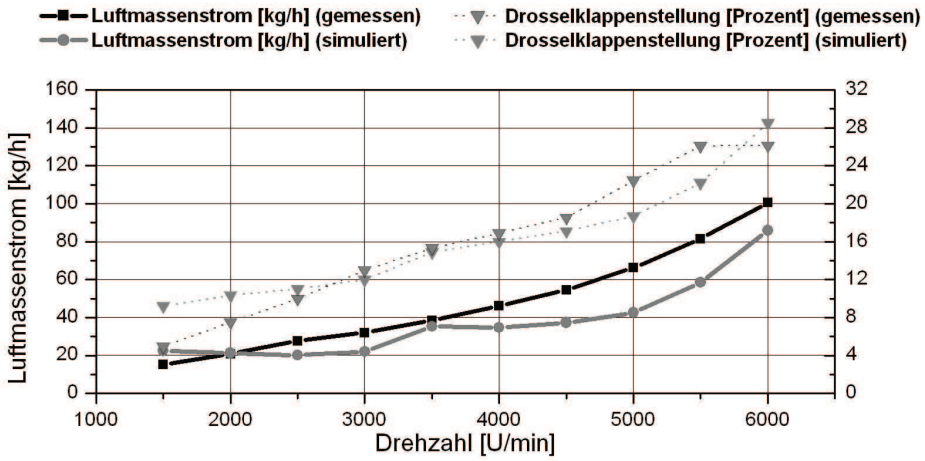

Abbildung 6.7: *Vergleich der gemessenen mit den simulierten Luftmassenströmen bei einem effektiven Mitteldruck von 4 bar [61]*

die weiteren Iterationen hin zu B-Muster, C-Muster und Seriensteuergerät verfolgt werden.

6.2 OSEK Tasks mit unterschiedlicher Zeitbasis

Insbesondere Motor- und Getriebsteuergeräte verwenden neben zeitbasierten Berechnungen auch winkelbasierte Berechnungen. Das bedeutet, dass bestimmte Berechnungen z.B. einmal je Umdrehung ausgeführt werden, während andere mit einer konstanten zeitlichen Periode ausgeführt werden (vgl. hierzu auch Abschnitt 4.5 und insbesondere Abbildung 4.13). Die Architektur des hier vorgestellten Entwicklungswerkzeugs unterstützt diese Anforderung, indem in der Modellierungsumgebung unterschiedliche Betriebssystem Tasks definiert werden können, die entweder winkel- oder zeitbasiert ausgeführt werden können. Der Betriebssystemstandard OSEK erlaubt es dafür unterschiedliche Tick-Basen zu deklarieren zu denen die einzelnen Tasks synchron ausgeführt werden. Im hier vorliegenden Fall wurde eine Tick-Basis definiert, die zeitbasiert inkrementiert wird, und eine zweite, die mit jedem Grad der Umdrehung inkrementiert wird. Zudem wird jedem Task eine Priorität zugeordnet, so dass sich sehr viele Zustände und Nebenläufigkeiten ergeben, deren Verhalten getestet werden muss.
Durch das in dieser Arbeit vorgestellte System ist es möglich, diese Effekte bereits in der Simulation zu testen. Daher ist es das Ziel dieser Fallstudie zu zeigen, dass

das Verhalten in der Simulation das Verhalten der Software auf der Zielplattform abbildet.

6.2.1 Versuchsaufbau

Als Versuchsaufbau wird ein einfaches Modell bestehend aus drei Tasks unter Matlab/Simulink erstellt (vgl. Abbildung 6.8). Die Konfiguration der Betriebssystemblöcke ist in der XML Konfiguration hinterlegt. Dabei ist *TaskA* als kurbelwinkelbasierter Task definiert, während *TaskB* und *TaskC* als zeitbasierte Tasks mit unterschiedlicher Periode definiert sind.
Das Betriebssystem wird durch das Platzieren des Blockes *OSEKScheduler* aktiviert. Dieser Block besitzt einen Statuseingang und einen Eingang für den Kurbelwinkel (*Crank*). Dieser Eingang wird ausschließlich für die Simulation benötigt, da die Information über den aktuellen Kurbelwinkel des Motors vom Motormodell berechnet wird. In dem hier gezeigten Testmodell wird der Kurbelwinkel durch einen sägezahnförmigen Kurvenverlauf gespeist, der von 0°KW bis 720°KW linear ansteigt, um anschließend wieder bei 0°KW zu beginnen. Für jeden Task wird ein *Task*-Block im Modell platziert, der jeweils einen Triggerausgang für ein Subsystem enthält. Die Berechnungen des jeweiligen Tasks sind in jeweils einem Subsystem *TaskN_Func* untergebracht. Am Eingang werden alle Tasks durch einen sinusförmigen Kurvenverlauf gespeist. Der Augang der jeweiligen Task-Subsysteme und ihr Eingangssignal werden zur Darstellung auf einen Oszilloskop-Block geführt.
Die Berechnungen innerhalb der drei Task-Subsysteme ist in Abbildung 6.9 dargestellt. In jedem Task Subsystem muss ein *Begin*- und ein *End*-Block platziert werden, die den Beginn und das Ende der Berechnungssequenz markieren. Die Semaphor-Aufrufe *Memory Get* und *Memory Release* sind in diesem Fall so platziert, dass die gesamte Berechnung nur dann stattfindet, sobald der Task exklusiven Zugriff auf den durch den Semaphor geschützten Speicherbereich erhält.Die eigentliche Berechnung ist in einem weiteren Subsystem untergebracht, das durch den *Do*-Block getriggert und ausgeführt wird. Hier ist zu sehen, dass der Eingangswert (*In1*) an den Ausgang (*out1*) gegeben wird, sobald die Funktion ausgeführt wird.
Nach erfolgreicher Simulation wird das Modell auch hier auf die Zielplattform übertragen. Am Steuergerät wird ein Motorsignalgenerator angeschlossen um ein drehzahlvariables Kurbelwinkelsignal zu erzeugen. Der Zugriff auf die berechneten Werte erfolgt durch das Kalibrierwerkzeug ATI Vision, das über die CAN Schnittstelle mit dem Steuergerät verbunden ist.

6.2.2 Ergebnisse

Ziel dieser Evaluierung war es nachzuweisen, dass das Entwicklungswerkzeug den OSEK Standard unterstützt. Dabei wurde neben der Implementierung auf dem

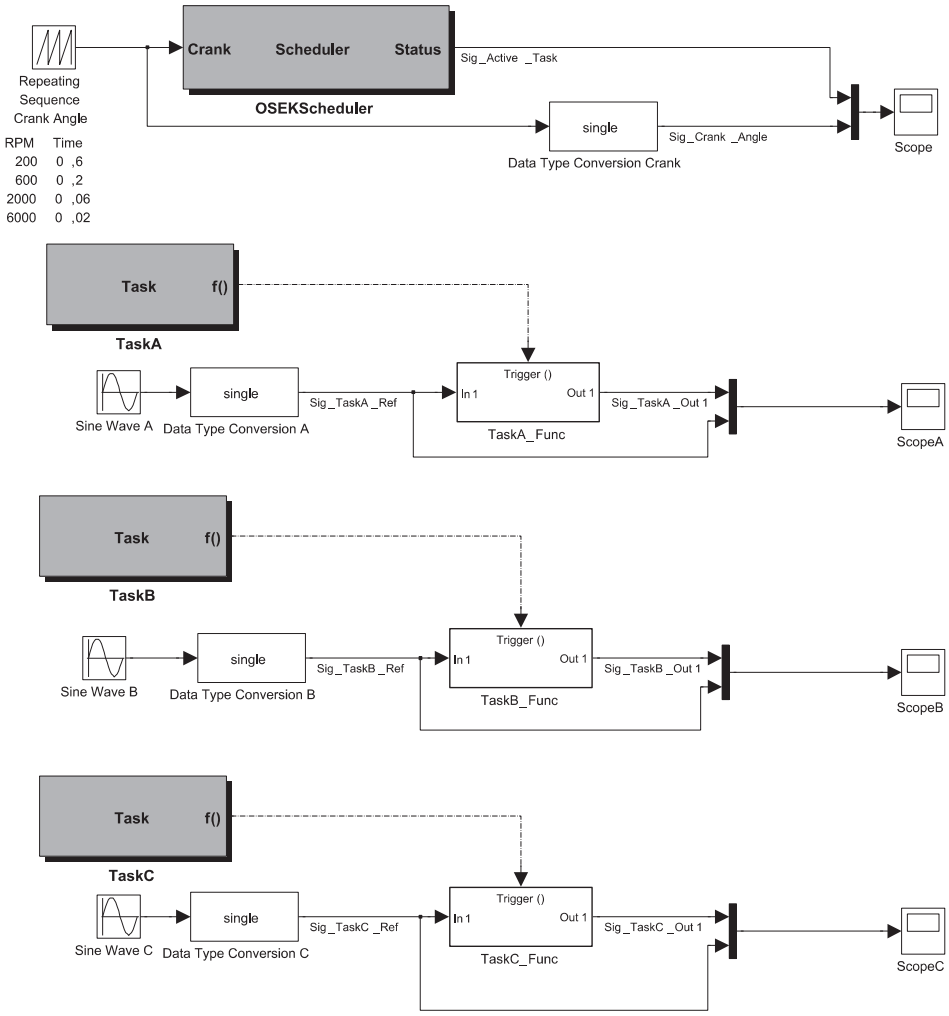

Abbildung 6.8: *Versuchsmodell bestehend aus drei Tasks*

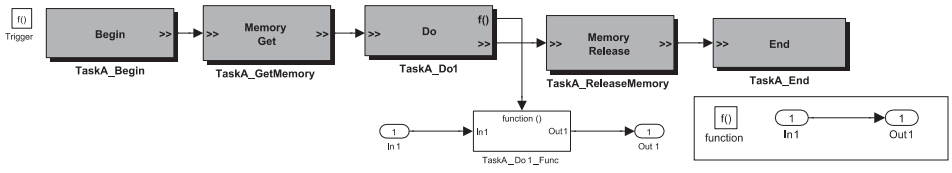

Abbildung 6.9: *Berechnung im Subsystem aller Tasks*

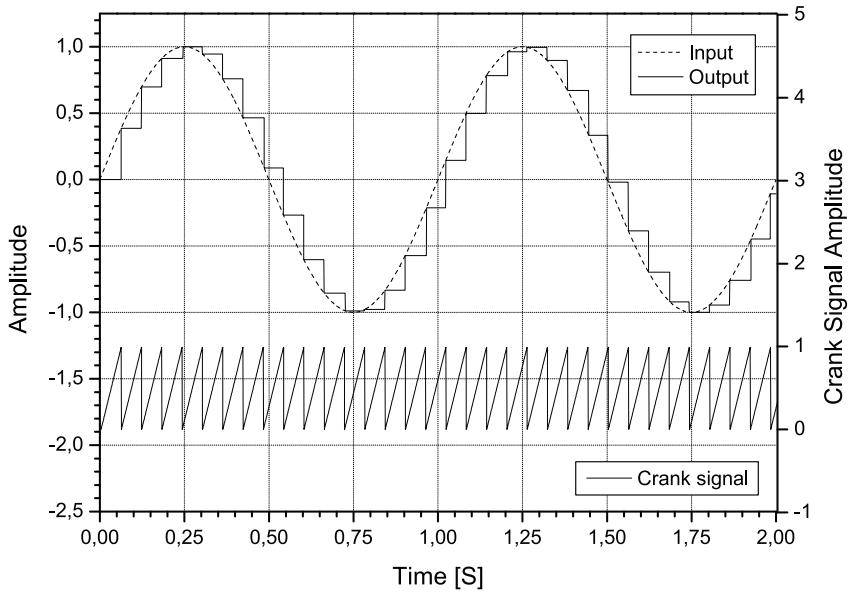

Abbildung 6.10: *Simulationsergebnis des winkelbasierten Tasks A bei einer Drehzahl von 2000 U/min*

Steuergerät vor allem auch gezeigt, dass der Einfluss des Betriebssystems auf dem Zielsystem auch in der Simulation berücksichtigt wird. Dies ist entscheidend, um die Modelle aus der Simulationphase auch für die Codegenerierung verwenden zu können.
Das Simulationsergebnis ist in Abbildung 6.10 dargestellt, während das Testergebnis auf der Zielplattform in Abbildung 6.11 dargestellt ist. Gut zu erkennen ist, dass der Task jeweils bei einem Kurbelwinkel von 0° ausgeführt wird und zu diesem Zeitpunkt den Eingangswert übernimmt. Das Testergebnis der Code Generierung zeigt prinzipiell das gleiche Resultat. An manchen Punkten jedoch wird der Eingangswert nur leicht zeitverzögert übernommen. Dieser Effekt liegt an der geringen Datenerfassungsrate von lediglich 20ms. Hier kann es passieren, dass ein Datenpaket genau dann versendet wird, wenn die Daten am Eingang bereits aktualisiert, die Daten am Ausgang jedoch noch nicht aktuell sind. Diese werden erst mit dem nächsten Datenpaket übernommen, das jedoch erst 20ms später versendet wird.
Eine detaillierte Untersuchung und weitere Test- und Simulationsergebnisse sind in [18] dargestellt.

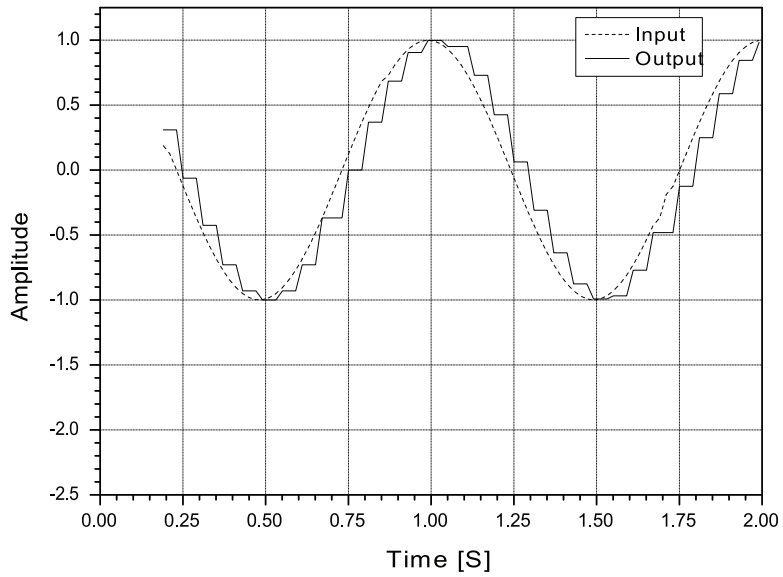

Abbildung 6.11: *Testergebnis des winkelbasierten Tasks A mit generierten Codes auf der Zielplattform bei einer Drehzahl von 2000 U/min*

6.3 Luftpfadregelung in einer HIL Umgebung

Die Unterstützung des modellbasierten Designprozesses ist das zentrale Ziel des Entwicklungswerkzeugs. Dass es möglich ist mit Hilfe des beschriebenen Ansatzes Modelle in eine Simulationsumgebung zu testen wurde bereits gezeigt. Um nun aber auch die Hardware inklusive des modellbasiert entwickelten Reglers zu testen gibt es je nach Testzweck unterschiedliche Möglichkeiten. Wenn neben der Software auch die Hardware des Steuergerätes im Fokus des Tests liegt, so wird man eine aufwändige Testumgebung aufbauen müssen, mit der an den Eingängen des Steuergerätes Sensoren simuliert werden und an den Ausgängen des Steuergerätes realistische elektrische Lasten zur Verfügung stellt. Das Streckenmodell muss in dieser Testumgebung in Echtzeit ausgeführt werden. Soll hingegen lediglich die Software in Kombination mit der Hardware untersucht werden, so ist es ausreichend die Software auf dem Zielsystem auszuführen, ohne jedoch die Hardwareschnittstellen zu berücksichtigen. Auf diese Weise ist es möglich Ausführungzeiten und Berechnungsergebnisse auf der Zielhardware zu untersuchen. Das hier verwendete XCP Protokoll bietet hierfür die Voraussetzungen. Insbesondere für die Entwicklungsphase stellt dies eine interessante Möglichkeit dar *Hardware-In-the-Loop (HIL)* [1] Untersuchungen durchzuführen.
Der hier vorgestellte Testfall wurde gemeinsam mit dem Institut für Regelungstechnik (IRT) der RWTH-Aachen durchgeführt. Eine detaillierte Beschreibung wurde

in [21] veröffentlicht.

6.3.1 XCP Protokoll

Als Weiterentwicklung des bekannten CCP-Standards stellte der ASAM-Verein[1] im Jahr 2003 den XCP-Standard vor [5]. Die wesentliche Änderung von XCP gegenüber CCP ist die Trennung von Protokollschicht und Transportschicht. Während bei CCP der CAN-Bus als ausschließliche Transportschicht genutzt wurde, sind bei XCP bis heute die Bussysteme CAN, Ethernet, USB, Flexray und SxI spezifiziert [69]. Eine zusätzliche Erweiterung gegenüber CCP ist die Möglichkeit, Stimulidaten synchron zu übertragen. Während bei CCP zur Datenerfassung lediglich der synchrone Datentransfer vom Slave zum Master spezifiziert war (DAQ), ermöglicht XCP auch den synchronen Datentransfer vom Master zum Slave (STIM). Somit können in einem vorgegebenen Zeitraster Messwerte von einem Steuergerät zu einem anderen Echtzeitsystem übertragen werden und es können Stellwerte ebenfalls in einem Zeitraster von dem Echtzeitsystem wieder zurück an das Steuergerät übertragen werden. Diese Funktionalität ermöglicht somit sowohl den Aufbau von Bypass-Systemen [50] als auch von Hardware-in-the-Loop (HIL) Systemen über eine XCP-Verbindung [53].
Für den Aufbau eines HIL-Systems, welches das Steuergerät als XCP-Slave einbindet, stellt das Echtzeitsystem, welches das Streckenmodell rechnet, den XCP-Master dar. Während der Initialisierung konfiguriert der Master die sogenannten Events des Slaves so, dass dieser in DAQ-Nachrichten zyklisch Messwerte an den Master überträgt. Nach Abschluss der Initialisierung wartet der Master zunächst auf eine Datenübertragung mit Messwerten (DAQ), die vom Slave gesendet wird (vgl. Abbildung 6.12). Anschließend führt der Master einen Berechnungsschritt durch und sendet das Ergebnis mit einer Datenstimuli-Nachricht (STIM) an den Slave. Dieser empfängt die Nachricht und übernimmt die Daten nach Beendigung des aktuellen Berechnungsschrittes. Das Steuergerät übernimmt somit die Synchronisation mit dem HIL-System.

6.3.2 Versuchsaufbau

In dem hier besprochenen Versuch wurde ein modellprädiktiver Regler zur Luftpfadregelung eines Dieselmotors auf dem Entwicklungswerkzeug VeRa implementiert. Das Streckenmodell des zugehörigen Dieselmotors wurde auf dem Echtzeitsystem MicroAutoBox der Fa. dSpace implementiert. Der Datenaustausch der Modellein- und -ausgänge wurde über eine XCP-Verbindung realisiert. Für die Kalibrierung verfügt VeRa wie oben beschrieben über eine XCP-Slave-Schnittstelle, die den CAN-Bus als Transportschicht nutzt und sowohl die DAQ- als auch die STIM-Funktionalität bereitstellt. Mit der zusätzlichen XCP-Toolbox ist die MicroAutoBox in der Lage,

[1] ASAM Verein: http://www.asam.de

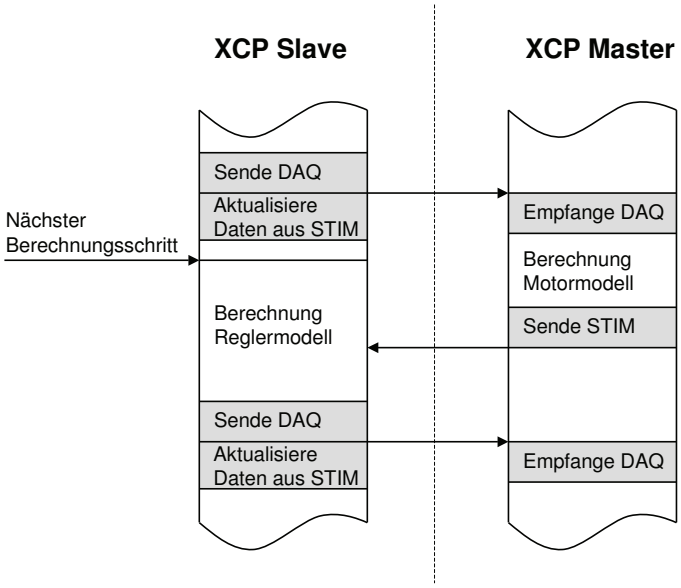

Abbildung 6.12: *HIL-System mit XCP-Verbindung [21]*

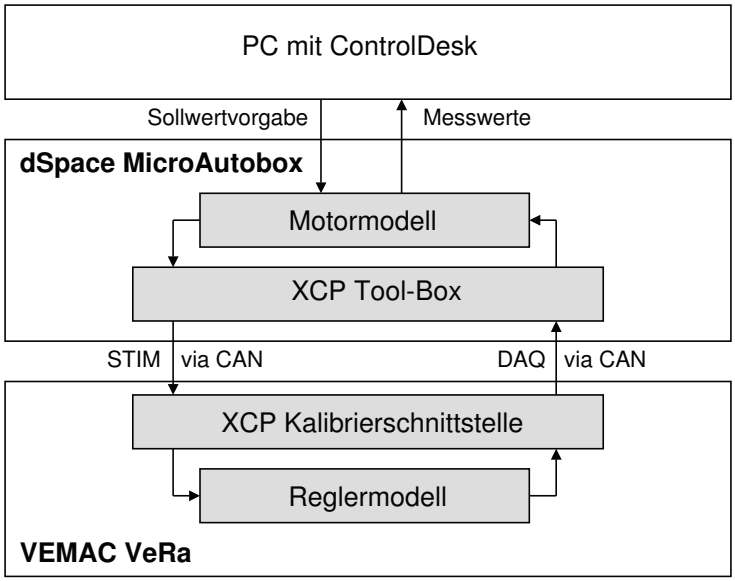

Abbildung 6.13: *Schematischer Versuchsaufbau [21]*

eine XCP-Master-Schnittstelle über den CAN-Bus zur Verfügung zu stellen [21].
Der Regler, der in dieser Versuchsanordnung verwendet wurde regelt die VTG[2]-Stellung des Turboladers und gleichzeitig den Öffnungsgrad des AGR[3]-Ventils. Diese Stellgrößen werden in einer XCP-DAQ-Meldung über den CAN-Bus an das Motormodell übergeben (vgl. Abbildung 6.13). Diese Werte dienen dem Motormodell als Eingangsgrößen um daraus die sich ergebende Frischluftmasse und den Ladedruck zu errechnen. Diese Werte werden dann als STIM-Nachricht ebnfalls über den CAN-Bus zurück an das Reglermodell übergeben. Der Regler bestimmt anhand dieser Größen dann wiederum die neuen Werte für die Stellgrößen. Über einen an die MicroAutoBox angeschlossenen PC werden von der Software ControlDesk Sollwertänderungen an das Motormodell übertragen. Für deren Übertragung wird ebenfalls eine STIM-Nachricht verwendet. Die Reaktion auf eine Sollwertänderung kann mit ControlDesk beobachtet und aufgezeichnet werden, da Messwerte von der MicroAutoBox an die Software übertragen werden können.

[2] **V**ariable **T**urbinen **G**eometrie
[3] **A**bgas**R**ückführ**R**ate

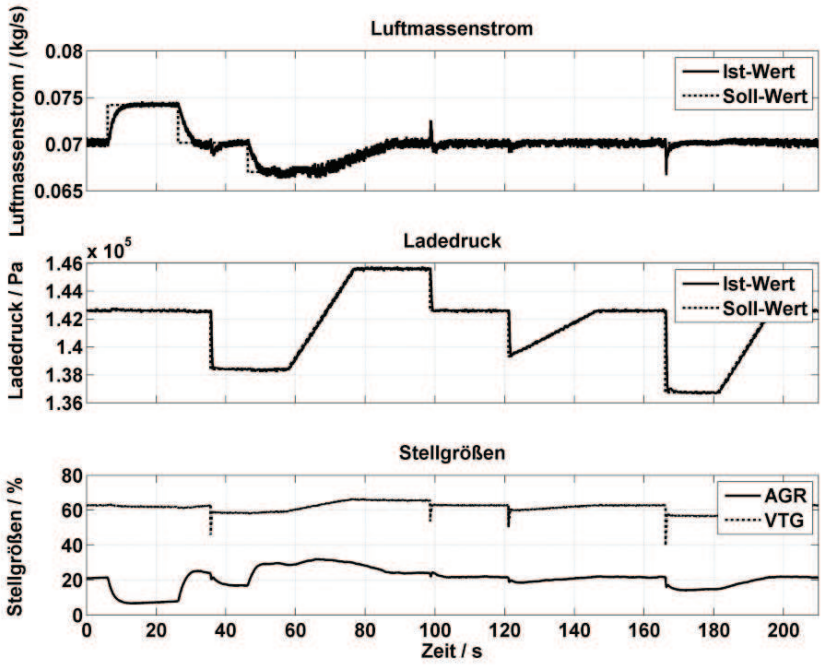

Abbildung 6.14: *Regelergebnisse im HIL-Test mit acht Abtastschritten bei N = 2000 1/min und* $\dot{m}_B = 0.002$ *kg/s [21]*

6.3.3 Ergebnisse

In Abbildung 6.14 sind die Regelergebnisse des HIL-Tests dargestellt (Sprungantwort Luftmasse (oben), Sprungantwort Ladedruck (unten), Stellgrößen (unten)). Gut zu erkennen sind die Sprungfolgen und Rampen, die als Sollwerte für den Regler dienten. Charakteristisch für diese Art der Regelung ist es, dass um einen Sollwertsprung bei einer Eingangsgröße wie z.B. dem Ladedruck, beide Stellgrößen durch den Regler verändert werden. Dies ist erforderlich, da es sich bei dieser Regelstrecke um ein gekoppeltes System handelt.
Anhand der Ergebnisse konnte gezeigt werden, dass sich der Regler gut für die Luftpfadregelung eines Dieselmotors eignet [21]. Im Vergleich zu bisher für diese Aufgabe eingesetzten gekoppelten Regler zeichnet sich der hier untersuchte Regler besonders durch die einfache und automatisierbare Parametrisierung aus. Allerdings führt der komplexe Berechnungsalgorithmus zu einem erhöhten Rechenaufwand, der bei der Reglerauslegung früh berücksichtigt werden muss. Mit dem hier gezeigten Versuchsaufbau wurde gezeigt, dass dies durch die frühe Einbeziehung von HIL-

Versuchen schon während der Prototypenphase der ECU-Entwicklung gelingt.

6.4 Ankopplung von Targetlink mit bestehender Architektur

Die Voraussetzung für den Einsatz einer Werkzeugkette in der Entwicklung von sicherheitskritischen Anwendungen ist die Qualifizierung nach dem gefordertem Sicherheitsstandard wie z.B. ISO 26262 [30] oder IEC 61508 [26]. Dabei ist es im Sinne einer kosteneffizienten Entwicklung sinnvoll in der Werkzeugkette auf Komponenten zurückzugreifen, die bereits für diesen Einsatzzweck qualifiziert sind. Dies gilt beispielsweise für Targetlink[4] als Codegenerator, wenn gleichzeitig ein Softwareentwicklungsprozess aufgesetzt wird, der an den durch den Codegenerator betroffenen Teilen einem durch die Fa. dSpace entwickeltem Referenzworkflow entspricht [11] [8]. Zentrales Element ist dabei die Validierung von Testergebnissen aus der Model-In-the-Loop (MIL) Phase durch Tests bei dem der Code des Steuerungsmodells auf dem Zielsystem gerechnet wird. Darüberhinaus fordern Sicherheitsstandards unter anderem auch die Einhaltung von Codierrichtlinien. Im Falle der Automobilindustrie sind dies in der Regel die Richtlinien der Motor Industry Software Reliability Association (MISRA)[5]. Es handelt sich hierbei um eine von der Automobilindustrie gemeinschaftlich finanzierte Vereinigung, um Standards und Richtlinien für die Entwicklung von Elektronik und Software für sicherheitskritische Anwendungen zu definieren. Die Unterstützung eines entsprechenden Codegenerators, wurde daher auch in die Anforderungen für die modellbasierte Werkzeugkette aufgenommen.
Weiterhin wird auch der einfache Austausch des Simulations- und Modellierungswerkzeug gefordert. Dies beinhaltet im wesentlichen Maße auch den zu verwendenden Codegenerator. Die Anbindung von Targetlink an die Werkeugkette dient somit auch der Evaluation dieser Anforderung.

6.4.1 Realisierung

Für den Wechsel des Codegenerators musste die Repräsentation der Sensor- und Aktuatorbibliothek `VeRa Toolbox` im Modellierungswerkzeug ausgetauscht werden (vgl. Abbildung 6.15). Die Bibliothek beinhaltet für jeden Sensor und jeden Aktuator einen Block und bildet somit die Schnittstelle zur VeRa Abstraktionsschicht (vgl. Abschnitt 4.3.2). Im Anwendungsfall Simulation ruft ein Block der Bibliothek die entsprechende DLL-Funktion der VeRa Abstraktionsschicht auf, während im Anwendungsfall der Codegenerierung ein entsprechender Funktionsaufruf im Code generiert werden muss. Die Realisierung dieser Funktionalität ist bei den hier betrachteten Codegeneratoren *Realtime Workshop* und *Targetlink* jeweils unterschiedlich.

[4] http://www.dspace.de
[5] http://www.misra.org.uk

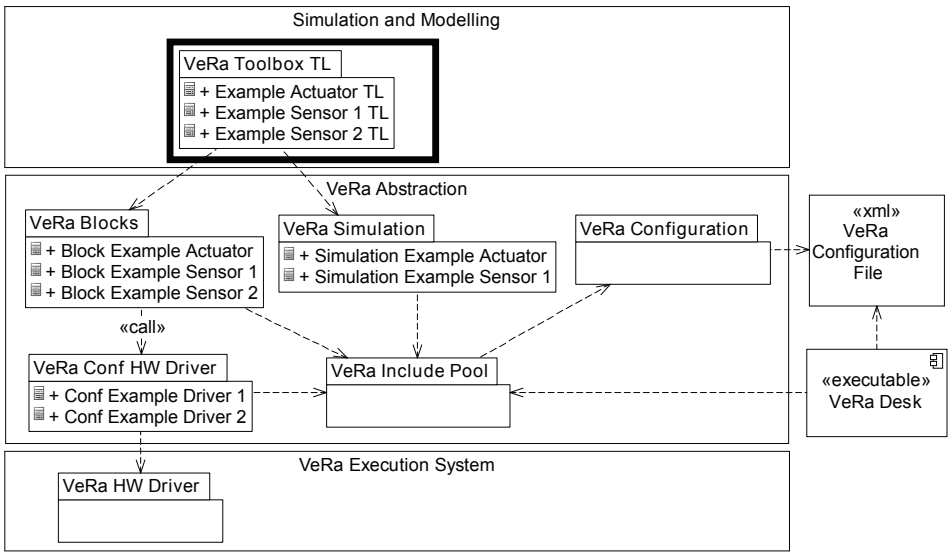

Abbildung 6.15: *Klassendiagramm der VeRa Abstraktionsschicht. Die VeRa Toolbox, die für den Einsatz des Code-Generators Targetlink geändert wurde, ist schwarz umrandet.*

Bei dem in der Regel verwendeten Realtime Workshop muss für den Aufruf der DLL-Funktion in der Simulation eine sogenannte *S-Function* hinterlegt werden, die es ermöglicht entsprechenden C-Code in der Simulation zu verwenden. Für die Codegenerierung verwendet Realtime Workshop für jeden Block ein Skript, welches in der proprietären Simulink-Skriptsprache TLC verfasst ist. Im Gegensatz dazu werden bei Targetlink sogenannte *custom code*-Blöcke verwendet. Hier wird der Code für die Simulation hinterlegt, der weitgehend aus den vorhandenen S-Functions übernommen werden kann. In Tabelle 6.3 ist dies an einem Beispielblock dargestellt. Gezeigt wird der Code der bei jedem Simulationsschritt des Modells aufgerufen wird. Im Beispiel stammt die Funktion `OUTPUT_FuelOutput` aus der entsprechenden DLL des Bibliothekblocks. Zusätzlich wird in den *custom code*-Blöcken auch der Code für die Codegenerierung hinterlegt. Dieser Code wird bei Targetlink ebenfalls in C geschrieben, so dass die Übernahme aus den TLC Skripten etwas aufwändiger ist (vgl. Tabelle 6.4). Dennoch ist auch im hier gezeigten Beispiel die Generierung des Funktionsaufrufes von `VERA_FUEL_Output` die einzige Funktion, die aus der VeRa Blibliothek aufgerufen werden muss. Die Übertragung der Bibliothek gelingt also durch die in der Softwarearchitektur implementierten Modularisierung in Form der Abstraktionsschicht ohne großen Aufwand. Neben der Bibliothek muss auch das Ausführungssystem an den geänderten Codegenerator angepasst werden. Dies

Realtime Workshop

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
  int_T j;
  VeraErrorType etyp;

  /* input */
  InputRealPtrsType input_stop = ssGetInputPortRealSignalPtrs(S, 0);
  InputRealPtrsType input_status = ssGetInputPortRealSignalPtrs(S, 1);
  InputRealPtrsType input_fltime = ssGetInputPortRealSignal(S, 2);
  InputRealPtrsType input_angle = ssGetInputPortRealSignal(S, 3);

  /* output */
  FLOAT64* error = ssGetOutputPortRealSignal(S, 0);
  FLOAT64* output_fltime = ssGetOutputPortRealSignal(S, 1);
  FLOAT64* output_angle = ssGetOutputPortRealSignal(S, 2);

  /* Output: Fuel */
  for (j = 0; j < conf_fuel->Num_Injectors; j++)
  {
   etyp = OUTPUT_FuelOutput( *input_fltime[j], *input_angle[j],
         *input_stop[0], *input_status[0], &output_fltime[j],
         &output_angle[j]);
   *error = etyp;
  }
}
```

Targetlink

```
/* flp_output_begin */

  int j;
  VeraErrorType etyp;

  for (j = 0; j < conf_fuel->Num_Injectors; j++)
  {
    etyp = OUTPUT_FuelOutput(fuelTimeIn[j], angleIn[j],
           stop, status, &fuelTimeOut[j], &angleOut[j]);
    error = (Float64)etyp;
  }
/* flp_output_end */
```

Tabelle 6.3: *Vergleich von Code für die Simulation an dem Beispielblock Fuel*

Realtime Workshop

```
%function Outputs(block, system) Output
 %% Rename S-Function parameter for easy reference

 %assign error = LibBlockOutputSignal(0, "", "", 0)
 %assign Blockname = SFcnParamSettings.Blockname

 /************************
  ** FUEL '%<Blockname>'
 *************************/
 %assign stop = LibBlockInputSignal(0, "", "", 0)

 // Update each fuel channel
 %assign rollVars = ["u2", "u3"]

 %roll sigIdx=RollRegions, lcv=RollThreshold, block, "Roller", rollVars

  %assign fltime = LibBlockInputSignal(2, "", lcv, sigIdx)
  %assign angle = LibBlockInputSignal(3, "", lcv, sigIdx)
  %<error> = VERA_FUEL_Output( (UINT8)(%< lcv == "" ? sigIdx : lcv>),
             (UINT16)%<fltime>, (FLOAT32)%<angle>, (UINT8)%<stop>);

 %endroll
%endfunction
```

Targetlink

```
/* fxp_output_begin */

  // Update each fuel channel
  UInt8 i1;
  for (i1=0; i1 < Fuel.Num_Injectors; i1++) {
    error = VERA_FUEL_Output( i1, fuelTimeIn[i1], angleIn[i1], stop);
  }

/* fxp_output_end */
```

Tabelle 6.4: *Vergleich von Code für die Codegenerierung an dem Beispielblock Fuel*

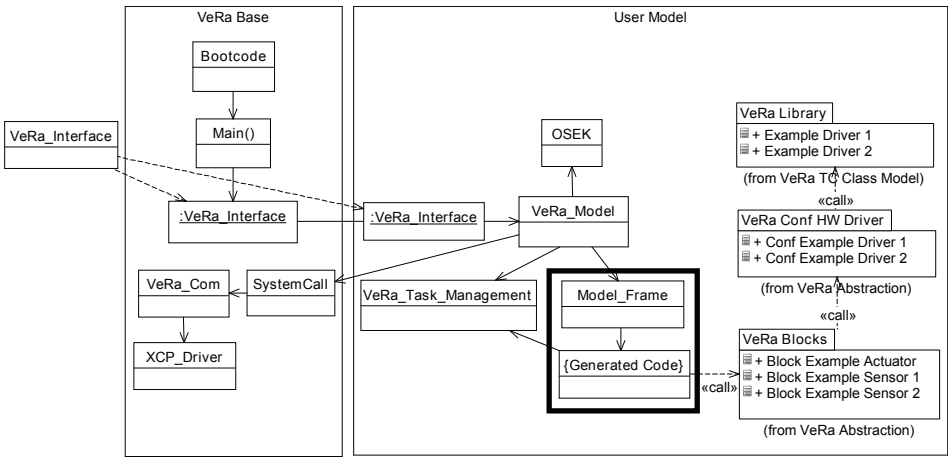

Abbildung 6.16: *Klassendiagramm des VeRa Ausführungssystems. Das Aufrufmodul sowie der mit Targetlink generierte Code sind schwarz umrandet.*

geschieht durch Austausch des Aufrufmoduls `VeRa_Frame` (vgl. Abschnitt 4.5.2). Der Codegenerator erzeugt für jedes Modell Funktionen mit gleicher Bezeichnung, jedoch modellabhängig unterschiedlichem Inhalt. Die Aufgabe des Aufrufmoduls ist es diese generierten Funktionen in der richtigen Reihenfolge aufzurufen. Da die Bezeichnungen der erzeugten Funktionen bei Targetlink anders als bei Realtime Workshop gewählt wurden, muss das Aufrufmodul einmalig angepasst werden. Je nach verwendeten Codegenerator muss auch das Makefile für den Kompiliervorgang angepasst werden, indem das für den jeweiligen Codegenerator passende Aufrufmodul kompiliert und gebunden wird. In Abbildung 6.16 ist das Klassendiagramm des VeRa Ausführungssystems in für Targetlink geänderter Form dargestellt.

6.4.2 Versuchsaufbau

Für die Untersuchung des Targetlink-Codegenerators wird das gleiche Modell wie in der Evaluierung für den Anwendungsfall Motorsteuerung verwendet (vgl. Abschnitt 6.1). Dazu muss das Modell zunächst konvertiert werden um die zuvor verwendeten Matlab/Simulink Blöcke durch die entsprechenden Targetlink-Blöcke zu ersetzen. Die Sensor- und Aktuatorblöcke müssen durch die entsprechenden Blöcke der für Targetlink geänderten Bibliothek ersetzt werden. Für den Vergleich werden die gleichen Simulationen durchgeführt wie sie zuvor mit der Realtime Workshop Bibliothek. Weiterhin wird der aus dem Modell durch Targetlink erzeugte Code kompiliert und mit dem Code verglichen, der durch Realtime Workshop erzeugt wurde.

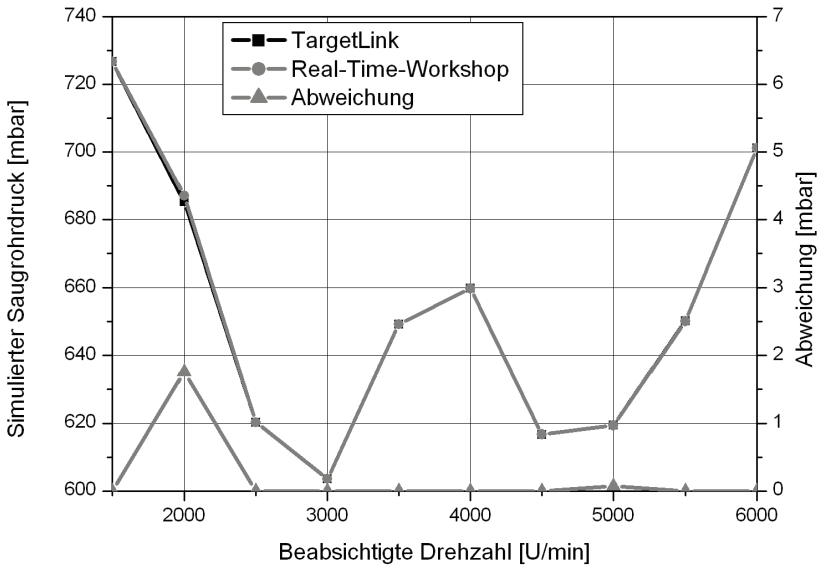

Abbildung 6.17: *Vergleich der Targetlink und der Simulink Simulationsergebnisse bei einem effektiven Mitteldruck von 4bar.*

6.4.3 Ergebnisse

Wie bereits erwähnt, wurde das hier betrachtete Steuerungsmodell für den Einsatz von Targetlink konvertiert. Zudem musste die Sensor-/ Aktuator-Bibliothek ebenfalls angepasst werden. Um zu bestätigen, dass diese Konvertierung unwesentliche Auswirkungen auf das Simulationsergebnis haben, wurden einige Simulationen durchgeführt und mit den ursprünglichen Ergebnissen verglichen. Als Beispiel sind in Abbildung 6.17 die Ergebnisse der bereits in Abschnitt 6.1.2 vorgestellten Simulation des Saugrohrdrucks bei konstantem Mitteldruck von 4 bar und unterschiedlichen Drehzahlen dargestellt. Aufgetragen sind die Simulationsergebnisse des bereits in Abschnitt 6.1 vorgestellten und des für Targetlink konvertierten Modells. Zusätzlich ist die absolute Abweichung des Targetlink Modells gegenüber dem Simulink Modell aufgetragen. Dabei beträgt die absolute Abweichung maximal 1,8 mbar, was einem relativen Fehler von ca. 0,2% entspricht. Dieser Fehler entsteht aufgrund der unterschiedlichen Solver und der Fehlerfortpflanzung, die sich aus dem geschlossenen Regelkreis zwischen Motor- und Steuerungsmodell ergibt. Die Abweichungen resultieren offensichtlich aus unterschiedlichen Ergebnissen bei der Interpolation im

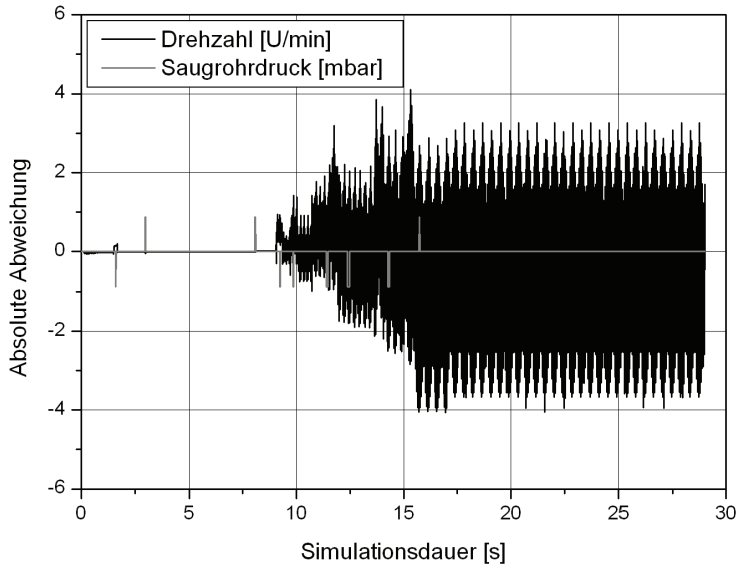

Abbildung 6.18: *Zeitliche Entwicklung und Stabilisierung des absoluten Abweichung zwischen dem Targetlink und dem Simulink Modell angegeben für die Größen Drehzahl uns Saugrohrdruck*

Nachkomma-Bereich. Zum Beispiel wird der die Interpolation des Einspritzkennfeldes sowohl bei Targetlink als auch bei Simulink mit den Eingangswerten 1.522 und 300 aufgerufen. Das Targetlink Ergebnis lautet 1.172,7263750 während Simulink einen Wert von 1.172,7263184 errechnet. Aufgrund des geschlossenen Regelkreises summiert sich diese geringe Differenz jedoch und führt zu den ermittelten Abweichungen. Nach einigen Simulationsschritten kommt es zu einem konstantem Fehler, der mit gleicher Amplitude um den Vergleichswert schwingt (vgl. Abbildung 6.18). In dem Diagramm ist die zeitliche Entwicklung der Abweichung zwischen den Simulationsergebnissen des Targetlink und des Simulink Modells anhand der Simulationsgrößen Drehzahl und Saugrohrdruck dargestllt.

Zur Bewertung der Code Effizienz wird ebenfalls analog zu Abschnitt 6.1.2 zunächst die statische Speichernutzung des generierten und kompilierten Modells verglichen (vgl. Tabelle 6.5[6]). Es ist zu erkennen, dass Targetlink den speichereffizienteren Code

[6]Die hier gezeigten Absolutwerte unterscheiden sich geringfügig von den in Tabelle 6.2 dargestellten Werten, da in den Untersuchungen ein jeweils anderer Entwicklungsstand der VeRa Bibliotheken

Speicherbereiche	Code Größe Realtime Workshop	Code Größe Targetlink
	belegte Speichergröße (in Byte)	belegte Speichergröße (in Byte)
ROM		
.text	161.748	141.232
.rodata	11.540	8.332
Σ	173.288	149.564
Δ		-23.724
RAM		
.sdata	176	136
.data	25.464	25.216
.sbss	380	376
.bss	18.704	17.748
.COMMON	1.976	1.120
Σ	46.700	44.596
Δ		-2.104

Tabelle 6.5: *Vergleich der Speichernutzung zwischen mit Targetlink und mit Realtime Workshop erzeugtem Code*

erzeugt. Im ROM Bereich ist der Code bei Targetlink um ca. 23 kB kleiner als bei Realtime Workshop, während er im RAM Bereich um ca. 2 kB kleiner ist. In der Ausführungszeit zeigen sich nur geringe Unterschiede zwischen dem mit Targetlink und dem mit Realtime Workshop erzeugtem Code. Die Ausführungszeit des Modells lag jeweils bei durchschnittlich 3,1 ms.

Neben dem statischen und dynamischen Vergleich des kompilierten Codes wurde der generierte Code mit Hilfe einer Software[7] auf die Einhaltung der MISRA Regeln [44] überprüft. Um generierten Code in sicherheitskritischen Anwendungen zu verwenden ist die Einhaltung dieser Regeln notwendig. Die Überprüfung ergab, dass Realtime Workshop bei der Codeerzeugung einige MISRA Regeln verletzt (vgl. Tabelle 6.6). So wurde zum Beispiel eine Funktion erzeugt, ohne einen entsprechenden Prototypen in einer Header-Datei zu generieren. Dies bedeutet eine Verletzung der MISRA Regel 8.1. In dem durch Targetlink erzeugten Code wurden hingegen keine Regelverletzungen festgestellt.

Neben der Umsetzung des Beispielmodells auf einen anderen Codegenerator ist auch die Integration des geänderten Codegenerators selbst Ziel dieser Evaluation.

verwendet wurde.

[7] PC Lint, http://www.gimpel.com/

MISRA Regelverletzungen durch von Realtime Workshop erzeugtem Code
`rtliSetLogXSignalPtrs(Motorsteuerung_M->rtwLogInfo, NULL);` Motorsteuerung.c 999 Error 64: Type mismatch (assignment) (ptrs to qualification,void/nonvoid) [MISRA Rule 5.4]
`Motorsteuerung_initialize( unsigned char)` defined without a prototype in scope [MISRA Rule 8.1]
`#undef rtmGetRTWSolverInfo` Violates MISRA 2004 Required Rule 19.6, use of '#undef' is discouraged [MISRA Rule 19.6]

Tabelle 6.6: *Beispiele für MISRA Regelverletzungen durch von Realtime Workshop erzeugtem Code*

Eine Anforderung an die für kleine Unternehmen angepasste Werkzeugkette ist die möglichst große Unabhängigkeit vom Modellierungswerkzeug im Sinne der Softwarequalitäten Wiederverwendbarkeit und Änderbarkeit. Wie oben dargestellt wurde konnte die Ankopplung des Modellierungs- und Codegenerierungswerkzeug an die hier vorgestellte Werkzeugkette durch Austausch weniger Module (`VeRa Toolbox` und `Model_Frame`) realisiert werden. Dies wird vor allem durch die Kapselung von Funktionen durch die VeRa Abstraktionsschicht erreicht. Für die Anpassung der Softwaremodule benötigte ein Entwickler weniger als zwei Wochen, da viele Codefragmente lediglich kopiert werden mussten. Geht man davon aus, das umfangreiche Tests und die komfortable Einbindung der Option Targetlink in die Anwednungsmenüs von VeRa Desk weitere vier bis sechs Wochen in Anspruch nehmen, kann der Aufwand für die Ankopplung eines neuen Modellierungswerkzeug als gering eingestuft werden. Jedoch muss ebenfalls berücksichtigt werden, dass der Aufwand für die Anbindung von anderen Modellierungswerkzeugen, die vollständig auf Matlab/Simulink als Simulationsumgebung verzichten, höher ausfällt als bei dem hier gezeigten Beispiel.

6.5 Weitere Arbeiten

Das Entwicklungswerkzeug wurde zumindest in Teilen auch bereits in mehreren Kundenprojekten eingesetzt. So wurde beispielsweise auch die Regelung eines Dieselmotors mit dem System entwickelt. Dabei konnte die Entwurfsmethodik gewinnbringend aus den hier vorgestellten Arbeiten übernommen werden. Auch die Basissoftware konnte wiederverwendet werden, so dass auch die Anforderungen an Wiederverwendbarkeit als erfüllt angesehen werden können.

Gegenüber der Steuerung eines Ottomotors (vgl. Abschnitt 6.1) musste die Bibliothek der Sensoren und Aktuatoren stark erweitert werden, da die realisierte Regelung deutlich komplexer ist und sich die elektrische Ausstattung eines Dieselmotors deutlich von der eines Otto-Motors unterscheidet. Dies bedeutete im Projekt jedoch einen hohen Initialaufwand, da der Bibliothek auch neue Matlab/SimulinkBlöcke hinzugefügt werden mussten, wenn nicht gleiche Typen mit geänderter Konfiguration verwendet werden können. Dieser Aufwand wurde im Projekt zunächst unterschätzt. Für das Hinzufügen eines neuen Sensor- oder Aktuatortyps müssen folgende Arbeiten durchgeführt werden:

- Erstellung eines Blocks unter Simulink
- Erstellung einer S-Funktion für den Aufruf des Simulationmoduls der Abstraktionsschicht.
- Implementierung eines Simulationsmoduls (DLL) als Teil der Abstraktionsschicht
- Implementierung eines Moduls für die Codegenerierung
- Ergänzung der Konfigurationsdatei

Da es sich hierbei jedoch immer um die gleichen Arbeiten handelt, bietet es sich an eine Werkzeugunterstützung zu implementieren. Diese könnte die entsprechenden Templates für die Erweiterung an den unterschiedlichen Stellen des Softwaredesigns anlegen, und über ein Skript die notwendigen Kompiliervorgänge starten. Die Implementierung eines solchen Werkzeugs ist aufgrund der für die gute Erweiterbarkeit entwickelten Architektur problemlos möglich. Des Weiteren würde ein solches Werkzeug auch Dritt-Unternehmen die Möglichkeit bieten eigene Bibliotheken zu erstellen.
Ein weiterer Aspekt der sich aus der Verwendung des Entwicklungswerkzeugs in Kundenprojekten ergab, war das Fehlen der Kalibriermöglichkeit von Parametern, die über die Konfiguration eingestellt werden. Dies könnte jedoch einfach durch Erweiterung des integrierten Code-Generators erreicht werden. Zusätzlich zu den Dateien `configuration.h` und `configuration.c` müsste dieser noch eine A2L Datei aus den Konfigurationsdaten erzeugen, die in die A2L Modelldatei integriert wird.

In einer weiteren studentischen Arbeit wurde am Lehrstuhl Informatik 11 der RWTH-Aachen die Austauschbarkeit von Sensoren nachgewiesen ohne das Modell zu verändern, sondern lediglich durch Veränderung der Konfiguration. Als Beispiel wurde hier ein Parkassistent mit einem Modell-Fahrzeug realisiert, bei dem das hier vorgestellte Steuergerät als Steuereinheit verwendet wurde. Eine detaillierte Beschreibung dieser Fallstudie erfolgt in [59].

6.6 Zuordnungstabelle

Zur Nachverfolgung der Beziehungen zwischen den durchgeführten Fallstudien, den Architekturelementen und den Softwareanforderungen sind diese in den folgenden Zuordnungstabellen (Traceability Matrix) zusammengefasst. Dabei werden in Tabelle 6.7 die Anforderungen hinsichtlich Brauchbarkeit und in Tabelle 6.8 die Anforderungen hinsichtlich Wartbarkeit zugeordnet. Zur Übersichtlichkeit werden für die durchgeführten Evaluierungen folgende Abkürzungen verwendet.

Abkürzungen	**Durchgeführte Evaluierung**
Motorsteuerung	Verwendung des Entwicklungswerkzeugs als Motorsteuerung (vgl. Abschnitt 6.1)
OSEK Einbindung	Evaluierung der Betriebssystembibliothek mit zeitbasierten und winkelbasierten Tasks (vgl. Abschnitt 6.2)
HIL	Untersuchung einer Dieselmotor Luftpfdregelung in einer Hardware-In-the-Loop (HIL) Testumgebung (vgl. Abschnitt 6.3)
Targetlink	Ankopplung von Targetlink als Modellierung- und Code-Generierungs-Werkzeug (vgl. Abschnitt 6.4)
Kundenprojekte	Einsatz des Entwicklungswerkzeugs in Kundenprojekten (vgl. Abschnitt 6.5)
Einparkassistent	Einsatz des Entwicklungswerkzeugs als Einparkassistent (vgl. Abschnitt 6.5)

Alle Anforderungen wurden durch die durchgeführten Evaluierungen untersucht. Dabei wurde gezeigt, das die Anforderungen zum überwiegenden Teil erfüllt werden. An den Stellen, wo Nachbesserungsbedarf besteht, wurde dies durch die Evaluierungen aufgedeckt.

#	Anforderung	Architekturelement	Evaluierung
B11	Modellbasierte Entwicklung wird unterstützt	VeRa Toolbox, VeRa Abstraction	Motorsteuerung, HIL, Kundenprojekte
Die modellbasierte Entwicklung wurde nachgewiesen, indem in den Evaluierungen unterschiedliche Steuerungssysteme modellbasiert erstellt wurden.			
B12	Bibliothek von Sensoren und Aktuatoren	VeRa Toolbox	Motorsteuerung, Kundenprojekte
Es wurden Sensor- und Aktuatorbibliotheken für die Verwendung in einem Motorsteuerungssystem realisiert und untersucht.			
B13	Offene Schnittstelle für User-Bibliotheken	VeRa Toolbox (Template)	–
In den Evaluierungen *Motorsteuerung*, *Kundenprojekte* und *Einparkassistent* wurden Bibliotheken entwickelt. Allerdings hat sich hier auch gezeigt, das die Bedienbarkeit für die Weitergabe an Dritt-Unternehmen verbessert werden muss (vgl. Abschnitt 6.5)			
B14	Verwendung des Werkzeugs als A-Muster	VeRa Hardware	Motorsteuerung
Das wesentliche Ziel des A-Musters ist die Darstellung der Funktionalität. Die wurde gezeigt, indem das System als Motorsteuerung eingesetzt werden konnte. In weiteren Projekten können Varianten als Seriensteuergerät abgeleitet werden.			
B21	Integration aller Softwarekomponenten in einer Bediensoftware	VeRa Desk	Motorsteuerung, Kundenprojekte
Das Benutzerprogramm VeRa Desk stellt für die Bedienung aller Softwareteile eine Oberfläche zur Verfügung. Die Brauchbarkeit dieses Moduls wurde im Fallbeispiel *Motorsteuerung* und in Kundenprojekten vielfach nachgewiesen.			
B31	Unterstützung eines qualifizierbaren Code-Generators	VeRa Abstraction	Targetlink
Die Ankopplung eines qualifizierbaren Code Generators und die Anpassung an unterschiedliche Simulations- und Modellierungswerkzeuge wurde in der Fallstudie *Targetlink* diskutiert.			

Tabelle 6.7: *Zuordnungstabelle von Evaluierungen zu Architekturelemente und Brauchbarkeitsanforderungen*

#	Anforderung	Architekturelement	Evaluierung
W11	Wiederverwendung von Architekturteilen	VeRa Base, VeRa Abstraction	alle
Die Komponenten der Basis Software und die Bibliothekselemente werden bei allen Projekten wiederverwendet.			
W12	Unabhängigkeit vom Modellierungswerkzeug	VeRa Abstraction	Targetlink
Durch Ankopplung eines zweiten Modellierungs- und Codegenerierungswerkzeugs wurde die Unabhängigkeit nachgewiesen.			
W13	Konfiguration und Softwarevarianten	VeRa Configuration file	Motorsteuerung, Kundenprojekte, Einparkassistent
In den unterschiedlichen Motorsteuerungsprojekten wurden unterschiedliche Software-Konfigurationen und Hardware-Varianten verwendet. Die Austauschbarkeit von Sensoren oder Aktuatoren ohne das Modell zu ändern wurde mit dem *Einparkassistenten* gezeigt.			
W21	Einheitlicher Treiberaufbau	VeRa Library	alle
Die Treiber nach dem HIS Standard wurden in allen Evaluationen getestet.			
W22	Weitgehende Verwendung von SW-Standards	VeRa Library, VeRa Base, ASAM MCD2	HIL, OSEK Tasks
Durch den HIL Versuch wurde die Brauchbarkeit des XCP Protokolls auch für HIL Simulationen nachgewiesen. Die Integration von OSEK Komponenten wurde mit der Untersuchung von *OSEK Tasks* evaluiert.			
W23	Coding Standard für die Programmierung	–	–
Diese Anforderung wird durch den Prozess gewährleistet. Bei der Implementierung der Architekturkomponenten wurde ein erweiterter MISRA Coding Standard beachtet.			
W31	Unterstützung von Simulation	VeRa Toolbox, VeRa Abstraction	Motorsteuerung, HIL
Durch die umfangreiche Unterstützung des modellbasierten Ansatzes wird Simulation als Hilfsmittel in der Entwicklung ermöglicht. Der zweckmäßige Einsatz von Simulation konnte mit der Simulation der Motorsteuerung (Model-In-the-Loop, MIL) und mit der Echtzeitankopplung bei der Luftpfadregelung (Hardware-In-the-Loop, HIL) gezeigt werden.			

Tabelle 6.8: *Zuordnungstabelle von Architekturelemente zu den Wartbarkeitsanforderungen*

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Das Ziel dieser Arbeit war es, einen Ansatz für ein KMU-orientiertes Software-Engineering aufzuzeigen, das Methoden und Werkzeuge zur Verfügung stellt, welche die Vorteile von kleinen und mittelständischen Unternehmen fördern. Die Vorteile sind die hohe Flexibilität, das kreative Umfeld und die starke Kundenorientierung. Gleichzeitig soll aber auch eine nachhaltige Qualität sichergestellt werden.
Der in dieser Arbeit vorgestellte Ansatz beruht insbesondere darauf, dass die Schnittstellen der einzelnen Teilprozesse zu den Prozessen der Großindustrie kompatibel sind. Die Prozesse selbst aber werden im Gegensatz zu den Prozessen großer Unternehmen vereinfacht, zusammengelegt und anders gewichtet. Dies ist möglich, da die heute in der Großindustrie verwendeten Entwicklungsprozesse neben dem Ziel der Qualitätssicherung auch das Ziel der Organisation der massiv verteilten Entwicklung verfolgen, was bei kleinen Unternehmen eine eher untergeordnete Rolle spielt. Der vorgestellte Prozessansatz sieht eine intensive Anforderungserfassung gemeinsam mit dem Kunden vor, mit dem Ziel ein möglichst detailliertes Problemverständnis zu erreichen. Anschließend wird ein Konzept erarbeitet, das konkrete Lösungen und Implementierungsvorgaben beschreibt. In einem gemeinsam mit dem Kunden durchgeführten Review wird dieses Konzept geprüft und angepasst. Die anschließende Softwareerstellung wird durch ein eigenes Werkzeug unterstützt. Für die Umsetzung von Reviews wurde ein eigenständiger Prozess definiert, der sich an klassischen Methoden orientiert, aber an die Unternehmensgröße angepasst wurde. Darüber hinaus wird vor allem die modellbasierte Entwicklung bei der Softwareerstellung als besonders geeignet für kleine Unternehmen herausgearbeitet. Die Vorteile sind zum einen die erweiterte Austauschbarkeit von Modellteilen mit Projektpartnern und zum anderen entspricht der modellbasierte Entwicklungsansatz, der prototypisch geprägten Entwicklung von Steuerungssystemen (A-Muster, B-Muster, usw.) in der Automobilindustrie. Eine konsistente Modellbasis in allen Phasen der Entwicklung ist dabei das Ziel. Das in dieser Arbeit vorgestellte Entwicklungswerkzeug unterstützt die modellbasierte Softwareerstellung. Zudem unterstützt es durch seine Softwarearchitektur insbesondere kleine und mittelständischen Unternehmen dabei, sich an die Prozess- und Softwareschnittstellen ihrer Auftraggeber anzupassen. Das Werkzeug erlaubt eine Abstraktion von der verwendeten Hardware und unterstützt die Entwickler dabei Softwaremodelle unabhängig von der sich im

Projektverlauf ändernden Hardware zu entwickeln. Dies wird vor allem durch die breite Unterstützung von Softwarestandards erreicht.

Bewertung des vorgestellten Prozessansatzes

Zur Bewertung der einzelnen Prozesse wurde die praktische Umsetzung der Prozesse in einem kleinen Unternehmen beschrieben. Diese Umsetzung wurde konzeptionell anhand des Automotive SPICE Bewertungsschemas für den ersten SPICE Reifegrad bewertet (vgl. Abschnitt 5.1). Dafür definiert Automotive SPICE Prozessergebnisse und Basispraktiken für die unterschiedlichen Teilprozesse, die in der Prozessimplementierung umgesetzt werden sollen. Durch Überprüfen und Kommentieren wurde gezeigt, dass die in dieser Arbeit vorgestellten Prozesse die einzelnen Prozessergebnisse erreichen und die Basispraktiken umgesetzt werden. Damit wird der erste SPICE Reifegrad erreicht. Dies ist die Voraussetzung dafür, durch Prozessoptimierung zukünftig auch höhere SPICE Reifegrade zu erreichen. Ergänzend zu dieser Arbeit müssen die vorgestellten Prozessansätze auch im praktischen Einsatz weiter evaluiert und an unterschiedliche Unternehmen angepasst werden. Im industriellen Einsatz muss die SPICE-Konformität darüber hinaus von unabhängiger Stelle bestätigt werden.
Die hier beschriebene Anforderungserfassung bezieht den Kunden durch gemeinsame Workshops und Prüfungen sehr stark mit ein (vgl. Abschnitt 5.2). Dieses Vorgehen ist auch unabhängig von der Unternehmensgröße vorteilhaft. Kleine und mittelständische Unternehmen können sich jedoch stärker an den Kunden anpassen und so wird ein enger Kontakt und ein umfassendes Problemverständnis erreicht. Dies ist wichtig, da manche Anforderungen, wie z.B. der Einsatz bestimmter Werkzeuge, durch die Unternehmensstruktur begründet werden. Die Prozessanforderungen nach Automotive SPICE werden somit voll erfüllt. Um die vielfältigen Abhängigkeiten jedoch noch besser darstellen und formalisieren zu können wurde zusätzlich untersucht, die Anforderungserfassung modellbasiert durchzuführen. Dafür wurde i* [68] als Modellierungssprache verwendet und in Zusammenarbeit mit dem Fraunhofer Institut für angewandte Informatik (FIT) ein Domänenmodell für den konkreten Anwendungsfall Motorsteuerung erstellt. Das Domänenmodell wird für gleichartige Projekte jeweils angepasst und in einer Datenbank abgelegt. Insbesondere für kleine und mittelständische Entwicklungsunternhemen ist jedoch die Flexibilität und Innovation entscheidend. Das bedeutet aber, dass hier ein sehr volatiles Domänenwissen vorherrscht, was zu einem Zielkonflikt führt, wenn ein Modell verwendet wird, das versucht das Domänenwissen vollständig abzubilden. Dennoch konnte insgesamt gezeigt werden, dass die Modellierung von Anforderungen ein erstrebenswertes Ziel darstellt, um die Anforderungserfassung und -verwaltung darin zu unterstützen eine gemeinsame Sichtweise aller Beteiligten auf das zu entwickelnde System zu erlangen. In dem hier beschriebenen Prozessansatz folgt auf die Anforderungserfassung die

Konzepterstellung (vgl. Abschnitt 5.3). In diesem Prozessschritt werden aus den Kundenanforderungen System- und Softwareanfordreungen abgeleitet. Zusätzlich werden auch die System- und die Softwarearchitektur erstellt. In Bezug auf Automotive SPICE werden somit vier Einzelprozesse (ENG.2-5) in einem Prozess integriert. Die Bewertung zeigt, dass die Prozessziele und Basispraktiken weitgehend erreicht werden. Für den ersten SPICE Reifegrad ist dies ausreichend. Dennoch wird durch die Zusammenlegung der Prozesse und durch das wenig detaillierte Softwarefeindesign deutlich weniger Prozess- und Dokumentationsaufwand erzeugt, als dies bei großen Unternehmen der Fall sein muss. Der Schwerpunkt wird bei dem beschriebenen Vorgehen vielmehr auf ein hohes Kunden- bzw. Auftraggebereinverständnis gelegt, indem ein ausführliches Konzept-Review durchgeführt wird. Dies gibt dem Kunden die Gelegenheit seine Ideen mit einzubringen und der Auftragnehmer gewinnt Sicherheit die Evaluierung durch den Kunden zu bestehen.
Für die Durchführung der Reviews wurde ein eigenständiger Prozess entworfen, der an das jeweilige Review-Objekt (wie z.B. das Konzept) angepasst werden kann (vgl. Abschnitt 5.5). Der Review Prozess wurde in Bezug auf die Architekturbewertung und die Codebewertung evaluiert. Hierbei konnte gezeigt werden, dass durch das beschriebene Vorgehen auch mit vergleichsweise geringem Aufwand Fehler gefunden werden und die Softwarequalität dadurch signifikant verbessert wird. Dies wurde erreicht, obwohl die beschriebenen Review-Teams nur aus wenigen Personen bestanden. An dieser Stelle sind weitergehende Untersuchungen von Interesse, die zeigen inwieweit die Besetzung Einfluss auf die Zahl gefundener Fehler hat. Der Aufwand für ein solche Untersuchung ist jedoch sehr hoch.
Das vorgegebene Ziel für die Softwareerstellung war es sie möglichst offen und an Kundensysteme anpassbar zu gestalten (vgl. Abschnitt 5.4). Dadurch soll der Vorteil der Flexibilität kleiner und mittelständischer Unternehmen gestärkt werden. Erreicht wird dies vor allem durch den Einsatz modellbasierter Softwareentwicklung, die durch den Einsatz des Entwicklungswerkzeugs unterstützt wird. Das beschrieben Vorgehen bindet die Simulation konsequent in die Softwareerstellung ein. Dabei wird das Ziel erreicht eine einheitliche Modellbasis von der Prototypenphase bis zum Seriensteuergerät aufzubauen und zu pflegen. In der Bewertung nach Automotive SPICE werden die Prozessziele weitestgehend erreicht. In der Weiterentwicklung müssen Softwaretestmethoden noch stärker in den den Softwareerstellungsprozess integriert werden.

Bewertung des Entwicklungswerkzeugs

Für die Bewertung wurde das Entwicklungswerkzeug in mehreren unterschiedlichen praktischen Evaluierungen untersucht. Dabei war es das Ziel zu zeigen, dass die in Kapitel 4 genannten Anforderungen durch die Implementierung erfüllt werden.
In einem ersten Projekt wurde mit Hilfe des Werkzeugs ein Motorsteuerungssys-

tem für einen Otto-Motor implementiert und getestet (vgl. Abschnitt 6.1). Dabei konnte gezeigt werden, dass die Ergebnisse in der Simulation unter Verwendung der neu implementierten Softwareteile gleich sind mit denen ohne Verwendung dieser Komponenten. Aus dem mit Hilfe der Simulation entwickelten Modell wurde mit dem Werkzeugs automatisch C-Code generiert und für eine seriennahe Hardware kompiliert. Ein realer Verbrennungsmotor wurde mit dem Steuerungssystem auf dem Prüfstand getestet, dabei konnten die Simulationsergebnisse weitgehend bestätigt werden. Es wurde somit gezeigt, dass das hier vorgestellte Entwicklungswerkzeug für die Entwicklung von Motorsteuergeräten geeignet ist. Der dargestellte Entwicklungsstand kann als A-Muster interpretiert werden, bei dem die geforderte Funktionalität vollumfänglich gegeben ist.
In einem weiteren Projekt wurde gezeigt das der wichtige Betriebssystemstandard OSEK durch das Entwicklungswerkzeug unterstützt wird (vgl. Abschnitt 6.2). Die besondere Schwierigkeit liegt in der Berücksichtigung der durch das Betriebssystem hervorgerufenen Effekte auch in der Simulation. Anhand einiger Beispielprogramme wurde diese Funktionalität nachgewiesen. In folgenden Arbeiten muss dies jedoch auch noch im realen Einsatz, wie z.B. einem Motorsteuerungssystem gezeigt werden.
Als Ziel für den Softwareerstellungsprozess wurde der Aufbau einer durchgängigen Modellbasis genannt. Durch ein weiteres Evaluierungsprojekt konnte die Verwendung des Entwicklungswerkzeugs in einer Hardware-In-the-Loop (HIL) Umgebung gezeigt werden (vgl. Abschnitt 6.3). Dabei wurde das Steuerungssystem über den CAN-Bus und mit Hilfe des XCP Protokolls mit einem Rechner verbunden, der das Modell der zu regelnden Strecke in Echtzeit ausführte. Es konnte gezeigt werden, dass das Vorgehen sehr gut geeignet ist, um Simulationsergebnisse mit den Berechnungsergebnissen auf der Zielhardware zu vergleichen. Das Verfahren eignet sich ebenfalls, um Teile der Berechnungen für Entwicklungszwecke auf ein zweites System auszulagern (Bypass, vgl. [54]). Für HIL Versuche, die auch hardwareseitige Sensoreingänge und Aktuatorendstufen berücksichtigt ist dieses Verfahren jedoch nicht geeignet, da lediglich Softwareschnittstellen für die Kommunikation verwendet werden.
Um auch die Anpassbarkeit des Entwicklungswerkzeugs an unterschiedliche externe Software nachzuweisen wurde das System an das System Targetlink der Fa. dSPACE als Alternative zu Realtime Workshop der Fa. Mathworks angekoppelt (vgl. Abschnitt 6.4). Die Anbindung konnte aufgrund der Softwarearchitektur des Entwicklungswerkzeugs innerhalb weniger Wochen durch einen Entwickler durchgeführt werden. In durchgeführten Beispielsimulationen konnten zum Teil große Abweichungen in der Berechnung der beiden Werkzeuge festgestellt werden. Dies zeigt, dass es besonders wichtig ist während der Entwicklung nicht das Werkzeug zu wechseln. Der Wert von Simulationsergebnissen kann dadurch verlorengehen und die durchgängige Modellbasis ist nicht mehr gegeben.
Insgesamt wurde in dieser Arbeit der Ansatz für ein modellbasiertes Vorgehen in der Entwicklung von automobilen Steuerungssystemen in kleinen und mittelständischen

Unternehmen vorgestellt, realisiert und evaluiert. Der Ansatz setzt sich zusammen aus einem an die Unternehmensstruktur angepassten Entwicklungsprozess und einem Werkzeug, das die modellbasierte Softwareentwicklung insbesondere in kleinen und mittelständischen Entwicklungsunternehmen unterstützt.

7.2 Ausblick

Die Anforderungserfassung wurde in dieser Arbeit als besonders effektiv dargestellt, um mit dem Kunden ein gemeinsames Problemverständnis aufzubauen. Zukünftige Arbeiten können dies noch weiter ausbauen, indem modellbasierte Methoden zur Anforderungserfassung und -dokumentation eingesetzt werden. Der hier vorgestellte i* Formalismus kann dafür ein Ausgangspunkt sein. Dabei erscheint es auch möglich aus Anforderungsmodellen erste funktionale Modelle automatisch zu generieren. Dies würde die durchgängige Modellbasis noch weiter vervollständigen.
Darüber hinaus ist es wünschenswert, wenn der in der Arbeit beschriebene Prozessansatz auch in weiteren Unternehmen Anwendung findet, um eine größere Erfahrungsbasis zu erhalten. Es ist sicher nicht möglich einen Prozess zu definieren, der zu allen kleinen und mittelständischen Entwicklungsunternehmen passt, aber die hier beschriebenen Prinzipien können auch auf andere Unternehmen und deren Prozese übertragen werden. Die Prozesse, die dem hier beschriebenen Ansatz entsprechen, sollten zudem auch von unabhängiger Stelle nach dem Automotive SPICE Bewertungsschema assessiert werden.
Die in dieser Arbeit dargestellten Prozesse und auch das Entwicklungswerkzeug beziehen sich im Wesentlichen auf den Entwurfs- und Implementierungsteil eines Softwareentwicklungsprozesses und umfassen damit grob den absteigenden Ast des V-Modells. Für eine erfolgreiche Softwareentwicklung muss jedoch auch der aufsteigende Ast des V-Modells, d.h. die Testphase intensiver in der Adaption für kleine und mittelständische Unternehmen untersucht werden. Dabei kann auf Ergebnisse dieser Arbeit aufgebaut werden, wenn modellbasierte Testmethoden wie z.B. Hardware-In-the-Loop (HIL) intensiver eingesetzt werden. Weiterhin sollte auch untersucht werden, ob die intensive Einbindung von werkzeugbasierten Testmethoden, wie z.B. Model-Checking, insbesondere für kleine und mittelständische Entwicklungsunternehmen einen Vorteil bedeuten.
Bei vielen Steuergeräteentwicklungen wird mit AUTOSAR derzeit eine zusätzliche Ausführungsschicht eingeführt. Die hiermit verfolgten Ziele sind eine stärkere Unabhängigkeit von der Hardware sowie eine weitgehende Entkopplung einzelner Softwarefunktionen. Der in dieser Arbeit beschriebene Entwicklungsprozessansatz ist für AUTOSAR Projekte ebenfalls gut geeignet. Die Verwendung einer derartigen Ausführungsschicht bietet kleinen und mittelständischen Unternehmen sogar gute

Möglichkeiten einzelene Softwarefunktionen zu entwickeln, da die Gesamtsoftware besser modularisiert werden kann. Das in dieser Arbeit vorgestellte Werkzeug bietet die Voraussetzungen, auch für den Einsatz in AUTOSAR Projekten weiterentwickelt zu werden. Besondere Aufmerksamkeit muss dabei auf eine korrekte Simulation der AUTOSAR Umgebung gelegt werden.
Mit der weiteren Verbreitung des beschriebenen Entwicklungswerkzeugs und durch die Anwendung der beschriebenen Prozesse wird der vorgestellte Ansatz weiterentwickelt werden. Da als wesentliche Zielgruppe kleine und mittelständische Unternehmen angesprochen werden, wird diese Weiterentwicklung auch mit hohem Tempo voranschreiten, da diese Unternehmen die Innovationsträger der Wirtschaft sind.

Literaturverzeichnis

[1] Abel, D. ; Bollig, A.: *Rapid Control Prototyping*. Springer, 2006

[2] Allweyer, T.: *BPMN 2.0, Business Process Model and Notation*. Books on Demand GmbH, 2009

[3] Angermann, A. ; Beuschel, M. ; Rau, M. ; Wohlfarth, U. ; Verlag, Oldenbourg (Hrsg.): *Matlab - Simulink - Stateflow*. 2004

[4] ASAM: ASAM MCD-2 MC – ECU Measurement and Calibration Data Exchange Format / ASAM e.V., www.asam.net. – Standard

[5] ASAM: XCP – The Universal Measurement and Calibration Protocol, Ver. 1.0 / ASAM e.V., www.asam.net. 2003. – Standard

[6] Bass, L. ; Clements, P. ; Kazman, R.: *Software Architecture in Practice*. 2. Addison Wesley, 2003 (SEI Series in Software Engineering)

[7] Bass, L. ; Clements, P. ; Kazman, R. ; Northrop, L. ; Zaremski, A.: Recommended Best Industrial Practice For Software Architecture Evaluation / Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh. 1997. – Forschungsbericht

[8] Beine, M.: Modellbasierte Softwareentwicklung nach ISO 26262. In: *Elektronik automotive* 8 (2009), S. 41–43

[9] Boehm, B.W.: Guidelines for verifying and validating software requirements and design specifications. In: *EURO IFIP 79*. North Holland, 1979, S. 711–719

[10] Boehm, B.W. ; Brown, J.R. ; Lipow, M.: Quantitative Evaluation of Software Quality. In: *Proceedings of 2nd ICSE, San Francisco, IEEE Computer Society, 592-605*, 1976

[11] Bärwald, A. ; Beine, M.: Sichere Codegenerierung. In: *Hanser Automotive* (02/2010), S. 30–33

[12] Cockburn, A.: *Crystal Clear: Agile Software-Entwicklung für kleine Teams*. mitp, 2005

[13] Colin, J. N. ; Laplante, P. A.: Requirements Engineering: The State of the Practice. In: *IEEE Software* 20 (11/2003), S. 40–45

[14] Conrad, M. ; Fey, I. ; Grochtmann, M. ; Klein, T.: Modellbasierte Entwicklung eingebetteter Fahrzeug-Software bei DaimlerChrysler. In: *Informatik - Forschung und Entwicklung* 20, Sonderteil "Modellierung"(2005), S. 3–10

[15] Ebert, C.: *Systematisches Requirements Engineering.* 3. Auflage. dpunkt, 2010

[16] Eicker, S. ; Hegmanns, C. ; Malich, S.: Auswahl von Bewertungsmethoden für Softwarearchitekturen / Institut für Informatik und Wirtschaftsinformatik (ICB), Universität Duisburg-Essen. 2007. – Forschungsbericht

[17] Fagan, M.: Design and code inspections to reduce errors in program development. In: *IBM Systems Journal*, 1976, S. 182–211

[18] Gasomson, T.: *Improvement of a software architecture for a Rapid-Control-Prototyping-System for parallel execution of different model parts*, Lehrstuhl Informatik 11, RWTH Aachen, Masterthesis, 2009

[19] Guzzella, L. ; Onder, C.H.: *Introduction to Modeling and Control of Internal Combustion Engine Systems.* Springer, 2004

[20] Hanser, E.: *Agile Prozesse: Von XP über Scrum bis MAP.* Springer, 2010

[21] Hesseler, F. J. ; Drews, P. ; Abel, D. ; Reke, M. ; Düsterhöft, M.: Hardware-in-the-Loop-Umgebung für ein seriennahes RCP-System für den Test von modellprädiktiven Reglern. In: *Haus der Technik - 8. Tagung Hardware-in-the-Loop-Simulation*, 2008

[22] HIS: HIS Process-Scope automotiveSPICE V.01 / HIS - Working Group Assessment. 2005. – Standard

[23] HIS: HIS WG - Assessments V.31 / HIS - Working Group Assessment. 2008. – Standard

[24] Höhn, H. ; Sechser, B. ; Dussa-Zieger, K. ; Messnarz, R. ; Hindel, B.: *Software Engineering nach Automotive SPICE.* dpunkt.verlag, 2009

[25] Hörmann, K. ; Dittmann, L. ; Hindel, B. ; Müller, M.: *SPICE in der Praxis.* dpunkt.verlag, 2006

[26] IEC: IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems / International Electrotechnical Commission. 2010. – Standard

[27] IEEE: IEEE Std 610.12 1990: IEEE Standard Glossary of Software Engineering Terminology / Institute of Electrical and Electronics Engineers. 1990. – Standard

[28] Ionita, M. ; Hammer, D. ; Obbink, H.: Scenario-Based Software Architecture Evaluation Methods: An Overview / Department Software Architectures, Philips Research,Department of Mathematics and Computing Science, Technical University Eindhoven. – Forschungsbericht

[29] ISO: ISO 9001:2008: Quality management systems - Requirements / International Organization for Standardization. 2008. – Standard

[30] ISO: ISO/FDIS 26262: Road vehicles - Functional safety, Final Draft International Standard / International Organization for Standardization. 2011. – Standard

[31] ISO/IATF: ISO/TS 16949:2009: Quality Management Systems - Particular requirements for the application of ISO 9001:2008 for automotive production and relevant service part organizations / International Organization for Standardization, International Automotive Task Force. 2009. – Standard

[32] ISO/IEC: ISO/IEC 15504: Information technology – Software process assessment / International Organization for Standardization. 2004. – Standard

[33] ISO/IEC: ISO/IEC 19501:2005, Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2 / International Organization for Standardization. 2005. – Standard

[34] ISO/IEC: ISO/IEC 42010: Recommended Practice for Architectural Description of Software-Intensive Systems / International Organization for Standardization. 2007. – Standard

[35] Jacobsen, I. ; Booch, G. ; Rumbaugh, J.: *The Unified Software Development Process*. Addison Wesley, 1999

[36] Jarke, M. ; Gallersdörfer, R. ; Jeusfeld, M. A. ; Staudt, M.: ConceptBase - A Deductive Object Base for Meta Data Management. In: *Journal of Intelligent Information Systems* 4 (1995), Nr. 2, S. 167–192

[37] Jung, S.: Ausgewählte Ergebnisse für kleine und mittlere Unternehmen in Deutschland 2007 / Statistisches Bundesamt. 01/2010. – Forschungsbericht

[38] Kladroba, A. ; Grenzmann, C. ; Kreuels, B.: FuE-Datenreport 2010 / Stifterverband Wissenschaftsstatistik. 2010. – Forschungsbericht

[39] Kämmer, A.: *Erstellung von Echtzeitmotormodellen aus den Konstruktionsdaten von Verbrennungsmotoren*, Technische Universität Dresden, Diss., 2003

[40] Kruchten, P.: *The Rational Unified Process: An Introduction.* Addison Wesley, 1998

[41] Lichter, H. ; Ludewig, J.: *Software Engineering, Grundlagen, Menschen, Prozesse, Techniken.* 1. Auflage. dpunkt.verlag, 2007

[42] Liu, L. ; Yu, E.: *Organization Modeling Environment OME.* http://www.cs.toronto.edu/km/ome, 2008

[43] Meier, E.: V-Modelle in Automotive-Projekten. In: *AUTOMOBIL-ELEKTRONIK* (Februar 2008), S. 36–37

[44] MISRA: *MISRA-C Guidelines for the use of the C language in critical systems.* http://www.misra.org.uk/, 2004

[45] Müller, M. ; Hörmann, K. ; Dittmann, L. ; Zimmer, J.: *Automotive SPICE in der Praxis.* dpunkt.verlag, 2007

[46] NASA: Software Formal Inspections Guidebook / NASA Office of safety and mission assurance. 1993. – Richtlinie

[47] Nissen, H. W. ; Schmitz, D. ; Jarke, M. ; Rose, T. ; Drews, P. ; Hesseler, F. J. ; Reke, M.: Evolution in Domain Model-Based Requirements Engineering for Control Systems Development. In: *17th Int. Requirements Engineering Conf.* Atlanta, USA : IEEE, 2009

[48] Nissen, H.W. ; Schmitz, D. ; Jarke, M. ; Rose, T.: How to Keep Domain Requirements Models Reasonably Sized. In: *2nd Workshop on Managing Requirements Knowledge (MaRK@RE).* Atlanta, USA, September 2009

[49] P. Kuvaja, A. B.: BOOTSTRAP - a European assessment methodology. In: *Software Quality Journal* 3 (1994), S. 117–127

[50] Patzer, Andreas: Bypassing basierend auf Standards. In: *automotive* Ausg. 11 (12/2005), S. 54–56

[51] Polzer, A. ; Palczynski, J.: BMBF-Projekt ZAMOMO: Verzahnung von modellbasierter Softwareentwicklung und modellbasiertem Reglerentwurf. 2009. – Forschungsbericht

[52] Reinhard Höhn, Stephan H.: *Das V-Modell XT.* Springer, 2008

[53] Rolfsmeier, Andre: XCP in der ECU-Entwicklung. In: *automotive* Ausg. 5 (6/2008), S. 56–59

[54] Schäuffele, J. ; Zurawka, T.: *Automotive Software Engineering*. Vieweg, 2. Auflage, 2004

[55] Schmitz, D. ; Drews, P. ; Hesseler, F. ; Jarke, M. ; Kowalewski, S. ; Palczynski, J. ; Polzer, A. ; Reke, M. ; Rose, T.: Modellbasierte Anforderungserfassung für softwarebasierte Regelungen. In: *Software Engineering*, 2008, S. 257–271

[56] Schmitz, D. ; Nissen, H. W. ; Jarke, M. ; Rose, T. ; Drews, P. ; Hesseler, F. J. ; Reke, M.: Requirements Engineering for Control Systems Development in Small and Medium-Sized Enterprises. In: *16th International Requirements Engineering Conference*. Barcelona, Spain, September 2008, S. 229–234

[57] Schwaber, K.: *Agile Project Management with Scrum*. Microsoft Press, 2004

[58] SEI: CMMI© for Development, Version 1.2 / Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh. 2006. – Standard

[59] Steffens, C.: *Systementwurf eines hardware-unabhängigen Parkassistenten mit Hilfe einer Rapid Control Prototyping Plattform*. 2008. – Diplomarbeit, RWTH Aachen

[60] Sutcliffe, A. G. ; Maiden, N. A. M.: Use of Domain Knowledge for Requirements Validation. In: *Proc. of IFIP WG8.1 Conference on Information System Development Process, Como Italy, 1-3 September*, Elsevier, 1993

[61] Theissen, S.: *Weiterentwicklung einer Softwarearchitektur für ein Rapid-Control-Prototyping-System für den Einsatz als Motorsteuerung*, Lehrstuhl Informatik 11, RWTH Aachen, Diplomarbeit, 2008

[62] *Kapitel* Model-Based Development of Automotive Embedded Systems. In: Törngren, M. ; Chen, D. ; Malvius, D. ; Axelsson, Jakob: *Automotive Embedded Systems Handbook*. CRC Press, 2009

[63] VDA: Automotive SPICE Prozessassessment, 1. Auflage / Verband der Automobilindustrie (VDA). 2007. – Standard

[64] VDI: VDI 2519: Vorgehensweise bei der Erstellung von Lasten-/Pflichtenheften / Verein deutscher Ingenieure. 2001. – Forschungsbericht

[65] VDI: VDI 2206: Entwicklungsmethodik für mechatronische Systeme / Verein deutscher Ingenieure. 2004. – Standard

[66] Vibe, I.: *Brennverlauf und Kreisprozeß von Verbrennungsmotoren*. Verlag Technik, 1970

[67] Wissenschaftsstatistik, Stifterverband: facts Forschung & Entwicklung / Stifterverband Wissenschaftsstatistik. 01/2011. – Forschungsbericht

[68] Yu, E.: *Modelling Strategic Relationships for Process Reengineering*, University of Toronto, Diss., 1995

[69] Zimmermann, W. ; Schmidgall, R.: *Bussysteme in der Fahrzeugtechnik*. Bd. 2. Auflage. Vieweg, 2007

[70] Zuser, W. ; Greching, T. ; Köhle, M.: *Software Engineering mit UML und dem Unified Process*. Pearson Studium, 2004

Tabellenverzeichnis

4.1 *Übersicht der Anforderungen an das Entwicklungswerkzeug eingeordnet in den Qualitätenbaum* 64
4.2 *VeRa Varianten* 71
4.3 *Zuordnungstabelle von Architekturelemente zu den Brauchbarkeitsanforderungen* 87
4.4 *Zuordnungstabelle von Architekturelemente zu den Wartbarkeitsanforderungen* 88

5.1 *SPICE Bewertungsskala nach ISO/IEC 15504-2 [32]* 91
5.2 *Bewertung der Anforderungserhebung nach ENG.1* 96
5.3 *Bewertung der System- bzw. Softwareanforderungsanalyse nach ENG.2 bzw. ENG.4* 100
5.4 *Bewertung der System- und Softwarearchitekturentwicklung nach ENG.3 und ENG.5* 101
5.5 *Bewertung des Softwareerstellungsprozesses nach ENG.6* 105
5.6 *Ergebnis der Architekturbewertung [61]* 111
5.7 *Ergebnis des Code Reviews* 113
5.8 *Bewertung der gemeinsamen Reviews nach SUP.4* 114

6.1 *Technische Daten des Versuchsmotors (Quelle: Weber Motor AG)* . 118
6.2 *Vergleich der Speichernutzung* 124
6.3 *Vergleich von Code für die Simulation an dem Beispielblock Fuel* . . 138
6.4 *Vergleich von Code für die Codegenerierung an dem Beispielblock Fuel* 139
6.5 *Vergleich der Speichernutzung zwischen mit Targetlink und mit Realtime Workshop erzeugtem Code* 143
6.6 *Beispiele für MISRA Regelverletzungen durch von Realtime Workshop erzeugtem Code* 144
6.7 *Zuordnungstabelle von Evaluierungen zu Architekturelemente und Brauchbarkeitsanforderungen* 147
6.8 *Zuordnungstabelle von Architekturelemente zu den Wartbarkeitsanforderungen* 148

Abbildungsverzeichnis

1.1 *Projekte in kleinen und mittlere Entwicklungsunternehmen im Spannungsfeld zwischen Kosten, Qualität und Zeit* 3
1.2 *Einbindung von kleinen und mittelständischen Unternehmen in die Entwicklung* 5

2.1 *ZAMOMO-Entwicklungsprozess für softwareintensive Regelungssysteme [51]* 12
2.2 *Modellbasierte Entwicklung aus [54]* 19
2.3 *Qualitätenbaum nach [41]* 20
2.4 *Architecture Business Cycle nach [6]* 23
2.5 *Allgemeine Vorgehensweise beim Prototyping [41]* 32
2.6 *Das V-Modell nach Boehm [9]* 35
2.7 *V-Modell 97: Tätigkeiten zur Systemerstellung [52]* 36
2.8 *Ablauf des Scrum-Prozesses in Anlehnung an [20]* 41
2.9 *Prozesse in Automotive SPICE [45]* 44
2.10 *Die sechs Reifegradstufen in Automotive SPICE* 45
2.11 *Einige Basis-Notationselemente der BPMN* 48

3.1 *Ansatz eines Softwareentwicklungsprozesses für kleine und mittelständische Unternehmen* 51
3.2 *Prozess zur Anforderungserfassung für KMUs* 52
3.3 *Prozess zur Erstellung eines integrierten Konzepts. Dies umfasst für System, Hardware und Software, die Anforderungen sowie die Architektur- und Designbeschreibung* 54
3.4 *Prozess zur Softwareerstellung, wie er für A-, B- und C-Muster durchgeführt wird.* 55
3.5 *Review Prozess, der auf unterschiedliche Review-Objekte angewendet werden kann* 56
3.6 *Prinzipielle Darstellung des Review-Rahmenwerkes [61]* 58

4.1 *Grundlegendes Use Case Diagramm* 65
4.2 *Detailliertes Use Case Diagramm für die Projektarbeit, in der Steuerungen und Regelungen für reale Strecken realisiert werden* 66

4.3 *Entwicklungssystem VeRa bestehend aus Hardware und Software Toolchain* . . . 67
4.4 *VeRa Hardwareaufbau* . . . 68
4.5 *Die unterschiedlichen Softwarekomponenten des VeRa Gesamtsystems sind auf mehrere Plattformen verteilt. Hier im Deployment Diagramm dargestellt* . . . 69
4.6 *Das VeRa System im Komponentendiagramm* . . . 70
4.7 *Klassendiagramm der VeRa Abstraktionsschicht* . . . 73
4.8 *Die Modellierungselemente Sensor und Aktuator enthalten für den Anwendungsfall der Simulation ein simuliertes Verhalten. Für den Fall der Codegenerierung werden über Hardware-Treiber die realen Sensorwerte eingelesen und die Aktuatorendstufen angesteuert.* . . . 76
4.9 *Objektdiagramm für den Anwendungsfall Simulation* . . . 77
4.10 *Sensoraufruf im Anwendungsfall Simulation dargstellt im Sequenzdiagramm* . . . 78
4.11 *Objektdiagramm für den Anwendungsfall Code Generierung* . . . 80
4.12 *Sensoraufruf im Anwendungsfall Code Generierung dargstellt im Sequenzdiagramm* . . . 81
4.13 *Beispiel eines Taskmodells, Task 1 ist zeitbasiert mit kleiner Periodendauer (T1), Task 2 ist zeitbasiert mit großer Periodendauer (T2), Task 3 ist winkelbasiert mit einer Periodendauer (T3)* . . . 83
4.14 *Das VeRa Ausführungssystem im Klassendiagramm* . . . 83
4.15 *Startsequenz des VeRa Ausführungssystems* . . . 85

5.1 *Beurteilung von PA1.1 nach [45]* . . . 90
5.2 *Generisches Domänenmodell für Motorsteuerungssysteme [56]* . . . 94

6.1 *Ansatz zur Zerlegung des Motors in Komponenten (nach [39], Hintergrundbild: Bosch)* . . . 119
6.2 *Modulare Modellstruktur [61]* . . . 119
6.3 *Das Modell mit Verwendung der Abstraktionsschicht [61]* . . . 120
6.4 *Der Block Manifold Pressure in einer geeigneten Testumgebung [61]* 121
6.5 *Testergebnisse des Block Manifold Pressure [61]* . . . 122
6.6 *Vergleich der gemessenen mit den simulierten Saugrohrdrücken bei einem effektiven Mitteldruck von 4 bar [61]* . . . 126
6.7 *Vergleich der gemessenen mit den simulierten Luftmassenströmen bei einem effektiven Mitteldruck von 4 bar [61]* . . . 127
6.8 *Versuchsmodell bestehend aus drei Tasks* . . . 129
6.9 *Berechnung im Subsystem aller Tasks* . . . 129

6.10 *Simulationsergebnis des winkelbasierten Tasks A bei einer Drehzahl von 2000 U/min* 130
6.11 *Testergebnis des winkelbasierten Tasks A mit generierten Codes auf der Zielplattform bei einer Drehzahl von 2000 U/min* 131
6.12 *HIL-System mit XCP-Verbindung [21]* 133
6.13 *Schematischer Versuchsaufbau [21]* 134
6.14 *Regelergebnisse im HIL-Test mit acht Abtastschritten bei N = 2000 1/min und $\dot{m}_B = 0.002$ kg/s [21]* 135
6.15 *Klassendiagramm der VeRa Abstraktionsschicht. Die VeRa Toolbox, die für den Einsatz des Code-Generators Targetlink geändert wurde, ist schwarz umrandet.* 137
6.16 *Klassendiagramm des VeRa Ausführungssystems. Das Aufrufmodul sowie der mit Targetlink generierte Code sind schwarz umrandet.* . . 140
6.17 *Vergleich der Targetlink und der Simulink Simulationsergebnisse bei einem effektiven Mitteldruck von 4bar.* 141
6.18 *Zeitliche Entwicklung und Stabilisierung des absoluten Abweichung zwischen dem Targetlink und dem Simulink Modell angegeben für die Größen Drehzahl uns Saugrohrdruck* 142

Curriculum Vitae

Name	Michael Reke
Geburtsdatum	05.01.1975
Geburtsort	Eslohe (Sauerland)

Seit 2009	Technischer Leiter bei VEMAC GmbH & Co. KG, Aachen
2005 – 2009	Abteilungsleiter Software- und Hardwareentwicklung bei VEMAC GmbH & Co. KG, Aachen
2001 – 2005	Projektingenieur bei VEMAC GmbH & Co. KG, Aachen

2006 – 2012	Promotionsstudium der Informatik an der RWTH Aachen
1995 – 2001	Studium der Elektrotechnik an der RWTH Aachen
1994 – 1995	Zivildienst
1985 – 1994	Gymnasium der Benediktiner, Meschede
1981 – 1985	Grundschule Eslohe

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:

`http://aib.informatik.rwth-aachen.de/`

To obtain copies please consult the above URL or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: `biblio@informatik.rwth-aachen.de`

2010-01 *	Fachgruppe Informatik: Jahresbericht 2010
2010-02	Daniel Neider, Christof Löding: Learning Visibly One-Counter Automata in Polynomial Time
2010-03	Holger Krahn: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering
2010-04	René Wörzberger: Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme
2010-05	Daniel Retkowitz: Softwareunterstützung für adaptive eHome-Systeme
2010-06	Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Computing maximum reachability probabilities in Markovian timed automata
2010-07	George B. Mertzios: A New Intersection Model for Multitolerance Graphs, Hierarchy, and Efficient Algorithms
2010-08	Carsten Otto, Marc Brockschmidt, Christian von Essen, Jürgen Giesl: Automated Termination Analysis of Java Bytecode by Term Rewriting
2010-09	George B. Mertzios, Shmuel Zaks: The Structure of the Intersection of Tolerance and Cocomparability Graphs
2010-10	Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, René Thiemann: Automated Termination Analysis for Logic Programs with Cut
2010-11	Martin Zimmermann: Parametric LTL Games
2010-12	Thomas Ströder, Peter Schneider-Kamp, Jürgen Giesl: Dependency Triples for Improving Termination Analysis of Logic Programs with Cut
2010-13	Ashraf Armoush: Design Patterns for Safety-Critical Embedded Systems
2010-14	Michael Codish, Carsten Fuhs, Jürgen Giesl, Peter Schneider-Kamp: Lazy Abstraction for Size-Change Termination
2010-15	Marc Brockschmidt, Carsten Otto, Christian von Essen, Jürgen Giesl: Termination Graphs for Java Bytecode
2010-16	Christian Berger: Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles
2010-17	Hans Grönniger: Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten
2010-18	Ibrahim Armaç: Personalisierte eHomes: Mobilität, Privatsphäre und Sicherheit
2010-19	Felix Reidl: Experimental Evaluation of an Independent Set Algorithm

2010-20 Wladimir Fridman, Christof Löding, Martin Zimmermann: Degrees of Lookahead in Context-free Infinite Games

2011-01 * Fachgruppe Informatik: Jahresbericht 2011

2011-02 Marc Brockschmidt, Carsten Otto, Jürgen Giesl: Modular Termination Proofs of Recursive Java Bytecode Programs by Term Rewriting

2011-03 Lars Noschinski, Fabian Emmes, Jürgen Giesl: A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems

2011-04 Christina Jansen, Jonathan Heinen, Joost-Pieter Katoen, Thomas Noll: A Local Greibach Normal Form for Hyperedge Replacement Grammars

2011-06 Johannes Lotz, Klaus Leppkes, and Uwe Naumann: dco/c++ - Derivative Code by Overloading in C++

2011-07 Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe: An Operational Semantics for Activity Diagrams using SMV

2011-08 Thomas Ströder, Fabian Emmes, Peter Schneider-Kamp, Jürgen Giesl, Carsten Fuhs: A Linear Operational Semantics for Termination and Complexity Analysis of ISO Prolog

2011-09 Markus Beckers, Johannes Lotz, Viktor Mosenkis, Uwe Naumann (Editors): Fifth SIAM Workshop on Combinatorial Scientific Computing

2011-10 Markus Beckers, Viktor Mosenkis, Michael Maier, Uwe Naumann: Adjoint Subgradient Calculation for McCormick Relaxations

2011-11 Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, Bernd Becker: Hierarchical Counterexamples for Discrete-Time Markov Chains

2011-12 Ingo Felscher, Wolfgang Thomas: On Compositional Failure Detection in Structured Transition Systems

2011-13 Michael Förster, Uwe Naumann, Jean Utke: Toward Adjoint OpenMP

2011-14 Daniel Neider, Roman Rabinovich, Martin Zimmermann: Solving Muller Games via Safety Games

2011-16 Niloofar Safiran, Uwe Naumann: Toward Adjoint OpenFOAM

2011-17 Carsten Fuhs: SAT Encodings: From Constraint-Based Termination Analysis to Circuit Synthesis

2011-18 Kamal Barakat: Introducing Timers to pi-Calculus

2011-19 Marc Brockschmidt, Thomas Ströder, Carsten Otto, Jürgen Giesl: Automated Detection of Non-Termination and NullPointerExceptions for Java Bytecode

2011-24 Callum Corbett, Uwe Naumann, Alexander Mitsos: Demonstration of a Branch-and-Bound Algorithm for Global Optimization using McCormick Relaxations

2011-25 Callum Corbett, Michael Maier, Markus Beckers, Uwe Naumann, Amin Ghobeity, Alexander Mitsos: Compiler-Generated Subgradient Code for McCormick Relaxations

2011-26 Hongfei Fu: The Complexity of Deciding a Behavioural Pseudometric on Probabilistic Automata

2012-01 Fachgruppe Informatik: Annual Report 2012

2012-02 Thomas Heer: Controlling Development Processes

2012-03 Arne Haber, Jan Oliver Ringert, Bernhard Rumpe: MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems

2012-04 Marcus Gelderie: Strategy Machines and their Complexity

2012-05 Thomas Ströder, Fabian Emmes, Jürgen Giesl, Peter Schneider-Kamp, and Carsten Fuhs: Automated Complexity Analysis for Prolog by Term Rewriting

2012-06 Marc Brockschmidt, Richard Musiol, Carsten Otto, Jürgen Giesl: Automated Termination Proofs for Java Programs with Cyclic Data

2012-07 André Egners, Björn Marschollek, and Ulrike Meyer: Hackers in Your Pocket: A Survey of Smartphone Security Across Platforms

2012-08 Hongfei Fu: Computing Game Metrics on Markov Decision Processes

2012-09 Dennis Guck, Tingting Han, Joost-Pieter Katoen, and Martin R. Neuhäußer: Quantitative Timed Analysis of Interactive Markov Chains

2012-10 Uwe Naumann and Johannes Lotz: Algorithmic Differentiation of Numerical Methods: Tangent-Linear and Adjoint Direct Solvers for Systems of Linear Equations

2012-12 Jürgen Giesl, Thomas Ströder, Peter Schneider-Kamp, Fabian Emmes, and Carsten Fuhs: Symbolic Evaluation Graphs and Term Rewriting — A General Methodology for Analyzing Logic Programs

2012-15 Uwe Naumann, Johannes Lotz, Klaus Leppkes, and Markus Towara: Algorithmic Differentiation of Numerical Methods: Tangent-Linear and Adjoint Solvers for Systems of Nonlinear Equations

2012-16 Georg Neugebauer and Ulrike Meyer: SMC-MuSe: A Framework for Secure Multi-Party Computation on MultiSets

2013-01 * Fachgruppe Informatik: Annual Report 2013

2013-03 Markus Towara and Uwe Naumann: A Discrete Adjoint Model for OpenFOAM

2013-04 Max Sagebaum, Nicolas R. Gauger, Uwe Naumann, Johannes Lotz, and Klaus Leppkes: Algorithmic Differentiation of a Complex C++ Code with Underlying Libraries

2013-05 Andreas Rausch and Marc Sihling: Software & Systems Engineering Essentials 2013

2013-06 Marc Brockschmidt, Byron Cook, and Carsten Fuhs: Better termination proving through cooperation

* These reports are only available as a printed version.
Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.