Gesellschaft für Informatik e.V. (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in cooperation with GI and to publish the annual GI Award dissertation.

Broken down into
• seminars
• proceedings
• dissertations
• thematics
current topics are dealt with from the vantage point of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure high quality contributions.

The volumes are published in German or English.

Information: http://www.gi.de/service/publikationen/lni/

ISSN 1617-5468
ISBN 978-88579-292-5

This volume contains papers from the Software Engineering 2012 conference held in Berlin from February 27th to March 2nd 2012. The topics covered in the papers range from software requirements, technologies or development strategies to reports that discuss industrial project experience.

Stefan Jähnichen, Axel Küpper, Sahin Albayrak (Hrsg.): Software Engineering 2012

198

GI-Edition

Lecture Notes in Informatics

Stefan Jähnichen, Axel Küpper, Sahin Albayrak (Hrsg.)

Software Engineering 2012

Fachtagung des GI-Fachbereichs Softwaretechnik

27. Februar – 2. März 2012 in Berlin

Proceedings

Stefan Jähnichen, Axel Küpper, Sahin Albayrak (Hrsg.)

Software Engineering 2012

Fachtagung des GI-Fachbereichs Softwaretechnik

27. Februar - 2. März 2012
in Berlin

Gesellschaft für Informatik e.V. (GI)

Lecture Notes in Informatics (LNI) - Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-198
ISBN 978-3-88579-292-5
ISSN 1617-5468

Volume Editors
Stefan Jähnichen
Technische Universität Berlin / Softwaretechnik
Sekr. TEL 12-3, Ernst-Reuter-Platz 7, 10587 Berlin, Germany
Email: stefan.jaehnichen,@tu-berlin.de
Axel Küpper
Telekom Innovation Laboratories / Service-centric Networking
Sekr. TEL 19, Ernst-Reuter-Platz 7, 10587 Berlin, Germany
Email: axel.kuepper@tu-berlin.de
Sahin Albayrak
Technische Universität Berlin / DAI-Labor
Sekr. TEL 14, Ernst-Reuter-Platz 7, 10587 Berlin, Germany
Email: sahin.albayrak@dai-labor.de

printed by Köllen Druck+Verlag GmbH, Bonn

Software: the New Driving Force

so formuliert die Business Week bereits am 24. Februar 1983 auf ihrer Titelseite. In dem dazugehörigen Artikel werden der damalige Stand der Softwaretechnik und vor allem ihre Perspektiven dargestellt. Viele der damals aufgestellten Thesen sind noch heute richtig und belegen damals wie heute die Bedeutung der Softwaretechnik.

Software spielt eine zentrale Rolle bei der Nutzung innovativer Technologien und ihre Qualität ist für die Wertschöpfung der Produkte ein entscheidender Faktor. Sie muss höchsten Qualitätsanforderungen entsprechen und trotzdem kostengünstig hergestellt werden. Zuverlässigkeit, Sicherheit und Akzeptanz beim Nutzer sind Eigenschaften, die wir von moderner Software erwarten. Die Techniken zu ihrer Gewährleistung sind zwar zwischenzeitlich in hohem Maße professionalisiert und weiterentwickelt, bedürfen aber wegen der immer noch steigenden Komplexität der Systeme und der Integration mit anderen Technologien noch immer intensiver Forschung und weiterer Professionalisierung.

Die Software Engineering-Tagungsreihe wird vom Fachbereich Softwaretechnik der Gesellschaft für Informatik e.V. getragen. 2012 hat die Fakultät IV der TU Berlin die Gestaltung und Organisation der Tagung übernommen. Die SE 2012 bietet im Hauptprogramm sowohl eingeladene wissenschaftliche Vorträge als auch vom PC begutachtete detaillierte Berichte über laufende Forschungsarbeiten und -ergebnisse. Darüber hinaus werden in Praxisvorträgen am Industrietag aktuelle Problemstellungen, Lösungsansätze und gewonnene Erfahrungen präsentiert und zur Diskussion gestellt. Vor dem Hauptprogramm der Konferenz finden 4 Workshops sowie 3 Tutorials zu aktuellen, innovativen und praxisrelevanten Themen im Software Engineering statt. Abgerundet wird das Programm durch ein Doktorandensymposium, auf dem promovierende junge Wissenschaftlerinnen und Wissenschaftler ihre Arbeiten vorstellen und von erfahrenen Forscherinnen und Forschern konstruktive Rückkopplung zu ihren Dissertationsvorhaben erwarten. Fast schon traditionell ist ein Lehrertag in die SE 2012 integriert, auf dem sich Informatiklehrerinnen und -lehrer über neue Ansätze der Softwaretechnik für den Schulunterricht informieren.

Die Durchführung der Tagung Software Engineering 2012 wäre ohne die Mitwirkung der Sponsoren und vieler engagierter Personen nicht möglich gewesen. Ich bedanke mich daher bei allen Sponsoren, vor allem aber auch bei den vielen Helfern und Helferinnen der SE 2012:

- für die Gestaltung des Industrietags Axel Küpper, Ulrich Bareth und Sahin Albayrak

- für die Gestaltung des Doktorandensymposiums Petra Hofstedt und Claus Lewerentz

- für die Koordination der Workshops/Tutorials Holger Schlingloff und Bernhard Rumpe

Der Lehrertag wurde von der GI-Fachgruppe "Informatik-Bildung in Berlin und Brandenburg" organisiert. Dank gilt hier insbesondere Helmut Witten, der als Sprecher der Fachgruppe die Koordination übernommen hat.

Dank auch an alle Mitglieder des Programmkommittees, des Steering Committes, an den Fachbereich Softwaretechnik und die Organisatoren bei der Geschäftsstelle der GI und der bwo Marketing GmbH.

Und letztlich ganz besonderer Dank an alle meine Mitarbeiter, insbesondere Marc-Oliver Reiser, Doris Fähndrich, Marcus Mews, Andreas Mertgen und Steffen Helke.

Ohne diese viele Unterstützung wäre die SE 2012 nicht möglich.

Berlin, im Februar 2012

Stefan Jähnichen, Tagungsleiter

Tagungsleitung
Stefan Jähnichen, TU Berlin/FhG FIRST

Leitung Industrietag
Axel Küpper, TU Berlin/T-Labs
Sahin Albayrak, TU Berlin/DAI-Labor

Leitung Workshops und Tutorials
Bernhard Rumpe, RWTH Aachen
Holger Schlingloff, HU Berlin/FhG FIRST

Tagungsorganisation
Doris Fähndrich, TU Berlin
Steffen Helke, TU Berlin
Andreas Mertgen, TU Berlin
Marcus Mews, TU Berlin
Mark-Oliver Reiser, TU Berlin

Programmkomitee
Steffen Becker, Universität Paderborn
Klaus Beetz, Siemens AG
Manfred Broy, TU München
Bernd Brügge, TU München
Jürgen Ebert, Universität Koblenz-Landau
Gregor Engels, Universität Paderborn
Martin Glinz, Universität Zürich
Michael Goedicke, Universität Duisburg-Essen
Volker Gruhn, Universität Duisburg-Essen
Jens Happe, SAP
Wilhelm Hasselbring, Christian-Albrecht-Universität Kiel
Maritta Heisel, Universität Duisburg-Essen
Stefan Jähnichen, TU Berlin/FhG FIRST
Matthias Jarke, RWTH Aachen
Gerti Kappel, TU Wien
Udo Kelter, Universität Siegen
Jens Knoop, TU Wien
Heiko Koziolek, ABB
Claus Lewerentz , BTU Cottbus
Horst Lichter, RWTH Aachen
Peter Liggesmeyer, TU Kaiserslautern
Oliver Mäckel, Siemens AG
Florian Matthes, TU München
Oscar Nierstrasz, Universität Bern
Andreas Oberweis, KIT/FZI Karlsruhe
Barbara Paech, Universität Heidelberg
Peter Pepper, TU Berlin
Klaus Pohl, Universität Duisburg-Essen

Alexander Pretschner, KIT, Karlsruhe
Ralf Reussner, KIT/FZI Karlsruhe
Matthias Riebisch, TU Ilmenau
Andreas Roth, SAP
Bernhard Rumpe, RWTH Aachen
Thomas Santen, European Microsoft Innovation Center
Wilhelm Schäfer, Universität Paderborn
Klaus Schmid, Universität Hildesheim
Kurt Schneider, Leibniz Universität Hannover
Andy Schürr, TU Darmstadt
Rainer Singvogel, msg systems AG
Stefan Tai, KIT/FZI Karlsruhe
Andreas Winter, Universität Oldenburg
Mario Winter, Fachhochschule Köln
Uwe Zdun, Universität Wien
Andreas Zeller, Universität des Saarlandes
Heinz Züllighoven, Universität Hamburg
Albert Zündorf, Universität Kassel

Weitere Gutachter

Ulrike Abelein
Petra Brosch
Erik Burger
Alarico Campetelli
Sebastian Eder
Tobias George
Veit Hoffmann
Christian Janiesch
Ruben Jubeh
Andreas Koch
Joern Koch
Rudolf Koster
Anne Koziolek
Martin Küster
Thomas Kurpick
Achim Lindt
Lars Patzina
Uwe Pohlmann
Hanna Remmel
Jorge Ressia
Johannes Rost
Karsten Saller
Martin Schindler
Frederik Schulz
Norbert Seyff
Michael Striewe
Nelufar Ulfat-Bunyadi
Martin Wieber
Dustin Wüest
Gabriele Zorn-Pauli

Offizieller Veranstalter

Fachbereich Softwaretechnik der Gesellschaft für Informatik (GI)

Mitveranstalter

Technische Universität Berlin

Sponsoren

PLATIN

GOLD

SILBER

BRONZE

Inhaltsverzeichnis

Eingeladene wissenschaftliche Vorträge

Thomas Santen
Herausforderungen und neue Technologien zur Verifikation nebenläufiger verteilter Systeme 17

Walter Tichy
Die Multicore-Transformation und ihre Herausforderung an die Softwaretechnik 19

Andreas Schroeder, Martin Wirsing
Developing Physiological Computing Systems: Challenges and Solutions 21

Präsentationen des Industrietages

Philipp Sprengholz, Ursula Oesing
Durchführung von Modultests durch den Auftraggeber in Softwareentwicklungsprojekten mittels jCUT 39

Birgit Boss
SW-Architektur, SW-Sharing & Standardisierung 41

Dehla Sokenou
Softwarearchitektur für Geschäftsanwendungen auf Basis von OSGi 43

Jan-Peter Richter, Marion Kremer
TechnoVision: Neue Technologien gezielt für den Unternehmenserfolg nutzen 45

Marc Segelken
Anforderungen auf Konsistenz überprüft – Formalisierung hilft 47

Ralf Engelschall
Software-Architektur und Open-Source-Lizenzrecht in Einklang bringen 49

Simon Giesecke, Niels Streekmann
Architekturmanagement für eine föderale Produktlinienarchitektur 51

Torsten Frank
Trusted Cloud im Gesundheitswesen mit TRESOR 53

Stefan Buschner
Datenschutz und Usability bei Smartcards: Card-to-Card-Authentication 55

Forschungsarbeiten

Dirk Riehle, Carsten Kolassa, Michel A. Salim
Developer Belief vs. Reality: The Case of the Commit Size Distribution 59

Marco Konersmann, Azadeh Alebrahim, Maritta Heisel, Michael Goedicke, Benjamin Kersten
Deriving Quality-based Architecture Alternatives with Patterns 71

Glib Kutepov
Improving the software architecture design process by reusing technology-specific experience 83

Pit Pietsch, Hamed Shariat Yazdi, Udo Kelter
Controlled Generation of Models with Defined Properties 95

Tim Frey
Hypermodelling for Drag and Drop Concern Queries 107

Wolfgang Goerigk, Wilhelm Hasselbring, Gregor Hennings, Reiner Jung, Holger Neustock, Heiko Schaefer, Christian Schneider, Elferik Schultz, Thomas Stahl, Reinhard Von Hanxleden, Steffen Weik, Stefan Zeug
Entwurf einer domänenspezifischen Sprache für elektronische Stellwerke 119

Andreas Mertgen
Generic Roles for Increased Reuseability 131

Harald Cichos, Malte Lochau, Sebastian Oster, Andy Schürr
Reduktion von Testsuiten für Software-Produktlinien 143

Erfahrungsberichte

Mario Winter, Karin Vosseberg, Andreas Spillner, Peter Haberl
Softwaretest-Umfrage 2011 - Erkenntnisziele, Durchführung und Ergebnisse 157

Daniel Merschen, Yves Duhr, Thomas Ringler, Bernd Hedenetz, Stefan Kowalewski
Model-Based Analysis of Design Artefacts Applying an Annotation Concept 169

Christian Hopp, Fabian Wolf, Holger Rendel, Bernhard Rumpe
Einführung eines Produktlinienansatzes in die automotive Softwareentwicklung am Beispiel von Steuergerätesoftware 181

Workshops

Frank Huch, Janis Voigtländer
5. Arbeitstagung Programmiersprachen (ATPS 2012) 195

Birgitta König-Ries, Volkmar Pipek, Jens Pottebaum, Stefan Strohschneider
IT-Unterstützung für Public Safety & Security: Interdisziplinäre Anforderungsanalyse, Architekturen und Gestaltungskonzepte (IT4PSS 2012) 196

Andreas Birk, Florian Markert, Sebastian Oster
Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation (PIK 2012) 197

Michaela Huhn, Stefan Gerken, Carsten Rudolph
Zertifizierung und modellgetriebene Entwicklung sicherer Software (ZeMoSS 2012) 198

Eingeladene wissenschaftliche Vorträge

Herausforderungen und neue Technologien zur Verifikation nebenläufiger verteilter Systeme

Thomas Santen

Microsoft Research Advanced Technology Labs Europe
Europäisches Microsoft Innovations Center GmbH
Ritterstrasse 23
52072 Aachen
thomas.santen@microsoft.com

Abstract: Nebenläufige und verteilt arbeitende Softwaresysteme sind nicht neu. Betriebssysteme sind seit mehr als 20 Jahren inhärent nebenläufig. Verteilte Datenbanken werden mindestens genauso lange erfolgreich eingesetzt. Neue Entwicklungen wie Multi- und Many-Core Architekturen, Cloud und Ubiquitous Computing oder die immer größer werdenden Anforderungen an Performanz, Energieeffizienz und andere Qualitätseigenschaften stellen jedoch die Effektivität etablierter Qualitätssicherungstechniken in Frage. Insbesondere ist systematisches Testen immer weniger geeignet, die erforderliche Qualität solcher Systeme adäquat und kosteneffizient zu sichern. In dieser Situation stellen leistungsfähige modellbasierte und formale Verifikationstechniken eine attraktive Alternative dar. Der Vortrag zeigt, wie diese Techniken bereits heute in die industrielle Praxis der Softwareproduktion wirken.

Die Multicore-Transformation und ihre Herausforderung an die Softwaretechnik

Walter F. Tichy

Institut für Programmstrukturen und Datenorganisation
Karlsruher Institut für Technologie
Am Fasanengarten 5
76131 Karlsruhe
tichy@kit.edu

Abstract: Mehrkern-Rechner sind im Begriff, zu übernehmen. Mobiltelefone werden bereits mit Doppelprozessoren ausgestattet, Tablettrechner mit vier Prozessoren, PCs mit acht und mehr, Server mit Dutzenden, und GPUs mit einem halben Tausend. Diese Entwicklung stellt die Softwaretechnik vor die Herausforderung, parallele Software mit der gleichen Qualität und zu gleichen Kosten wie sequentielle herzustellen. Dieser Vortrag diskutiert einige der Forschungsthemen, die auf diese Herausforderung reagieren. Zu aller erst ist die Frage nach der Parallelisierungsmethodik: Wie findet man die Komponenten in neuer oder existierender Software, die von Parallelisierung profitieren? Was sind brauchbare Parallelisierungsmuster, Algorithmen, Daten- und Steuerungsstrukturen? Was sind geeignete Programmiersprachenerweiterungen, um Parallelität verständlich auszudrücken? Wie entdeckt man Parallelisierungsfehler oder vermeidet sie von vorne herein? Wie findet man eine gute Konfiguration der vielen neuen Parameter, die man in einem parallelen Programm berücksichtigen muss? Wie garantiert man Leistung? Und schließlich: Wie kommt man mit der zunehmenden Heterogenität der Prozessoren zurecht? Ohne Anspruch auf Vollständigkeit zeigt dieser Vortrag einige Forschungsrichtungen auf, die helfen können, Parallelprogrammierung zur Routine zu machen.

Developing Physiological Computing Systems: Challenges and Solutions

Extended Abstract

Andreas Schroeder, Martin Wirsing
Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstr. 67
80538 München
{andreas.schroeder, wirsing}@ifi.lmu.de

Abstract: Today's computing systems provide all kinds of media output to the users including pictures, sounds, lights and visual animations; but user input is mostly restricted to narrow, time-consuming input modes such as mouse, touch screen or keyboard, even if the system could sense this information implicitly from the human body. Physiological computing allows one to re-balance this information asymmetry by considering also physiological data of the users as input during on-line processing. Using such inputs, physiological computing systems are becoming able to monitor, diagnose and respond to the cognitive, emotional and physical states of persons in real time.

In this paper we give an introduction to physiological computing, discuss the challenges that typically arise when developing physiological computing systems, and solutions that we found promising in the creation of case studies. More specifically, we discuss the concepts software frameworks should incorporate to provide guidance in the development process: component-orientation, data processing support, and distribution support were found useful when dealing with physiological computing systems, where real-time sensing, on-line data processing, actuator control, context-awareness and self-adaptation are involved. Furthermore, we discuss why agile software development methodologies seem appropriate to structure development efforts involving physiological computing. Finally, we cover the issues in validation and verification of physiological computing systems that arise from the characteristics of the application domain, and discuss how empirical validation through psychological experiments, software validation and verification can interact with each other in the domain of physiological computing.

1. Introduction

Today's computing systems and infrastructures are constantly failing to satisfy the increasing expectations of everyday users. This inability has several causes, one of which is the asymmetry and shortcomings of current communication channels between computer systems and users [No07].While the output of computer systems features a high bandwidth (visuals, audio, even tactile vibrations are used as output channels), the input to computers is still fairly limited. Even though touch screens are beginning to

enter everyday use through smartphones and tablets, communication paths that are used in human-to-human interactions through facial expression and gesture are not available in human-computer interaction. Instead, computers mostly offer keyboard and mouse as input devices – with speech input just now being added to the picture. Together with touch screens and general forms of button-pushing, they constitute virtually the complete set of input possibilities that modern computer systems offer. It is a very limited input channel, still, and entails that computer systems remain unaware of the environment, the state of the user and his goals. This unawareness is often perceived as stubbornness and dullness. As users learn to adapt to this behaviour of their computer, an odd phenomenon can be observed: the command chain between computer and user is virtually inverted [No07, Fa09]. The flexible and smart user subdues himself to the obstinate and dense computer in order to achieve his own goal with the help of the limited machine available. Giving the computer awareness of the context and user will alleviate the imbalance of communication between user and machine and the entailed inversion of command. Computers that are aware of the user and the environment can be made able to detect their own wrong-doings easier and gain the possibility to compensate their actions. Extending the communication bandwidth from the user to the computer also promises to offer means for capturing motivations and goals of the user even on a subconscious level. This would allow to foresee a user's decision and conflicts with the current system operation and to adjust the system operation smoothly, making interaction with the everyday computing facilities of the user more enjoyable.

This paper is based on the results of the EU project REFLECT [IST-2007-215893; http://reflect.pst.ifi.lmu.de] and the PhD thesis of the first author [Sc12]. We start with giving an overview on physiological computing and its challenges in Section 2. The specifics of data processing, abstraction, and the creation of adaptation logic as feedback loops are discussed in Sections 2.2, 2.3, and 2.4 respectively. Before detailing these issues, we present a case study to which we contributed in its development: the Intelligent Co-Driver is introduced in Section 2.1. In Section 3, we discuss how a software framework can support the development of physiological computing systems. In this discussion, we present engineering concepts that were found to be useful in the creation of the Intelligent Co-Driver; we take our insights from the construction of the REFLECT framework [Wi11, Sc12] that was widely adopted in the creation of this physiological computing system. Insights on the benefits of a component-based approach (Sec. 3.1), support for data processing (Sec. 3.2) and distribution (Sec. 3.3) are elaborated. Beyond software framework support, we also discuss how the development of physiological computing systems can benefit from an agile approach in Section 4. We specifically elaborate on Scrum as software development approach for physiological computing systems in Section 4.1. In Section 4.2, we also discuss the challenges of validation and verification that arise in physiological computing and how empirical validation from psychological research can be leveraged and interplay with software and system development efforts. Finally, we present related work in Section 5 and conclude in Section 6.

2. Physiological Computing

Physiological computing [Fa09] is a relatively new area of research that tries to provide more input to computing systems – namely, physiological input – in order to alleviate the communication asymmetry that state-of-the-art computing systems feature. It finds itself at the intersection between artificial intelligence, human-computer interaction and applied psychology. The term of physiological computing itself is integrative since it denotes all research about computing systems that measure physiological signals from the human body in with the intent of processing it in real-time and leverage it in on-line operation; physiological computing is defined by the continuous on-line processing of physiological data. The continuous stream of processed data can be used to create a more complete picture of the operational context of a system – especially of an assistive system – and can enable continuous and discrete adaptation and improvement of the operational environment of their users, thereby improving the user's environment. As a physiological computing system monitors the user's state that it improves, it creates a feedback loop involving both human users and computers. This feedback loop has therefore been coined the *bio-cybernetic loop* [SF09]. Applications featuring this feedback loop can produce a more accurate representation of the user's operational context through synthesis of information available through the computing infrastructure and environmental sensors, and information deducible from sensors placed on the human body (see Figure 1).

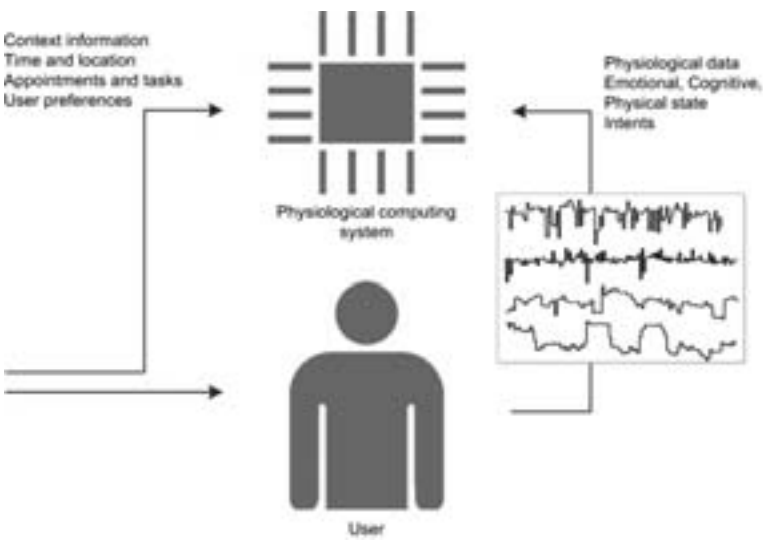


Figure 1: Bio-cybernetic feedback loop structure

While the definition of physiological computing does not specify any kind of computing infrastructure it should be running on, a convenient infrastructure for physiological computing are not desktop computers, but computing infrastructures that permeate the environment and are capable of controlling and altering parameters of the environment such as illumination, temperature, and background music, as well as providing

contextual information directly to the user. Those infrastructures also have the benefit of allowing access to physiological parameters, as part of the infrastructure may become wearable as gadgets or jewels, integrated in clothes and even in the human body [BC00]. At the same time, a computing infrastructure that offers software controls for a significant part of the environment promises to provide meaningful means for improving the experience of the user.

In physiological computing, there are a number of challenges that have to be faced with in the creation of new systems. Before discussing them in Sections 2.2, 2.3 and 2.4, we first give an example of a physiological computing application in the automotive domain: the intelligent co-driver.

2.1 The Intelligent Co-Driver

The intelligent co-driver is an example physiological computing system that was realized within the REFLECT project ([RV11], also named the REFLECT Automotive Demonstrator). The goal in the creation of this physiological computing system was to construct a supportive environment for the driver that mimics the behaviour of a co-driver taking care of the well-being of the driver. The intelligent co-driver should extend the driver's awareness for the driving situation as well as for his own well-being and his actions in a non-obtrusive and discrete way. The intelligent co-driver therefore consists of three parallel bio-cybernetic feedback loops that monitor and guide the driver's cognitive load, emotions, and physical comfort.

Figure 2: Intelligent co-driver cockpit deployment

The *Effort loop* is concerned with determining the mental workload of the driver and adjusting the driver's tasks accordingly. In the automotive demonstrator, the effort loop should reduce the number of secondary tasks the driver is asked to perform, i.e. in the automotive context, incoming mobile phone calls must be suppressed, and music currently playing is faded out to allow the driver to focus on the (currently demanding) primary task of driving. The indicators for mental workload that were usable in the automotive setting were determined to be heart rate and heart rate variability. High mental workload translates to a slightly increased heart rate, and a decrease in heart rate

variability (i.e. the heart beats become more regular under high mental workload) [Mu92].

The *Emotion loop* is involved in managing the emotion of the driver according to a driver-selected goal. The goal state can be selected from the following three emotional goal states (for simplicity of the user interface): energetic, neutral, and relaxed. The emotional loop continuously monitors the current emotional state of the driver, and uses music from the driver's own music database to guide him to the desired goal state. For monitoring the driver's emotion, skin conductance level and skin temperature level features were selected that must be implemented in the demonstrator. Furthermore, the emotional loop requires the creation of a music database that stores the song effects on the driver, and a music player that selects songs according to the recorded song effects and the selected target mood. The concept of emotional control through personalised music, and the filtering processes required to detect the effects on the emotional state of the user are described by Janssen et al. in [Ja09].

The *Comfort loop* tracks the physical comfort the user is experiencing, and is able to adjust the driver seat moderately to improve on the sitting posture and stability of the driver. Physical comfort of the driver is tracked by the pressure pattern the driver is exerting on the seat, as well as by measuring the accelerations (longitudinal, lateral, and vertical) that the driver is experiencing through either fast driving or driving on a rough road [Be10, Be11]. Improving the sitting posture is achieved by inflating or deflating cushions that are available in the driver seat. Inflating the cushions result in more stabilised sitting, while deflated cushions allow more freedom to move. Additionally, the comfort loop monitors the driver's alertness to the road and should give indications of the user's drowsiness. Drowsiness is detected through eye blink duration, as blink durations increase significantly with driver drowsiness [Ca03].

The intelligent co-driver was implemented into the cockpit of a Ferrari California car (see Figure 2), during the REFLECT project , and evaluated and demonstrated at the end of the project in spring 2011. In the creation of the intelligent co-driver, a number of technical challenges were faced that we describe in the following.

2.2 Input Data

A first task in constructing physiological computing systems is creating processing code for physiological input data. The challenges here are dealing with high sampling rates, noise and interference in the signals, and addressing interpersonal variations.

Raw physiological data comes at sampling rates between 30 Hz (for skin conductance or skin temperature signals, for example) and 10 kHz (for EEG and ECG, for example). Due to the high sampling rate, some input signals require fast pre-processing code written in assembly or C, before it can be fed at lower sampling rates into systems written in high-level languages such as Java, C++, or Python. For example, skin conductance and skin temperature signals can be easily treated in Java directly, while ECG signals are normally pre-processed in C or assembly, and only extracted features (such as heart beats) are handled at much lower sampling rates in high-level languages.

Physiological input data, being physical data, also features varying levels of noise and interference that needs to be cleaned, or detected and accounted for. For example, skin conductance and ECG signals are significantly affected by sensor movements: the skin conductance level measured depends heavily on the skin trajectory length, and loose contacts create epochs of missing signals. In comparison, contact skin temperature sensors are fairly robust with respect to movement.

Finally, the absolute input values and amplitudes related to psychological reactions may vary significantly from person to person [Pi97] and may be heavily dependent on sensor placements. For example, skin conductance levels are known to be heavily dependent on individuals and placement of sensors, and therefore only relative values are used. Skin conductance is often compared to a baseline giving a relative scale through both mean and standard deviation of the baseline. The current skin conductance value is then normalized, i.e. given as the number of standard deviations it departs from the mean of the baseline. Heart beats on the other hand feature no dependence on sensor placement and a relatively low level of interpersonal variation. Here, recording a prolonged baseline of a person's heart rate and adjusting thresholds to this baseline may be sufficient for taking interpersonal variations into account.

2.3 Abstraction

Once the issues of processing the raw input signals are addressed, the next step in the design of a bio-cybernetic feedback loop involves the creation of the proper abstraction from the input signals. Here, the current state of the user needs to be inferred from the available data and transformed into a psychological model to represent the user state. The user state representation must include all information needed for decision-making in the bio-cybernetic loop. The psychological model may for example define that the emotional state of the user must be captured in terms of a valence scale ranging from -1 to 1 (negative or positive emotion), and arousal scale ranging from 0 to 1 (calm or energetic), as in the case of the emotional loop of the Intelligent Co-Driver (cf. Sec. 2.1).

The creation of the current user psychological state from physiological input features is however not achievable through feature extraction and direct mapping [Fa09]; A signal in physiological input data can be produced by several internal body processes. An increase in heart rate for example can be due to either physical exercise, or due to high mental workload and cognitive engagement. A decrease in skin temperature may be due to a positively valenced emotion, or due to a decrease in environmental temperature. Mapping from features extracted from physiological input data to psychological state is a complex process requiring careful consideration of input quality, means for associating signals to their cause, and accounting for the uncertainty in this association. In this abstraction process, techniques from artificial intelligence can be used to great benefits. In the mapping from skin temperature and skin conductance to emotional valence and energy, for example, Bayesian networks were used successfully to account for the uncertainty through probabilistic weighting.

2.4 Feedback loop

Once the algorithms for creating psychological state from physiological inputs are defined, the next step is to create the application logic leveraging the available information, adapting the environment, and adjusting its own behaviour. These activities can be based solely on the psychological model of the user, but it may well be useful to complement the user model with a model of the environment, containing information about the user context describing for example the user location and current activities that the user needs to perform.

Based on the available information, the physiological computing system makes informed decisions on the adaptation of the environment. In this adaptation process, physiological information can be used on different levels of meta-operation. On a basic level, physiological information can be used to provide a direct feedback, as in the case of bio-feedback systems that inform the user about his current psychological state to induce self-regulation. One meta-level up, the physiological information can be used to direct the operation of the system. For example, the emotional state of a user can be used as input to a music selection process, as in the emotional loop of the intelligent co-driver (cf. Section 2.1). On that level, the user does not see the physiological state the system inferred, but he experiences it directly through the behaviour of the system. One further meta-level up, the system can use physiological information for improving its own behaviour. For example, a physiologically enhanced game may infer the intensity of the gaming experience, and adjusts challenges in the game to create a sinuous wave of game intensity level, i.e. altering relaxing with highly intense game phases (Ambinder [Am11] presents a variation of the game Left 4 Dead that uses this concept). While further meta-levels may be interesting to contemplate, we found no use for these levels in applications so far, and we therefore restrict our discussion of feedback loops (and framework support for them) on these three types, which we name the *bio-feedback type*, the *operational type*, and the *reflective type*.

A major challenge in the creation of bio-cybernetic feedback loops is their validation and testing. Before deploying a physiological computing system to a larger audience, we need to ensure that the system operates as expected, and exhibits the desired level of adaptivity. However, as the knowledge about reliable inference means from physiological data to psychological constructs and working bio-cybernetic feedback loops is not yet settled, the creation and validation of physiological computing concepts and applications require a significant amount of experimenting, prototyping, and basic trial-and-error. When thinking about possible framework support, devising support for these activities is therefore essential.

3. Framework support

Creating physiological computing applications involves a significant amount of challenges and problems to solve. A software framework can assist development efforts by offering guidance in the development process, offering a structure to follow, and embodying best practices in its architecture. In this section, we report on our experience

in creating the REFLECT framework [Wi11, Sc12] which embodies the concepts described in this section, and was widely adopted in the creation of the Intelligent Co-Driver introduced in Section 2.1.

3.1 Components

Using components [Sz02] as the underlying concept of a framework for physiological computing is a sensible choice, as it allows addressing several issues in the creation of bio-cybernetic feedback loops. First of all, using components allows developers to experiment with several solution variants. With a component approach, variants can be encapsulated behind a common interface, and the system configuration control also provides means to exchange variants. Hence, component orientation can simplify the creation of different prototypes and rapid switching between different software system setups. Of course, the work of defining variation points, creating the interface hiding variants, and defining and managing the system configurations creating the variants remains in the hand of the developers. Support for these activities may be given, but the activities themselves are very much specific to the physiological computing system developed, and therefore cannot be lifted from the developers.

Having a component-based system underneath may also allow leveraging reconfigurations [KM90, Me96] for the creation of bio-cybernetic feedback loops, especially for feedback loops of the reflective type (cf. Sec. 2.4). Reconfigurations denote changes of a system configuration at runtime and may allow altering component parameters, sending messages to components, removing existing components or adding new components to the configuration, as well as creating or deleting connectors between components. These means can be used to deliberately modify the system behaviour. By adjusting parameters of behaviour-controlling components or exchanging components involved in the feedback loop operation, the responses of the system can be altered to account for changes in the user's psychological state. As reconfigurations can be used to change component behaviour in reaction to physiological information, they facilitate the creation of bio-cybernetic feedback loops with meta-reasoning of the reflective type [Sc12].

In physiological computing applications, reconfigurations are not only useful for feedback loops of the reflective type, but can also be used in various other scenarios. Hot code updates for e.g. dynamic loading of drivers and patching of software can be realized using reconfigurations, as well as tracking of volatile devices and reacting to changes in the available device landscape. In the former scenario, means for runtime codebase modification are required, as not only new instances of known component types must be created, but components of previously unknown type. In some languages, such support is provided directly, while in others (such as Java), additional module platforms (such as OSGi) are required to allow for codebase modification at runtime. In the latter scenario, i.e. tracking available devices, reconfigurations can be used to model the available device landscape in the system configuration. Each device can be proxied by a component, and services offered by devices can be modelled as services of the component. In this way, it becomes possible to use reconfigurations for modelling the

response to the departure of a device, and for example to try to satisfy the requirements of dependent components by services from other available devices.

3.2 Data Processing

The management of physiological data and the inference of psychological information from that data is an area that can greatly benefit from a software infrastructure. First of all, we observed that providing a central storage location for physiological data and information derived from it can simplify the architecture of physiological computing systems. With a central data management facility, it becomes easier for software components to access, query and process physiological data. Furthermore, the central storage of physiological data simplifies the mocking of system parts: during experiments it becomes easier to stream physiological data and intermediate results to a persistent storage and to replay the stored data to the system with the goal of mocking sensors and analysis algorithms in test runs.

Also, we discovered that it is convenient to store physiological data in ordered sequences and tagging each data item with its time of creation. The creation time information allows creating auto-purging data sequences that discard old data exceeding a previously set maximum time to live; we call these auto-purging sequences *data windows*. Data windows can provide very simple and natural programming interfaces for computing features over the most recent data [Sc10], as e.g. computing the standard deviation in the last ten minutes of available skin conductance data may be performed by calling a standard deviation method on a data window containing the most recent ten minutes of data. As these computations may require a subset of the data available in data windows, creating (live) slices of data windows is often a necessity. Luckily, implementing slicing on data windows is straightforward as the data items are each tagged with their time of creation, and hence the slice boundaries are easily determined and updated.

In the processing chain from raw physiological inputs to psychological concepts and state, we can also observe that at least three different means of processing of physiological data and creation of abstractions are used. It is worthwhile for a software framework to envisage how all three types of processing can be supported. On the lowest level, polling is used to sample the physiological signal into a data stream of physiological data. Polling may well be suitable to create higher level abstractions with a similar constant (but lower) sampling rate. Also relatively low level is the processing of the data stream each time a new data item enters the data window. Here, a notification facility may allow performing lightweight computations directly in the process of publishing the data to the stream. Finally, the most high-level type of processing is the event-based query of data streams. Here, the data stream is not queried and processed regularly, but only as a reaction to events that emerge in the system. In general, the low-level processing of physiological data will start with a polling or notification-based approach, and higher levels will generate events from the data streams, or transform the data streams to higher-level representations on demand.

3.3 Distribution

As ubiquitous computing environments constitute a natural habitat for physiological computing, distribution issues need to be addressed by a software framework that is intended to support the development of physiological computing applications. Especially in system setups with narrow communication bandwidth, it is crucial that physiological data is processed – as far as possible – on the location of their collection, before sending a more compact, abstracted representation of the input data over the communication network. Nevertheless, streams of data need to be sent over the communication network. Here, a centralized data management facility as described in Section 3.2 allows the automatic replication of data to other computation nodes. The configuration of a computation node may declaratively request the local replication of data from a remote node, and even specify the update rate as well as the local cache size, enabling the framework to transparently replicate the requested data to the local node [Sc10, Sc12]. Also, the introduction of a (node-local) centralized data storage prepares the data processing subsystems for distribution. Once the data processing subsystem has been split up into components referring to their node-local data store, it becomes easier to distribute components to different nodes, and replicate the data between the centralized data stores.

In addition to the distribution of physiological data, creating a distributed physiological computing system faces the same challenges as ubiquitous computing systems do. They need to discover capabilities of the environments, and need to coordinate multiple nodes involved in the complete application. For this, applications need a generic distribution support; in the REFLECT framework, we offered a point-to-point message-based communication facility amongst remote components that allow for transparent communication between components on remote nodes, as well as an event broadcast facility that allow components to publish events and subscribe to event topics for notifications [Wi11, Sc12]. Here, providing control about whether events are distributed over the network and to which nodes they are delivered should allow for a sensible management of available network bandwidth.

Especially when adding distribution facilities to the framework, a question that often arises is how far the framework mechanisms should be made transparent to the framework user. Making distribution mechanisms fully transparent lifts the burden of managing distribution issues from the developer, but at the same time, it hinders the developer from controlling or influencing how the distribution mechanism operates. A fully transparent mechanism offers no control to the way the transparent mechanism work. On the other hand, a fully configurable mechanism puts the burden of configuration on the developer, and forces the developer to think about the mechanism, even if a standard configuration of the mechanism would suffice. Luckily, it is often very well possible to take the best of both worlds: providing a configurable transparent mechanism allows developers to control the inner workings of the mechanisms when needed, and rely on the default configuration of the mechanism if it is sufficient.

4. Agile Physiological Application Development

The development of physiological computing systems can benefit not only from the guidance a well-structured software framework provides, but also from a development approach that sensibly organizes all development activities. As the foundation for developing physiological computing applications, we propose an agile approach, the software development method Scrum. In the following, we first discuss the rationale for this proposal.

4.1 Scrum as Foundation

The domain of physiological computing is primarily characterised by its novelty and interdisciplinary nature: it adjoins to applied psychology, human computer interface research, and artificial intelligence. However, this novelty and interdisciplinary nature entails that a significant body of research has to be performed before knowledge about reliable inference of psychological state from physiological data and working bio-cybernetic feedback concepts can be considered as settled [Fa09]. Therefore, developing physiological computing applications requires a development approach able to deal with high complexity and considerable amounts of uncertainty and risks. As discussed in [SB02], constant inspection and adaptation is more important than detailed plans when dealing with high complexity and uncertainty in the development. In highly complex projects, creating detailed and far forward-looking plans constitute a largely wasteful activity, as knowledge established during the project will inevitably require substantial changes to every long-term project plan. The more detailed and complex the plans are the more effort needs to be put in the adjustment of the plan; the ever-changing nature the project plan will develop – if it is constantly updated at all – also diminishes the guiding value of detailed and far forward-looking plans. Restricting oneself to the feasible, i.e. concentrating the efforts to providing guidance for the immediate future becomes imperative. Here, Scrum's iterative nature and focus on quick feedback is a healthy guiding principle for developing physiological computing systems.

The basic structure of Scrum is shown in Figure 3. Scrum starts with a list of tasks that need to be completed to create a product, which is called the *product backlog*. From this list, the team assembles in a *sprint planning meeting*, and details a sprint backlog containing the product backlog items it estimates being feasible within a four week iteration called *sprint*. During the sprint, the team assembles for a 15 minutes *daily scrum meeting* every day to discuss its progress and immediate tasks. At the end of the sprint, the team presents its results in the *sprint review* to the stakeholders of the product to gather feedback about the product's adequateness and its ability to satisfy the users' needs. Scrum requires that the presented results constitute a *potentially shippable product increment*, that is to say, the result must have passed any quality assurance requirements imposed on the project and be of a quality that allows the product increment to be delivered to its users. The Scrum methodology provides also roles and underlying principles that guide the application and implementation of Scrum, which are not detailed here, but can be found in Schwaber's and Beedle's seminal book [SB02].

The constant availability of a potentially shippable product even in early development stages is a feature of Scrum that is of paramount importance in the domain of physiological computing. The highly experimental nature of development in physiological computing requires early available prototypes that can demonstrate the feasibility of the envisaged bio-cybernetic feedback loop. Scrum's iterative nature and focus on creating potentially shippable product increments is a perfect match for these

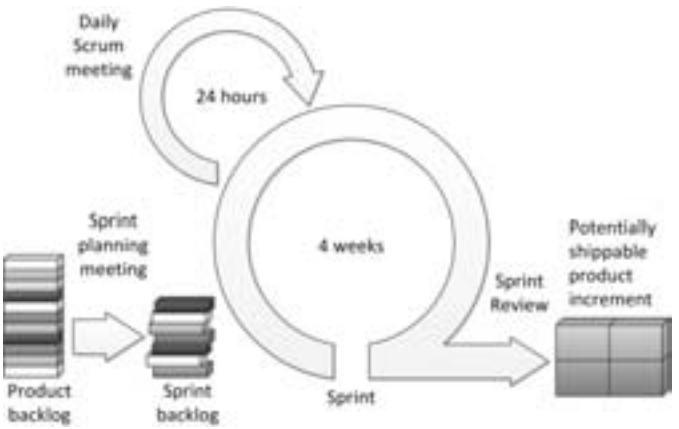


Figure 3: Scrum Overview

demands. In our experience in the creation of the intelligent co-driver (cf. Sec. 2.1), we discovered that constant feedback and testing with early versions of the product allowed to discover new development opportunities, guide the further development of the product by increasing confidence, and allow to guarantee a high quality of the created product by providing feedback on bugs and defects emerging in the real system setup.

4.2 Validation and Verification in Physiological Computing

Every software system must be validated to assure that it satisfies its user's needs. While this is true for general software systems, validating physiological computing systems brings additional challenges, and can benefit from specific techniques and additional sources of information. Physiological computing systems usually have non-functional requirements in the domain of security, privacy and safety that must be satisfied in addition to the functional requirements concerning the proper operation of the system.

The functional verification of a physiological computing system can be divided into two issues: first, the inference of psychological state from physiological data and second, the correct operation of the bio-cybernetic loop. For both issues, developers must validate both conceptual soundness and correctness of the implementation. Conceptual soundness can be established through empirical validation of the inference process or the bio-cybernetic loop. While validation of inference processes is now being performed more and more in applied psychology [Zw09, Ja11, FM11], the conceptual validation of bio-cybernetic loops needs to be established through extensive user tests.

However, the empirical validation provided in the psychology literature needs to be examined carefully. Psychological research validates concepts through controlled experiments that provide evidence of a statistically significant effect. As controlled experiments are executed in highly controlled environments and leverage highly sensitive medical-class sensory equipment, however, they are insufficient to establish evidence that systems based on the validated concepts operates as desired in their (uncontrolled) intended operation environment. Therefore, it is not enough to rely solely on results from the psychology literature for the validation of implementations. Additional validation of the concept implementation in the actual environment and using the final sensory equipment must be performed. On the other hand, operating without empirically validated concepts would amount to completely unguided trial-and-error, and the probability of finding a successfully operating bio-cybernetic feedback loop while handling the full complexity of the developed system (distribution, efficiency, etc) would be greatly diminished. Results of controlled experiments provide the necessary guidance in the creation of physiological computing systems.

A successful cooperation of psychological research and computer science with respect to validation can be seen in the creation of the emotional loop of the Intelligent Co-Driver. First, the creation of the emotional loop was based on validated inference algorithms for determining emotional valence and energy from skin conductance and skin temperature [Zw09]. Based on this knowledge and additional expertise in the acquisition of skin conductance and skin temperature measures, a first software prototype was created that selected songs based on current and target mood selection. This prototype was then used in psychological experiments, that is to say, in constrained controlled environments [Ja11, Zw11]. The experiments validated the soundness of the bio-cybernetic loop in controlled environments. Finally, user tests validated the applicability of the loop in its intended environment. During these validation steps, the prototype was gradually refined and improved based on the insights gained through experiments [RV11].

More established approaches to validation and verification, i.e. formal reviews, and verification approaches based of formal methods can also support the validation of physiological computing systems. For example, the creation of correctly behaving reconfigurable systems is especially challenging: while it may be easy to predict all possible reconfigurations given a specific system configuration, it becomes more and more difficult to foresee the potential architecture of a system once several reconfigurations have taken place; the interdependencies of multiple reconfiguration steps are hard to grasp intuitively. Therefore, we have created an assume-guarantee reasoning framework based on real-time linear temporal logic for reconfigurable systems [Sc11]. The real-time semantics allows verifying the correctness of reconfigurations in physiological computing systems, given a formal representation of the component-based design and its reconfiguration rules, and a globally desired behaviour. Here, we use a formal verification approach to provide a proof that the system exhibits the expected behaviour – depending on the specified behaviour, this may include a guarantee that the system adapts as expected, or that the system respects a limit of adaptation. Other domains in which formal verification techniques may be required are the validation and verification of safety and security requirements.

5. Related Work

Related work of development approaches for physiological computing systems does not yet exist, as it constitutes a novel field for software engineering research. Most closely related are methods for developing multi-agent systems and adaptive systems.

The DiVA methodology [De10] focuses on the requirements engineering and modelling aspects of adaptive features of software systems. It is primarily a model-driven, aspect-oriented approach in which an adaptive system is captured as a *base model* with *adaptation models* applied. The base model constitutes the least common denominator that the system consists of throughout all adaptation contexts, while the adaptation models represent aspects that are added in specific contexts only. Similar to our approach, DiVA proposes an iterative and incremental development process, but aspects may be difficult to use in highly dynamic settings where all sensor and actuator devices are considered volatile, and the control logic system is only instantiated if devices are available. The MUSIC approach [Ge11, Wa10] for developing ubiquitous self-adaptive applications is component-based. It focuses heavily on capturing adaptation contexts and system variants by so-called variability models. Domain models describe the relations between application contexts and component Quality of Service properties. In contrast to our proposal it follows a waterfall-like model-driven development process [Wa10].

Multi-agent-system development methodologies can be seen as another, different approach to the development of adaptive systems. In these development approaches (for an overview, see [HG05]), context modelling has a limited importance compared to the modelling of agent roles and interactions. To shed some light on these differences, we use the Gaia methodology [Za03] as one representative multi-agent based methodology. In this methodology, a software system is understood and modelled as an artificial society of cooperating entities (agents) that can fulfil different responsibilities and rights (roles) and interact with different communication partners at different times. In this view, understanding and modelling the agent roles, their associated capabilities and constraints, as well as the intended communication structures of roles, becomes the most significant activity. Other activities such as context understanding and modelling are only means to this end. In contrast to a plain component-based approach as advocated in this paper, the focus on roles entails a more normative and defensive view on the possible interactions of entities – one in which each agent can only be made accountable for its behaviour with respect to its current role. In our experience with physiological computing applications, the focus on potential role conflicts and role changes of agents seemed not to be necessary; instead, we found that such applications are easier to design and understand as a set of cooperating active components whose configuration may be altered at runtime.

6. Conclusion

Today's assistive systems and smart environments are increasingly failing short to satisfy the growing demands of their users. One reason for this shortcoming is found in

the lack of contextual information. Physiological computing has the potential of providing this much needed information to assistive systems and smart environments.

In this paper, we have investigated the software development challenges that physiological computing offers and potential solutions we see fitting. Handling physiological data, extracting the proper information, and creating adaptive systems that leverage the information made available constitute the major challenges in the creation of physiological computing systems. We have presented concepts and features that we found useful in the creation of physiological computing applications and that were successfully assembled into a software framework. Here, we have focussed specifically on the benefits of component orientation and reconfiguration, as well as data management and distribution facilities that a software framework can offer to simplify the development tasks at hand. We have also discussed why agile development approaches (Scrum specifically) are well suited for the creation of physiological computing systems. Finally, we have presented our experience in the validation and verification of physiological computing systems. We have argued that empirical validation and system validation must go hand in hand in order to create a successful physiological computing system, and that formal verification methods can find their application in the verification of the correct adaptation behaviour as well as in the validation of safety and security requirements.

This paper has given an initial overview on challenges and insights on solutions and approaches for the development of physiological computing systems that demand for more elaborate and precise study. The REFLECT framework [Wi11, Sc12] provides a first starting point for research in software framework support, but each of the concepts it embodies can benefit from deeper investigation. Similarly, the insights we provided on development methodology, verification and validation only discuss the issues and challenges we uncovered. Physiological computing is an interesting and exciting domain providing a significant amount of hands-on challenges for computer science research – so why not give it a try?

References

[Am11] Ambinder, M.: Biofeedback in gameplay: How valve measures physiology to enhance gaming experience. Game Developers Conference. 2011.

[BC00] Barfield ,W.; Caudell, T.: Fundamentals of wearable computers and augumented reality. Lawrence Erlbaum, 2000.

[Be10] Bertolotti G. M.; Cristiani, A.; Lombardi, R.; Ribaric, M.; Tomasevic, N.; Stanojevic, M.: Self-adaptive prototype for seat adaption. Fourth IEEE Int. Conf. Self-Adaptive and Self-Organizing Systems Workshop. IEEE, 2010. 136-141

[Be11] Bertolotti, G. M.; Lombardi, R.; Cristiani, A.; Stanojevic, M.; Ribaric, M.; Tomasevic, N.: Evaluation of Case Studies. Deliverable D5.3, REFLECT Third Year Report. Fraunhofer FIRST, 2011.

[Ca03] Caffier, P. P.; Erdmann, U.; Ullsperger, P.: Experimental evaluation of eye-blink parameters as a drowsiness measure. European J. of Applied Physiology, 89:319-325, 2003.

[De10] Dehlen,V.: DiVA Methodology. Deliverable 2.3, DIVA Project. SINTEF, 2010, 98 p. http://www.ict-diva.eu (accessed 2011/01/15).

[Fa09] Fairclough, S. H.: Fundamentals of physiological computing. Interacting with Computers, 21(1-2):133-145, 2009.

[FM11] Fairclough, S.H; Mulder, L.M.J.: Psychophysiological processes of mental effort investment. In Wright, R.; Gendolla, G.: How Motivation Affects the Cardiovascular Response: mechanisms and applications. APA, 2011.

[Ge11] Geihs K.; Evers, C.; Reichle, R.; Wagner, M.; Khan, M.U.: Development support for QoS-aware service-adaptation in ubiquitous computing applications. Proc. 2011 ACM Symposium on Applied Computing. ACM, 2011, 197-202.

[HG05] Henderson-Sellers, B.; Giorgini, P.: Agent-Oriented Methodologies. Idea Group Publishing, 2005.

[Ja09] Janssen, J.H.; van den Broek, E.L.; Westerink, J.H.D.M.: Personalized affective music player. 3rd Int. Conf. Affective Comp. and Intelligent Interaction. IEEE, 2009, 704-709.

[Ja11] Janssen, J.H.; van den Broek, E.L.; Westerink, J.H.D.M.: Tune in to your emotions: A robust personalized affective music player. User Modeling and User Adaptive Interaction. 2011, 1-25.

[KM90] Kramer, J.; Magee, J.: The evolving philosopher's problem: Dynamic change management. IEEE Transactions on Software Engineering, 16(11):1293-1306, 1990.

[Me96] Medvidovic , N.: ADLs and dynamic architecture changes. 2nd Int. Software Architecture and Int. Multiple Perspectives in Software Development Wshps. ACM, 1996, 24-27.

[Mu92] Mulder, L.J.M.: Measurement and analysis methods of heart rate and respiration for use in applied environments. Biological Psychology, 34(2-3):205-236, 1992.

[No07] Norman, D.A.: The Design of Future Things. Basic Books, 2007.

[Pi97] Picard, R.W.: Affective Computing. MIT Press, 1997.

[RV11] Ragnoni, A.; Visconti, A.: Pervasive Adaptive Demonstrator Implementation and Evaluation. Deliverable 6.4, REFLECT Third Year Rept. Fraunhofer FIRST, 2011, 64 p.

[SB02] Schwaber, K.; Beedle, M.: Agile Software Developmt. with Scrum. Prentice Hall, 2002.

[SF09] Serbedzija, N.B; Fairclough, S.H.: Biocybernetic loop: From awareness to evolution. IEEE Congress on Evolutionary Computation. IEEE, 2009, 2063-2069.

[Sc10] Schroeder, A.; Kroiß, C.; Mair, T.: Context acquisition and acting in pervasive physiological computing. 7th Int. ICST Conf. Mobile and Ubiquitous Systems, 2010.

[Sc11] Schroeder, A.; Bauer, S.S.; Wirsing, M.: A contract-based approach to adaptivity. Logic and Algebraic Programming, 80:180-193, 2011.

[Sc12] Schroeder, A.: Software engineering perspectives on physiological computing. PhD Thesis, Ludwig-Maximilians-Universität, 2012, .

[Sz02] Szyperski, C.; Gruntz, D.; Murer, S.: Component software: beyond object-oriented programming. Addison-Wesley Professional, 2002.

[Wa10] Wagner, M.: Modelling Notation and Software Development Method for Adaptive Applications in Ubiquitous Computing Environments. Deliverable 6.5, MUSIC Project, 2010, 126 p. http://ist-music.berlios.de (accessed 2011/01/15).

[Wi11] Wirsing, M.; Beyer, G.; Kroiß, C.; Schroeder, A.; Meier, M.: Middleware, Tools, and Evaluation. Deliverable D2.5, REFLECT Third Year Report. Fraunhofer FIRST, 2011, 39 p.

[Za03] Zambonelli, F.; Jennings, N.R.; Wooldridge, M.: Developing multi-agent systems: The Gaia methodology. ACM Transactions on Software Engineering and Methodology, 12:317-370, 2003.

[Zw09] van der Zwaag, M.D.; Westerink, J.H.D.M.; van den Broek, E.L.: Deploying music characteristics for an affective music player. 3rd Int. Conf. Affective Computing and Intelligent Interaction and Wshp. IEEE 2009, 1-7.

[Zw11] van der Zwaag, M.D.; Fairclough, S.H.; Spiridon, E.; Westerink, J.H.D.M.: The impact of music on affect during anger inducing drives. Proc. 4th Int. Conf. Affective Computing and Intelligent Interaction - Volume Part I. Springer, 2011, 407-416.

Präsentationen des Industrietages

Durchführung von Modultests durch den Auftraggeber in Softwareentwicklungsprojekten mittels jCUT

Philipp Sprengholz, Ursula Oesing

Fachbereich Wirtschaftsingenieurwesen
Fachhochschule Jena
Carl-Zeiss-Promenade 2
07745 Jena
s7phspre@stud.fh-jena.de
ursula.oesing@fh-jena.de

Abstract: Der Erfolg eines Softwareprojektes ist maßgeblich abhängig von der Einbindung des Auftraggebers, oder allgemeiner des Wissensträgers der geschäftsspezifischen Abläufe, in den Qualitätssicherungsprozess. Je früher er in die Prüfung von Teilentwicklungen einbezogen wird, umso eher können Fehler entdeckt und die Entwicklungen zur Zufriedenheit aller Beteiligten abgeschlossen werden. Um den Kunden in die Durchführung von Modultests zu integrieren, wurde an der Fachhochschule Jena die Java-Anwendung jCUT *(Customer Unit Testing Application for JUnit Tests)* entwickelt, mit der auf die von einem Entwickler definierten und parametrisierten JUnit-Testfälle ohne Kenntnis einer Programmiersprache oder Entwicklungsumgebung zugegriffen werden kann. Mittels jCUT kann der Kunde selbst über eine nutzerfreundliche Oberfläche die Testfälle variieren und überprüfen, ob sich eine zu testende Methode bei der Eingabe der aus seiner Sicht wichtigen Testparameterkombinationen wie gewünscht verhält. Weiterhin bietet das Programm eine automatische Generierung optimaler Testparameterkombinationen, zu denen der Kunde das von ihm erwartete Ergebnis eingeben kann.

1 Ausgangssituation und Zielstellung

In der Praxis kommt es häufig zu Verzögerungen im Projektablauf sowie erhöhten Entwicklungskosten. Häufige Ursache sind Softwarefehler, die zu lange unerkannt bleiben. Ziel ist es daher, den Auftraggeber, oder allgemeiner den Wissensträger, so früh wie möglich, nicht erst bei der Durchführung von Akzeptanztests, in den Qualitätssicherungsprozess einzubinden, da er die Funktionsfähigkeit eines Softwaremoduls in vielen Fällen besser einschätzen kann als der Auftragnehmer. Bereits die Modultests sollen unter seiner Beteiligung durchgeführt und variiert, sowie Testparameterkombinationen generiert werden. Dies kann zu einer höheren Testabdeckung sowie früheren Erkennung von Softwarefehlern führen und trägt damit zum erfolgreichen Abschluss eines Softwareprojektes bei. Da die Entwicklung von Modultests Programmierkenntnisse erfordert, die bei einem Kunden nicht vorausgesetzt werden können, wird ein grafisches Programm zur Vermittlung zwischen Anwender und Testframework benötigt.

2 Lösungsansatz und Funktionsweise von jCUT

Die an der Fachhochschule Jena entwickelte Anwendung jCUT ermöglicht einem Kunden die Entwicklung und Durchführung von Modultests. Er wird in die Lage versetzt, von Softwareentwicklern implementierte JUnit-Tests aufzurufen und darin hinterlegte Testparameter zu variieren. Diese Parameter müssen innerhalb der entsprechenden Testklasse als statische Attribute hinterlegt sein. Der Zugriff auf die Tests erfolgt mittels Reflection und wird durch die Anwendung gekapselt, sodass der Nutzer zu keinem Zeitpunkt mit Quellcode in Berührung kommt. Ihm wird lediglich eine Archivdatei mit verschiedenen Testinformationen zur Verfügung gestellt, welche in jCUT geöffnet werden kann. Anschließend können die Parameter der enthaltenen Modultests modifiziert werden. Testparameterkombinationen inklusive erwartetem Ergebnis kann der Kunde individuell eingeben oder aber automatisiert durch einen intelligenten Algorithmus generieren lassen. Nach Ausführung eines Tests werden detaillierte Informationen über die Testergebnisse angezeigt. Diese können gemeinsam mit Informationen zur Parameterbelegung in einer Datenbank gespeichert werden, sodass sich die Testläufe zu jedem späteren Zeitpunkt reproduzieren lassen. Darüber hinaus können die Testergebnisse per E-Mail-Report an die Entwickler versendet werden, sodass ein kontinuierlicher Verbesserungsprozess entsteht (siehe Abbildung 1).

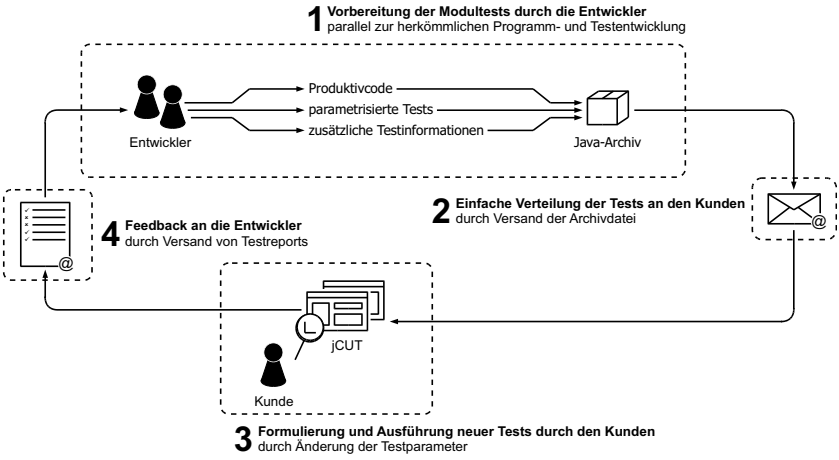


Abbildung 1: kontinuierliche Verbesserung der Entwicklung durch Einbeziehung des Kunden

3 Ausblick

Die Anwendung jCUT ermöglicht bisher nur das Variieren von Parametern, deren Datentyp ein Grunddatentyp oder String ist. Ein nächster Schritt ist die Ausweitung auf allgemeinere Datentypen. Weiterhin ist geplant, den Algorithmus zur Testparametergenerierung zu optimieren. Um die anstehenden Aufgaben priorisieren, auf Praxisrelevanz prüfen und dann durchführen zu können, ist die Zusammenarbeit mit der industriellen Praxis zum Erhalt weiterer Erfahrungswerte notwendig und geplant.

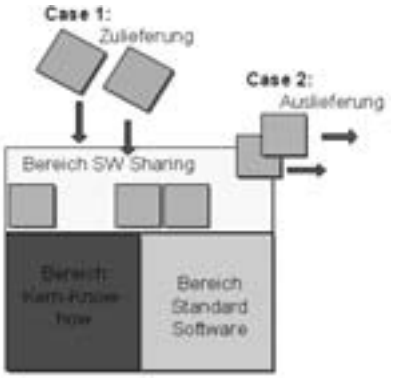
SW-Architektur, SW-Sharing & Standardisierung

Automotive Domäne aus Sicht Tier1

Birgit Boss

DGS-EC/ESB
Robert Bosch GmbH
Postfach 30 02 40
70442 Stuttgart
Birgit.Boss@de.bosch.com

Abstract: Die Zusammenarbeitsmodelle in der Automotive Software Domäne werden immer komplexer und flexibler. Die Auftraggeber (hier ein OEM) von Steuergeräten entwickeln immer mehr Software-Teile selbst und stellen sie dem Zulieferer (hier ein Tier1) zur Integration zur Verfügung. Unter bestimmten Voraussetzungen geht es aber auch in die umgekehrte Richtung: Software, die vom Tier1 entwickelt wurde, wird an den OEM lizenziert und an diesen ausgeliefert. Dieser lässt diese Software dann von einem weiteren Zulieferer von Steuergeräten und damit Mitbewerber von BOSCH integrieren. Solche Zusammenarbeitsmodelle – zusammengefasst unter dem Stichwort „Software Sharing“ oder SW-Sharing - stellen u.a. auch die Software-Architekten eines Steuergeräts vor Herausforderungen. In dieser Präsentation wird gezeigt, wie in der Praxis aus Sicht des Tier1 verfahren wird und wie die Erfahrungen mit dem eingeschlagenen Weg sind. Der Schwerpunkt liegt dabei auf dem Use-Case „Bereitstellung von Software-Teilen“ (Zulieferung durch OEM), weil dieser Fall größere Auswirkungen auf die SW-Architektur hat. Dabei wird insbesondere die Bedeutung von Standardisierung, z.B im Kontext der internationalen Automotive Standardisierungs-Initiative AUTOSAR, für SW-Sharing herausgearbeitet.



Literaturverzeichnis

[AI] AUTOSAR XML Specification of Application Interfaces. AUTOSAR_MOD_AISpecification. Ab R4.0.3. Siehe [AUTO].

[AIMC] Unique Names for Documentation, Measurement and Calibration: Modeling and Naming Aspects including Automatic Generation. TR_AIMeasurementCalibrationDiagnostics. Ab R4.0.3. See [AUTO].

[AUTO] AUTOSAR. http://www.autosar.org, 2012-01-03.

[Bo07] Boss, B.. Zulieferung und Integration in Automotive Produktlinien: Konfiguration von Prozessreihenfolgen innerhalb Tasks. In Proceedings of Software Engineering'2007. pp.51~54. LNI 105 GI 2007.

[Co09] Cordes, J.; Mössinger, J.; Grote, W.; Lapp, A. Autosar Standardised Application Interfaces. Support for Efficient Software Sharing. ATZautotechnology 02/2009 Volume 9.

[Gr10] Grote, W. Business Models for Software in the Automotive Industry. How to Retain Innovation? Software im Automobil, Talk, 2010-05-04

[Je10] Jehle, M.; Winkler, G. Offene Architektur für das Antriebsstrang.Management. ATZ elektronik 06/2010 5.

[St04] Steger, M; Tischer, C.; Boss, B; Müller, A.; Pertler, O.; Stolz, W.; Ferber, S. Introducing PLA at Bosch Gasoline Systems: Experiences and Practices. Software Product Line Conference SPLC, Boston, 2004.

[STD] AUTOSAR Standardization Template. AUTOSAR_TPS_StandardizationTemplate. Ab R4.0.3. Siehe [AUTO].

[Ti03] Tischer, C.; Hammel, C.; Weichel, B.; Ferber, S. Offene Software Systeme basierend auf der EDC/ME(D)17 Architektur. VDI, Baden Baden, 2003.

[Ti11] Tischer, C.; Müller, A.; Mandl, T.; Krause, R. Experiences from a Large Scale Software Product Line Merger in the Automotive Domain, München, Software Product Line Conference (SPLC), 2011.

Softwarearchitektur für Geschäftsanwendungen auf Basis von OSGi

Dehla Sokenou
GEBIT Solutions, Koenigsallee 75b, D-14193 Berlin
dehla.sokenou[at]gebit.de

1 Motivation

Werden wiederholt kundenspezifische Lösungen für fachlich ähnliche Anforderungen entwickelt, bieten domänenspezifische Plattformen große Produktivitäts- und Qualitätsvorteile. Eine tragfähige Komponentenarchitektur muss die gemeinsame Fachlichkeit im Plattform-Layer kapseln und Individualisierungen flexibel und wartungsfreundlich koppeln.

Ein typisches Beispiel für dieses Problem sind Kassensysteme (POS-Systeme). An diese werden folgende Anforderungen gestellt: Aus Softwareherstellersicht spielt Wiederverwendung und Wartbarkeit eine große Rolle. Aus Kundensicht sind in der Regel eine Reihe individueller Lösungen zu realisieren: Implementierung kundenspezifischer Workflows, variable Anpassung an verschiedene Umgebungen, z.B. wenn unterschiedliche Hardware eingesetzt wird, insbesondere auch die Integration desselben Treibers in unterschiedlichen Versionen für unterschiedliche Geräte desselben Herstellers, Einhaltung gesetzlicher Vorgaben und Länderspezifika, z.B. Fiskalisierung.

Es entsteht also ein Spannungsfeld zwischen der Standardisierung einerseits und der Individualentwicklung andererseits. In der bestehenden POS-Plattform wurden zwar auch bisher Technologien eingesetzt, um diese Anforderungen zu erfüllen, es handelte sich jedoch nicht um einen einheitlichen Ansatz, was zunehmend zu einer Verschlechterung der Wartbarkeit führte und die Weiterentwicklung erschwerte.

2 Lösungsansatz und Vorgehen

Um diese Anforderungen möglichst gut zu erfüllen, wurden verschiedene Ansätze für Java evaluiert, u.a. Context and Dependency Injection (CDI) der Java EE Plattform, Dependency Injection Frameworks wie Spring und die Komponentenplattform OSGi.

Als einzige Alternative bietet OSGi die Möglichkeit, Komponenten nicht nur statisch zu erzeugen und einzubinden, sondern zudem dynamisch zu aktivieren und deaktivieren und Komponenten erst zur Laufzeit einzubinden. Da diese Fähigkeit für die bestehende POS-Plattform unabdingbar war, wurde OSGi als Komponentenplattform gewählt.

Im Wesentlichen erfolgte die Umstellung der POS-Plattform in drei Schritten. Zunächst wurden die bestehenden POS-Komponenten in OSGi-Bundles umgewandelt. Nach diesem Schritt hatte man zunächst wieder ein lauffähiges System, was allerdings noch nicht die Vorteile von OSGi im vollem Umfang nutzen konnte, da viele der so erstellten OSGi-Bundles noch zu viele Abhängigkeiten hatten oder nicht optimal geschnitten waren, um in allen Fällen von der Fähigkeit der dynamischen Aktivierung zu profitieren. Zudem war noch Code vorhanden, der zuvor dazu diente, die richtigen Services anzusteuern bzw. die richtigen Komponenten zu laden. Die Anpassung und Bereinigung dieser Codestellen wurde in einem zweiten Schritt durchgeführt. Im dritten Schritt wurde der bestehende Code analysiert und, sofern notwendig oder sinnvoll, wurden einzelne POS-Komponenten auf mehrere OSGi-Bundles verteilt oder anders geschnitten.

3 Erfahrungen und Ergebnisse

Die POS-Plattform ist bereits bei mehreren Kunden im Einsatz. Allerdings basieren erst neuere Versionen auf OSGi. Trotz allem konnten während der aktuellen Entwicklung und dem Einsatz beim Kunden bereits einige Ergebnisse der Umstellung beobachtet werden.

Im Zuge der Umstellung wurden insbesondere folgende Aspekte betrachtet und bewertet:

Entwicklung: Komponenten werden wie bisher unabhängig voneinander entwickelt. Aufwand entsteht bei der Konfiguration des kunden- und länderspezifischen Systems, allerdings ist dieser Aufwand geringer als zuvor, da viele Arbeiten, z. B. das Auflösen referenzierter Services, bereits durch die OSGi-Plattform übernommen werden.

Debugging, Fehleranalyse: OSGi bietet zwar eine OSGi-Konsole, die zur Analyse von z.B. fehlerhaft nicht aktivierten Komponenten verwendet werden kann, allerdings ist diese Analyse meist sehr mühsam und im Fall, dass die OSGi-Plattform aufgrund fehlerhafter Konfiguration gar nicht erst hochgefahren wird, deutlich erschwert, da dann auch die OSGi-Konsole nicht zur Verfügung steht.

Performance: die dynamische Aktivierung von Komponenten hat keinen Einfluss auf die Einhaltung der geforderten Performanceanforderung, allerdings war ein erhöhter Optimierungsbedarf im Vergleich zu vorher festzustellen, um dieses Ziel zu erreichen.

Wartbarkeit: durch viele kleine, lose gekoppelter Komponenten, die jeweils einen Aspekt kapseln, ist die POS-Plattform deutlich einfacher zu erweitern und zu pflegen.

Testbarkeit: hier mussten einige Anpassungen an den eingesetzten Testframeworks vorgenommen werden, um diese OSGi-fähig zu machen. Der Testaufwand ist nicht höher, denn auch zuvor mussten alle Konstellationen getestet werden.

Zusammenfassend lässt sich aussagen, dass der Einsatz von OSGi in Geschäftsanwendungen lohnt, die eine dynamische Konfiguration des Systems zur Laufzeit verlangen. Andere Alternativen bieten zwar statische Konfiguration zur Compile- oder Startup-Zeit an; erfordert das zu implementierende System eine dynamische Konfiguration, gibt es zum Einsatz von OSGi im Augenblick keine Alternative.

TechnoVision: Neue Technologien gezielt für den Unternehmenserfolg nutzen

Dr. Jan-Peter Richter, Dr. Marion Kremer

Capgemini Deutschland GmbH
Lübecker Straße 128
22087 Hamburg
jan-peter.richter@capgemini.com
marion.kremer@capgemini.com

Abstract: IT-Technologie wird heute nicht mehr nur genutzt, um bekannte Geschäftsmodelle zu beschleunigen. Sie steht im Zentrum der Fortentwicklung des Geschäfts selbst. Folglich ist der weit überwiegende Teil der fachlichen Treiber (Businessdriver) großer Unternehmen heute direkt mit neuen IT-Technologien verknüpft – sei es, dass sie nur durch Einsatz von Technologie bewältigt werden können oder dass sie überhaupt erst durch die neuen Technologien entstehen. Capgeminis TechnoVision [Cap11] bietet eine Methodik, auf der Basis der Analyse individueller Businesstreiber eines Unternehmens eine strategische Auswahl von Technologien zu treffen und daraus ein Vorhabenportfolio zu erstellen.

1. TechnoVision: Agora, Businesstreiber und Technologiecluster

Herzstück der TechnoVision ist der Gedanke, dass Business und IT über neue Anforderungen des Business und neue Möglichkeiten auf Seiten der IT ins Gespräch kommen und ein gemeinsames Bild entwickeln müssen. Dazu dient die *Agora* (im Sinnes des antiken Marktplatzes) auf der sich beide treffen. Seinen finalen Ausdruck findet dieser Diskurs in einer Matrix, in der individuelle Businesstreiber in Beziehung zu neuen Technologien gestellt werden. Die TechnoVision bietet dazu eine Analyse branchenspezifischer, allgemeiner Businesstreiber als Ausgangspunkt sowie einen detaillierten Katalog aus 21 Technologiebausteinen, strukturiert in sieben Clustern.

Das **You Experience** Cluster beschreibt z.B. eine neue Generation von Benutzerschnittstellen und Geräten, die den Endanwendern ein Höchstmaß an Freiheit und Individualität erlauben. Die zugrundeliegenden Bausteine sind *Rich Internet Applications* von AJAX bis zu virtuellen 3D-Welten, die *iPodifcation*, d.h. die Bereitstellung aller Funktionen auf tragbaren, weit verbreiteten Endbenutzergeräten sowie *Human Mashup Interaction*, die Komposition von individuellen Mensch-Maschine-Schnittstellen mittels Portalen am Arbeitslatz bis zu Endkunden-eigenen Webseiten zur Dienstnutzung wie bei iGoogle.

Die weiteren Cluster sind: **We Collaborate** – lebende und kreative Kunden- und Partnerbeziehungen durch Social Software, **Process-on-the-Fly** – die adaptive Gestaltung von Geschäftsprozessen, **Triving on Data** – effiziente und innovative Nutzung großer Datenmengen, **Sector-as-a-Service** – Cloud-Technologien zur Auslagerungen ganzer Geschäftsfunktionen, **Invisible Infostructure** – IT-Basisdienste „aus der Steckdose“, sowie das „virtuelle“ Cluster **LiberArchitecture:** *Offene Standards*, *Serviceorientierung* und die *Cloud-Basistechnologien* zur Unterstützung der anderen Technologien und zur Befreiung der Organisationen von alten IT-Zwängen.

2. Workshop und Business Technology Agenda

Kern der Vorbereitung eines TechnoVision Workshops ist die Identifikation der Businesstreiber. Dies sind bedeutende Herausforderungen oder Vorhaben, die darauf zielen, eine Strategie umzusetzen, einen Missstand zu beheben, eine Chance zu nutzen, einem negativen Einfluss zu begegnen, etc. Ein gut abgegrenzter strategischer Businesstreiber ist z.B. „die Fähigkeit, 1 Million neue Kunden pro Monat zu bedienen“. Weitere Kategorien für Businesstreiber sind gesellschaftliche Trends, die Optimierung interner Prozesse, die Verbesserung der Außenwirkung oder regulatorische Auflagen.

Der TechnoVision Workshop beginnt mit einer ausführlichen Diskussion der Technologiebausteine und deren Clusterung. Durch Beispiele konkreter geschäftlicher Nutzung der Technologien wird die Übertragung, d.h. das Erkennen ihrer Bedeutung für die betrachtete Organisation erst möglich. Anschließend wird gemeinsam die Matrix aller Schnittpunkte gefüllt. Dabei werden die Cluster bewertet, ob sie ein „Muss“ für die Bewältigung des Businesstreibers darstellen, hilfreich oder selbst Ursache sind. Bei der Diskussion zu dieser Bewertung kommt es darauf an, eine klare Vision davon zu entwickeln, *welche* Technologie(n) des Clusters *wie* eingesetzt werden können. Abschließend werden die ausgefüllten Zellen auf ihre Umsetzbarkeit bewertet. Dabei hilft die stark von dynamischen Aspekten geprägte Sicht, in der Systeme danach betrachtet werden, wie sie sich im Zeitverlauf ändern. Dabei dienen Fahrzeuge als Metaphern um die unterschiedliche Flexibilität der einzelnen Anwendungen und Anwendungsteile zu unterscheiden: Der Scooter, das Auto, der Bus, der Zug und der Hub. Anhand dieser Metaphern werden die Vorhaben diskutiert und klassifiziert, um Budgetvolumen, Vorgehen und Architektur sinnvoll wählen zu können.

Mit der ausgefüllten TechnoVision-Matrix erhält eine Organisation konkrete Unterstützung bei der Aufstellung eines Business Technologie Projektportfolios. Die Diskussion um Businesstreiber und neue Technologien als treibende Kraft hinter neuen Geschäftsmodellen fördert das Verständnis und die Zusammenarbeit zwischen Business und IT.

Literaturverzeichnis

[Cap11] http://www.capgemini.com/insights-and-resources/by-publication/building-the-business-technology-agora-with-capgeminis-technovision/

Anforderungen auf Konsistenz überprüft - Formalisierung hilft

Dr. Marc Segelken
MathWorks GmbH, Germany
EMail: marc.segelken@mathworks.de

Abstract: Im Requirements Engineering sind Fragestellungen bzgl. der Eindeutigkeit, Widerspruchsfreiheit und Vollständigkeit Herausforderungen, die in herkömmlichen, manuell textbasierten Vorgehensweisen nur schwer zu beherrschen sind. Dieser Beitrag beschreibt für reaktive Systeme und Anforderungen, wie sie typischerweise in eingebetteten Systemen vorkommen, den Anwendungsfall der automatischen Modell-unabhängigen Überprüfung von Mengen von Requirements hinsichtlich Widerspruchsfreiheit und Vollständigkeit durch Verwendung von in graphischer Darstellung repräsentierter Anforderungen durch formale Methoden.

Das Requirements Engineering in der Spezifikationsphase ist ein wesentlicher Bestandteil eines Software Entwicklungsprozesses, der zumeist textuell oder semi-formal umgesetzt wird, unterstützt durch Requirements-Management Werkzeuge. Auf dieser Basis werden die erforderlichen wiederholten manuellen Reviews durchgeführt, welche Korrektheit, Eindeutigkeit, Widerspruchsfreiheit und Vollständigkeit der Anforderungen gewährleisten sollen. Gerade bei umfangreichen Anforderungskatalogen ist ein solcher Reviewprozess trotz signifikantem Aufwands jedoch überfordert, die genannten Zielstellungen umfassend zu erfüllen.

Abhilfe können hier automatisierbare Analyseverfahren auf Basis vollständig formalisierter Requirements schaffen und manuelle Reviews somit ersetzen. Die notwenigerweise mathematisch präzise Darstellung erzwingt Eindeutigkeit der Anforderungen und erlaubt die klassischen Verifikationsanwendungen zur Testüberwachung und die formale Verifikation selbiger, so dass sich der Mehraufwand für die Formalisierung der Anforderungen bereits hierfür schnell amortisiert. Insbesondere im Umfeld sicherheitskritischer eingebetteter Systeme, welche hier betrachtet werden sollen, sind diese Verfahren zur Absicherung daher weiter auf dem Vormarsch.

Noch weitgehend unbeachtet sind jedoch weitere Anwendungsfälle formalisierter Anforderungen zur automatischen Analyse von Widerspruchsfreiheit von Anforderungen untereinander, sowie der Überprüfung der Vollständigkeit der Spezifikation bzgl. einer praxistauglichen Definition der Vollständigkeit. Diese Analysen sind bereits nach Fertigstellung der Anforderungen möglich und völlig unabhängig von einer Implementierung durchführbar. Hierdurch können neben dem gezielten Aufdecken von Inkonsistenzen und Lücken in der Spezifikation unter dem Einsatz formaler Methoden sogar Konsistenz und Vollständigkeit bewiesen und damit im Entwicklungsprozess diese frühe wichtige Phase abgeschlossen werden ohne in späteren Iterationen teure Korrekturen und Ergänzungen vornehmen zu müssen.

Im Bereich reaktiver Systeme aus dem Embedded Bereich werden in der industriellen Praxis derzeit LTL-basierte Model Checker eingesetzt, bei denen das Requirement letztlich durch einen Observer-Automat repräsentiert ist. Mögliche Umsetzungen und Anwendung entsprechender Beweisdurchführungen zur Konsistenz und Vollständigkeit sind für diesen Bereich noch weitgehend unbekannt. In diesem Beitrag wird gezeigt, wie durch ein graphisches Entwicklungswerkzeug wie Simulink Beweise formalisiert und für automatische Konsistenz- und Vollständigkeitsanalysen genutzt werden können.

Die Formalisierung der Requirements wird graphisch unter Zuhilfenahme von Operatoren zur Spezifikation von temporalen Randbedingungen typischerweise in Form von logischen Wenn-Dann-Aussagen umgesetzt. Für jedes Requirement ensteht so ein einfaches Simulink-Subsystem, welches die zeitlichen Verläufe relevanter Größen der jeweiligen Anforderung beobachtet und bei Verletzung der Anforderung einen Fehler anzeigt. Genau diese werden bereits für anforderungsbasierte Testfallgenerierung und insbesondere für das Beweisen oder Widerlegen der Einhaltung einer Anforderung durch Model Checker verwendet.

Für Konsistenz- und Vollständigkeitsbeweise müssen die formalisierte Anforderungen logisch gruppiert und verbunden werden. Für einen Konsistenz-Beweis werden die Anforderungen hierzu für jedes (Ausgabe-)Signal zusammengefasst, welches sie betreffen, d.h. die zugehörigen Requirements fordern oder verbieten zu bestimmten Zeitpunkten bestimmte Wertverläufe. Durch geeignete logische Kombination wird hieraus ein neues Modell aufgebaut, an dem mögliche Widersprüche durch einfache Erreichbarkeitsanalyse mit Hilfe eines Model-Checkers durchgeführt werden können. Ergebnis hiervon ist die gezielte Ermittlung von Situationen, in denen sich mindestens zwei Requirements bzgl. des von dem später zu erstellendem Systems widersprechen, d.h. die Software-Komponente lässt sich ohne Änderungen wie z.B. Priorisierungen von Anforderungen nicht realisieren. Eine solche Situation kann gezielt aufgedeckt bzw., falls diese Situation nicht eintreten kann, ausgeschlossen werden.

Neben der Widerspruchsfreiheit stellt sich die Frage bzgl. der Vollständigkeit der Anforderungen. Hier ist zunächst eine sinnvolle praxisrelevante Definition von Vollständigkeit zu treffen, da Anforderungen typischerweise einen Spielraum im Wert- und Zeitbereich beinhalten. Hernach kann darauf aufbauend erneut das System hinsichtlich Verletzungen einer solchen Definition durch eine einfache logische Kombination der formalisierten Anforderungen auf Vollständigkeit untersucht werden, wobei ebenfalls automatische Erreichbarkeitsanalysen eingesetzt werden. Als Ergebnis der Analyse werden somit potentielle Unterspezifikationen aufgedeckt, welche durch Präzisierung oder Ergänzung der Requirements auszuschließen wären.

Der Vorteil dieser Vorgehensweise liegt in der frühen Anwendbarkeit bei der Entwicklung eines Software-Systems oder einer Software-Komponente: Lediglich die Schnittstelle muss bekannt sein – es wird keine Implementierung der Komponente benötigt. Beide Analysen lassen sich allein auf Basis der vorliegenden Anforderungen durchführen. Somit kann Widerspruchsfreiheit und Vollständigkeit von Anforderungen unabhängig von einem noch zu entwickelnden Modell gewährleistet werden. Zeitaufwändige Iterationen mit späten Korrekturen der Anforderungen in der Implementierungs- oder Testphase werden somit vermieden.

Software-Architektur und Open-Source-Lizenzrecht in Einklang bringen

Ralf S. Engelschall

msg Applied Technology Research
msg systems ag
Robert-Bürkle-Str. 1
85737 Ismaning
ralf.engelschall@msg-systems.com

Abstract: Beim Einsatz von Open-Source Software in kommerziellen Produkten müssen Software Engineering und Rechtsabteilung sich synchronisieren. Indem sowohl die Verbauung der Software-Komponenten als auch die Aussagen der beteiligten Open-Source Lizenzen strukturiert erfasst und Werkzeug-unterstützt ausgewertet werden, können lizenzrechtliche Konflikte zeitnah und präzise erkannt werden.

1. Herausforderung und Ziel

In der kommerziellen Software-Enwicklung ist es heute extrem wichtig, Engineering und Rechtsabteilung an einen Tisch zu bringen und ein *gemeinsames* Verständnis beim Einsatz von Open-Source Software-Komponenten zu haben. Dies ist eine große Herausforderung, da Architekten üblicherweise eine streng funktionelle und strukturelle Sicht und wenig Affinität zu Lizenztexten haben, während Rechtsanwälte zwar das lizenzrechtliche Verständnis mitbringen, sich aber sehr schwer tun, die konkrete technische Verbauung einer Software-Komponente rechtlich zu interpretieren.

2. Erkenntnisse und Lösungsansatz

1. Erkenntnis: Nutzungsarten sind entscheidend. Nicht einzelne Open-Source Lizenzen (wie z.B. die GPL) stellen ein Problem dar, sondern die konkrete Nutzungsart der Open-Source lizensierten Software-Komponente und die Aussage, die eine Lizenz bezüglich dieser Nutzungsart trifft ist entscheidend. Ein GPL-lizensiertes Werkzeug während der Software-Entwicklung zu nutzen ist rechtlich harmlos, eine GPL-lizensierte Bibliothek in ein kommerziell lizensiertes Produkt einzubauen ist dagegen fast unmöglich.

2. Prozess-Optimierung: Strukturelle Erfassung und Werkzeug-Unterstützung. Da man oft Duzende von Produkten entwickelt und in jedem eine Vielzahl an Fremd-Komponenten mit unterschiedlichen Nutzungsarten verbaut werden, benötigt man

eine strukturierte Erfassung der zahlreichen Informationen und eine Werkzeug-unterstützte Auswertung, um Konflikte zeitnah und präzise zu erkennen.

3. Vorgehen

1. Meta-Modellierung: Um eine strukturelle Erfassung und eine Werkzeug-unterstützte Konflikt-Auswertung zu erreichen, wird von einem Open-Source Experten, zusammen mit den Software-Architekten und einem externen Fachanwalt, ein Meta-Modell definiert, in dem sowohl die Lizenz-Seite als auch die Architektur-Seite modelliert und somit zusammengebracht werden.

2. Werkzeug-Unterstützung: Basierend auf dem Meta-Modell wird vom Open-Source Experten, zusammen mit dem internen und externen Fachanwalt, ein Werkzeug entwickelt, das einerseits die strukturelle Erfassung als auch die automatische Auswertung erlaubt.

3. Lizenz-Modellierung: Mit Hilfe des Werkzeugs werden vom externen Fachanwalt vorab die entscheidenden Aussagen in den Prosa-Texten aller relevanten Open-Source Lizenzen modelliert. Die Modellierung basiert primär auf Matrizen aus Nutzungsarten und Bedingungen/Bürden.

4. Produkt-Modellierung: Mit Hilfe des Werkzeugs werden von den Software-Architekten alle Produkte modelliert, indem pro Produkt alle verbauten Komponenten und ihre konkrete Nutzungsart erfasst werden. Hierzu verwendet man sowohl einen Bottom-Up Ansatz (semi-automatisiertes Zerlegen einer Anwendung in ihre *transitiv* verbauten Komponenten) als auch einen Top-Down Ansatz (Extrahieren der *direkten* Abhängigkeiten aus den Konfigurationen des Build-Prozesses).

5. Auswertungen: Mit Hilfe des Werkzeugs und dessen erfassten Informationen können nun sowohl die Software-Architekten (proaktiv) als auch Line- und Projekt-Manager (reaktiv) jederzeit sehen, ob ein lizenzrechtlicher Konflikt in der Verbauung von Open-Source Software-Komponenten existiert.

Erfahrungen

Die ausschließlich Prosa-basierten Lizenzen von Open-Source Komponenten strukturiert modellieren und erfassen zu können ist der Knackpunkt des Ansatzes. Erst dadurch können die beiden Seiten Software-Architektur und Lizenz-Recht sinnvoll miteinander verbunden werden.

Architekturmanagement für eine föderale Produktlinienarchitektur

Simon Giesecke, Niels Streekmann

BTC Business Technology Consulting AG
simon.giesecke@btc-ag.com
niels.streekmann@btc-ag.com

Abstract: Dieser Beitrag beschreibt die ersten Schritte auf dem Weg zu einer föderalen Produktlinienarchitektur im Rahmen der Produktentwicklung bei der BTC AG. Ein wichtiges Ziel ist dabei die Verbesserung der Interoperabilität bisher unabhängig voneinander entwickelter Produkte. Maßnahmen zur Erreichung dieses Ziel sind die Bearbeitung produktübergreifender Architekturthemen und die Durchführung von Architekturreviews.

1. Ausgangssituation

Im Rahmen der Entwicklung von Produkten im Bereich des Energie-Prozess-Managements (EPM) bei der BTC AG wurden verschiedene Entwicklungsvorhaben zu einer Prä-Produktlinie zusammengeführt. Diese Vorhaben wurden vorher in unterschiedlichen Kontexten entwickelt und sind daher sehr heterogen. Um diese Heterogenität zu reduzieren, wurde ein gemeinsamer Architekturstil definiert, der die Besonderheiten der Entwicklung von Produkten im Bereich EPM adressiert. Der Architekturstil definiert grundlegende Architekturstandpunkte, die insbesondere Wartbarkeit, Austauschbarkeit und Wiederverwendbarkeit fokussieren. Außerdem wird eine Basisplattform entwickelt, die Standardschnittstellen und wiederverwendbare Komponenten zur Verfügung stellt.

2. Ziele

Ein wichtiges Ziel des Architekturmanagements im Rahmen der Produktentwicklung ist die Verbesserung der technischen und fachlichen „Interoperabilität“ in Bezug auf die Architektur der unterschiedlichen Produkte. Der Fokus liegt dabei auf der Identifikation und Bearbeitung übergreifender Architekturthemen, wie z.B. homogener Fehlerbehandlung, verteilter Datenhaltung bei kritischen Performanceanforderungen und der Definition interner Standandschnittstellen für nicht-funktionale Variationspunkte. Diese fließen in den gemeinsamen Architekturstil ein und stellen somit den ersten Schritt auf dem Weg zu einer föderalen Produktlinienarchitektur dar, die alle Produkte für das Energie-Prozess-Management umfasst. Im Unterschied zu einer regulären Produktlinie soll es jedoch kein übergreifendes Variantenmodell geben, da das Produktspektrum hierfür zu breit ist. Aufbauend auf Basisarchitekturregeln wie der Komponentenorientierung soll es gemeinsame Architekturregeln insbesondere zum Einsatz der Basisplattform geben.

3. Lösungsansatz und Vorgehen

Um das Ziel einer verbesserten Interoperabilität zu erreichen, werden zwei Maßnahmen durchgeführt, die eng ineinander greifen. Dieses sind die Bearbeitung übergreifender Architekturthemen und die regelmäßige Durchführung von Architekturreviews.

Zur Bearbeitung produktübergreifender Architekturthemen wurde ein Arbeitskreis eingerichtet. Dessen Teilnehmer sind die Architekten der einzelnen Produkte sowie der Produktmanager und die Architekten der Basisplattform. Die Aufgaben des Arbeitskreises umfassen die Identifikation übergreifender Architekturthemen und dazugehöriger bestehender Best Practices sowie die Bewertung ihrer Relevanz für die Interoperabilität zwischen den einzelnen Produkten sowie zwischen den Produkten und der Basisplattform. Darauf aufbauend werden Arbeitspakete zur Erarbeitung gemeinsam genutzter Konzepte definiert. Zur Bearbeitung der Arbeitspakete werden Arbeitsgruppen aus Teilnehmern des Arbeitskreises und weiteren Entwicklern gebildet. Deren Ergebnisse bilden die Basis für vom Arbeitskreis zu beschließende Vorgaben und Richtlinien, die in den gemeinsamen Architekturstil einfließen. Durch diese werden zudem die einzelnen Produkte von der Definition und Dokumentation eigener Richtlinien befreit.

In regelmäßigen Abständen werden Architekturreviews aller Produkte und Basisplattformbestandteile durchgeführt. Diese dienen dazu einen einheitlichen Überblick über die Ausgangslage der Produktentwicklung zu bekommen und übergreifende Architekturthemen zu identifizieren. Für bereits durch den Arbeitskreis beschlossene Architekturrichtlinien dienen sie zudem zur Prüfung der Umsetzung und Einhaltung.

4. Ergebnisse

Ein Beispiel für ein übergreifendes Architekturthema ist die Fehlerbehandlung. Diese sollte innerhalb eines Produktes und im Zusammenspiel der Produkte mit der Basisplattform einheitlich sein. Die Abwesenheit eines einheitlichen Fehlerbehandlungskonzeptes erschwert vor allem die Wiederverwendung dieser Komponenten.

Bisherige Architekturreviews haben gezeigt, dass in unterschiedlichen Entwicklungsvorhaben bisher verschiedene Konzepte verwendet wurden. Das führt nicht zwingend zu fehlerhafter Software, aber zu höherem Entwicklungsaufwand, insbesondere bei der Verwendung von Basisplattformkomponenten. Hier können übergreifende Architekturrichtlinien zu einer Effizienzverbesserung führen.

5. Nächste Schritte

Aktuell wird ein produktübergreifendes Domänenmodell für den Bereich EPM definiert. Den Domänen werden Schnittstellen und Komponenten der Basisplattform zugeordnet. In den bisherigen Architekturreviews wurde eine Reihe von Themen identifiziert, zu denen nun Richtlinien erstellt werden, die den gemeinsamen Architekturstil ergänzen.

Trusted Cloud im Gesundheitswesen mit TRESOR

Torsten Frank

medisite Systemhaus GmbH
Karl-Wiechert-Allee 20
30625 Hannover
torsten.frank@medisite.de

Abstract:. TRESOR hat den Aufbau eines sicheren und datenschutzkonformen Cloud-Ecosystems zum Ziel, welches exemplarisch für den Anwendungsbereich der Patientenversorgung im Gesundheitswesen umgesetzt wird.

1. Über Trusted Cloud

Das Forschungsprojekt TRESOR der Konsortialpartner medisite Systemhaus GmbH, T-Systems International GmbH, Ubiry GmbH, TU Berlin - Service-centric Networking, TU Berlin - Wirtschaft und Management Fachgebiet Informations- und Kommunikationsmanagement, Deutsches Herzzentrum Berlin und das Paulinenkrankenhaus Berlin ist einer der Preisträger des BMWi-Technologiewettbewerbs "Sicheres Cloud Computing für Mittelstand und öffentlichen Sektor - Trusted Cloud".

Trusted Cloud ist ein Technologieprogramm des BMWi mit dem Ziel der Entwicklung und Erprobung innovativer, sicherer und rechtskonformer Cloud Computing-Lösungen. Das Programm Trusted Cloud wurde im September 2010 als Technologiewettbewerb des BMWi ausgeschrieben. Insgesamt haben 116 Projekte an der Ausschreibung teilgenommen. In einem mehrstufigen Prozess mit Unterstützung einer unabhängigen Expertenjury wurden 14 zu fördernde Projekte ausgewählt. Auf der CeBIT 2011 sind die Projekte öffentlich vorgestellt worden. An den 14 Forschungsprojekten sind insgesamt 44 Unternehmen und 22 wissenschaftliche Einrichtungen beteiligt. Die Forschungs- und Entwicklungsaktivitäten werden bis Ende 2014 laufen.

2. Sicheres und rechtskonformes Cloud-Ecosystem für standardisierte Dienste

Kernbestandteile dies TRESOR-Ecosystems sind eine neue offene PaaS-Plattform für die Bereitstellung und Nutzung standardisierter cloud-basierter System- und Anwendungsdienste sowie ein Cloud Broker als vertrauenswürdiger Mediator zwischen einem Klienten und den Cloud-Anbietern des Ecosystems. Die PaaS-Plattform zeichnet sich

aus durch die Verwendung etablierter Standards für die Beschreibung von System- und Anwendungsdiensten und die daraus resultierende Vermeidung von Lock-In-Effekten sowie die Möglichkeit der Orchestrierung von Diensten verschiedener PaaS-Anbieter. Der Cloud Broker ist ein vertrauenswürdiger, zentraler Einstiegspunkt, der für den Klienten Cloud-Ressourcen auf den verschiedenen Ebenen (IaaS, PaaS und SaaS) unter Berücksichtigung von gesetzlichen Vorschriften sowie Sicherheits-, Datenschutz- und sonstigen Unternehmensrichtlinien vermittelt, bereitstellt und bündelt. Dies geschieht unter Berücksichtigung etablierter Mechanismen des Rollen- und Rechtemanagements, die um neue innovative Konzepte zur geographischen Eingrenzung der Nutzung von Cloud-Diensten ergänzt werden.

3. Wegweisend für das Cloud-Computing im Gesundheitswesen

Das Vorhaben konzentriert sich exemplarisch auf Gesundheitseinrichtungen wie etwa Krankenhäuser oder Ärzte aber auch mittelständische und Industrieunternehmen aus dem Gesundheitswesen, da diese Zielgruppe besonders viele Charakteristiken für eine „Trusted Cloud“ aufweisen. So fordert dieser Bereich sehr hohe Standards bezüglich Datenschutz und Datensicherheit und hat aufgrund der Vielzahl der beteiligten Akteure hohe Anforderungen an Interoperabilität, Skalierbarkeit und Verfügbarkeit.

Ein Projektansatz wird daher die exemplarische Umsetzung einer solchen medienbruchfreien medizinischen Verlaufsdokumentation sein. Ein weiteres Szenario konzentriert sich auf die Umsetzung eines cloudbasierten, den Behandlungsprozess begleitenden Services zur verlässlichen Prüfung von Arzneimittelinteraktionen auf der Basis aktuellster Informationen. Die Umsetzbarkeit der im Rahmen von TRESOR entwickelten Lösungen des Cloud-Ecosystems wird anhand von Anwendungsszenarien aus dem Bereich der Patientenversorgung demonstriert.

Mit TRESOR wird erstmals eine "Trusted Cloud" - Infrastruktur bereitgestellt werden, die in der Lage ist, alle relevanten gesetzlichen Vorschriften, Sicherheits- und Datenschutzrichtlinien sowie individuelle Richtlinien mittelständischer Unternehmen und des öffentlichen Sektors zu berücksichtigen. Die exemplarische Umsetzung des Cloud Ecosystems für den Anwendungsbereich der Patientenversorgung wird zukunftsweisend für die weitere Entwicklung von Cloud Enabled Softwareservices im Gesundheitswesen sein.

Datenschutz und Usability bei Smartcards:

Card-to-Card-Authentication

Dr. S. Buschner

Competence Center Health 3 - Telematik
adesso AG
Rotherstr. 19
10245 Berlin
stefan.buschner@adesso.de

Abstract: Der Vortrag ist ein Bericht über die Sicherheitstechnologien bei der elektronischen Gesundheitskarte (eGK), die bei der gematik – Gesellschaft für Telematikanwendungen der Gesundheitskarte mbH spezifiziert wird. Die gematik ist ein langjähriger Kunde der adesso AG und zahlreiche Mitarbeiter der adesso AG haben in diesem Projekt mitgearbeitet, sei es in der Spezifikation, beim Bau von Referenzsystemen als auch bei Test und Zulassung der Komponenten für den Regelbetrieb. Das besondere an den Sicherheitstechnologien der Gesundheitskarte ist, dass es gelungen ist, die Sicherheit gegenüber Standardverfahren zu steigern und gleichzeitig die Usability des Systems zu verbessern.

1. Ausgangssituation

Smartcards sind das Mittel der Wahl, wenn es darum geht, Daten „mitnehmbar“ zu haben und gleichzeitig die Daten maximalen Schutz bedürfen. Die Karten in der Größe einer Visitenkarte sind in der Lage jeden illegalen Zugriff auf die Daten zu verhindern. Deswegen werden Smartcards auch als Träger des Schlüsselmaterials für die qualifizierte elektronische Signatur eingesetzt, die höchsten gesetzlichen Ansprüchen genügen muss. Selbst ein direkter Angriff auf die Microstruktur des Chips lässt keinen Zugriff auf die Daten, in diesem Fall das Schlüsselmaterial, zu. Die Authentifikation des Users gegenüber der Smartcard erfolgt dabei i.d.R. per PIN.

2. Card-to-Card-Authentication

Die Daten auf der eGK haben, da es sich um medizinische Daten der Versicherten handelt, besonderen Schutzbedarf. Den verschiedenen Akteuren im Gesundheitswesen soll dabei ein fachgerechter Zugriff auf diese Daten gegeben werden, so dass eine pauschale Freischaltung der eGK per PIN nicht in Frage kommt. Außerdem soll der Versicherte die PIN so selten wie möglich eingeben müssen, um maximale Usability zu gewährleisten. Die Lösung ist die Card-to-Card-Authentication. Dafür bekommen die Leistungserbringer im Gesundheitswesen ebenfalls Smartcards (so genannte HPCs), die

sich gegenüber der eGK des Versicherten per Zertifikat authentifizieren und so die eGK gemäß den Rechten des Leistungserbringers freischaltet.

Literaturverzeichnis

Spezifikationen zur elektronischen Gesundheitskarte (s. www.gematik.de)

Forschungsarbeiten

Developer Belief vs. Reality: The Case of the Commit Size Distribution

Dirk Riehle, Carsten Kolassa, Michel A. Salim

Friedrich-Alexander-University Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen, Germany
dirk@riehle.org, carsten@kolassa.de, michel@sylvestre.me

Abstract: The design of software development tools follows from what the developers of such tools believe is true about software development. A key aspect of such beliefs is the size of code contributions (commits) to a software project. In this paper, we show that what tool developers think is true about the size of code contributions is different by more than an order of magnitude from reality. We present this reality, called the commit size distribution, for a large sample of open source and selected closed source projects. We suggest that these new empirical insights will help improve software development tools by aligning underlying design assumptions closer with reality.

1. Introduction

In this paper, we study what developers believe is true about the size of code contributions to software development projects and how it compares with reality. We find that 1-3 lines of code are the most common commit size while developers predict it would be much higher. The contributions of this paper are:

- The results of a developer survey on commit sizes and an assessment of the beliefs developers hold about them,
- the assessment of the reality of commit sizes in software development for a large sample of open source software projects,
- the assessment of the reality of commit sizes in software development for selected closed source software projects of SAP, a large software vendor,
- the surprising conclusion that software developer beliefs differ significantly from development reality.

Finding that there is a significant difference between reality and what developers believe about code contributions leads us to suggest that software development tools may have suboptimal designs that can be improved with our newly found knowledge. Our work strongly suggests revisiting the design of code-centric development tools.

Our finding may be surprising, depending on which school of psychology you belong to. By now classic research into cognitive biases [6] has shown that humans are poor estimators and various biases get in the way of accurately estimating properties of distributions like the one discussed in this paper [15]. However, recent research also has shown that under defined experimental situations, people can be capable of predicting the mode and percentiles of a distribution well [12] [4]. In the discussion section, we offer some

possible explanations. However, it is not the purpose of this paper to explain this finding but only to bring it into the open so that we can build better software development tools.

The paper is structured as follows. Section 2 defines the necessary terms. Section 3 reports the results of a developer survey to understand what developers believe is true about commit sizes in software development. Section 4 then shows the measurable reality for open source and a few selected closed source projects. Section 5 compares developer belief with reality and shows that beliefs differ from reality by at least an order of magnitude. Section 6 discusses threats to the validity of our analysis. Section 7 discusses related work and Section 8 concludes the paper.

2. Commit Sizes

Software developers contribute code to software projects when going about their programming work. Such code contribution is commonly called a "commit". Any such commit adds, removes, or changes existing source code. A common measure of size for commits is lines of code (LoC), approximating the amount of work spent on the commit. We distinguish source code lines from comment lines from empty lines:

- A *source code line* (SLoC) contains program code,
- a *comment line* (CL) contains only comments, and
- an *empty line* contains only whitespace.

We define LoC = SLoC + CL and thus measure the size of a commit in source code lines + comment lines. Measuring the size of a commit is non-trivial. The primary tool for assessing commit sizes is the "diff" tool, which compares source code files and tells its user which lines have been added and which lines have been removed.

- A *commit* is the sum of all diffs over all affected files where
- a *diff* lists several diff chunks in one file, where
- a *diff chunk* identifies co-located lines added or removed.

Unfortunately, a diff tool cannot identify with certainty that a line was changed; such a changed line is always counted as one line removed and one line added. However, a changed line should count as one line of work, while an added and a removed line of code should count as two lines of work. Enhanced diff algorithms exist, for example [3], that use text analysis to calculate the probability of whether one line added and one line removed equals one line changed or two separate lines, one added, one removed. However, these algorithms can't provide certainty and are computationally expensive.

Prior work determined that for large numbers of commits, the mean of the minimally and maximally possible values is a statistically valid estimate for the diff chunk size [9]. In this work, we calculate commit sizes for more than 8 million commits. Table 1 provides the resulting equations: "a" represents the number of lines added and "r" represents the number of lines removed according to the diff tool. We compute commit sizes by adding up the diff chunk sizes computed using equation (c) of Table 1.

Table 1: Equations used to compute a commit's size

```
(a)  lower_bound(a, r) = max(a, r)                                  // full overlap, highest number of changed lines
(b)  upper_bound(a, r) = a + r                                      // no overlap, no changed lines in diff chunk
(c)  diff_chunk_size(a, r) = (lower_bound(a, r) + upper_bound(a, r)) / 2   // mean of both bounds
```

The *commit size distribution* of some commit population is the distribution of the number of occurrences (y-axis) of all possible commit sizes (x-axis). The *commit size distribution of open source* has all commits in all open source projects as the population, and the *commit size distribution of closed source* has all commits of all closed source projects as the population.

3. Developer Belief

To assess what developers believe is true about commit sizes, we performed a survey in early 2010. We asked about the commit size distribution of all of open source and all of closed source, respectively. We asked what developers thought was

- the "most frequent commit size" (mode),
- the "average commit size" (mean),
- the commit size at the 90th percentile, and
- the shape of the commit size distribution.

We asked these questions separately for all of open source and all of closed source software, and we assessed survey respondent demographics to the extent that it was relevant to evaluating this survey. An underlying assumption is that different projects have similar commit size distributions. We made this assumption explicit to survey respondents and Section 4 validates it. The survey is available on the web [14].

We performed pre-tests to ensure that definitions are clear and will be understood by survey respondents. We reached out to software developers using various channels, mostly mailing lists, but also social media tools like blogs, micro-blogging services, and social networking services. We received 73 valid and complete survey responses. In the threats to validity section we look at the sampling error and show why we believe that this low return rate does not affect the validity of our conclusions.

Figures 1-4 show developer beliefs for mode and 90th percentile of the open source or closed source commit size distributions. The 90th percentile commit size defines the size that respondents believe is larger than 90% of all other commit sizes. We omit the discussion of the mean in this Section, because the survey results are in line with those of the mode and and 90th percentile and don't add anything to the discussion.

Figures 5-6 show what survey respondents believe is the difference between open source and closed source. Between 10%-20% of all respondents believe that there is no difference between open source and closed source as to mode and 90th percentile. Thus, the vast majority believes there is a difference. In general, survey respondents believe that open source has a smaller mode and 90th percentile of commit sizes than closed source.

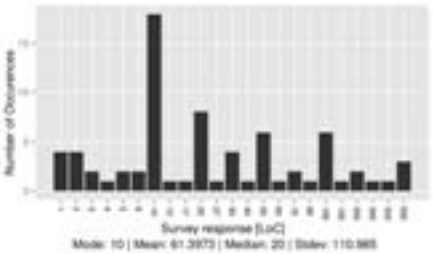


Figure 1: Survey results of developer belief for mode of open source commit size distribution

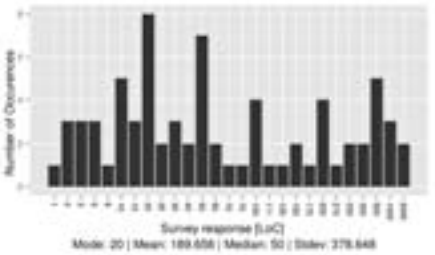


Figure 2: Survey results of developer belief for mode of closed source commit size distribution

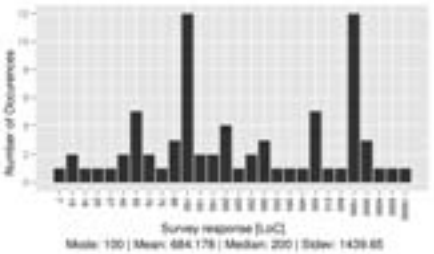


Figure 3: Survey results of developer belief for 90th percentile of open source commit size distribution

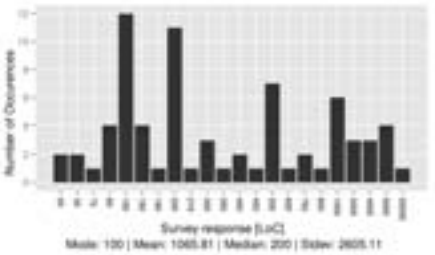


Figure 4: Survey results of developer belief for 90th percentile of closed source commit size distribution

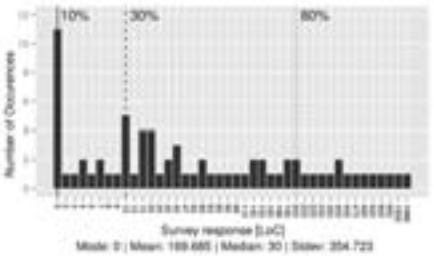


Figure 5: Absolute value of difference in developer belief between mode of open and closed source commit size distribution

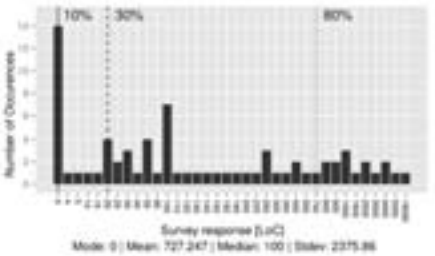


Figure 6: Absolute value of difference in developer belief between 90th percentile of open and closed source commit size distribution

We visualized three different distributions and let survey respondents choose which one they felt was closest to reality. The three alternatives were (a) a normal distribution, (b) a

Table 2: Survey results of developer belief for shape of open source commit size distribution

Normal distribution:	16%
Power law distribution:	59%
Skewed to large commits:	25%

Table 3: Survey results of developer belief for shape of closed source commit size distribution

Normal distribution:	35%
Power law distribution:	33%
Skewed to large commits:	32%

distribution following a power law, and (c) a distribution skewed towards large commits. Tables 2 and 3 show how respondents chose among the alternatives. There was no clear consensus and all three distributions had roughly equal support among respondents.

4. Development Reality

Open source software development is usually public software development. Thus, for open source, we can know the commit size distribution and do not have to rely on beliefs. For closed source software development we have to rely on selected projects as the complete commit size distribution of closed source or even a representative sample is practically impossible to determine.

4.1 The Open Source Distribution

To assess the commit size distribution of open source, we use a database snapshot of the Ohloh.net open source project data website. Our snapshot is dated March 2008. Unlike data sources like SourceForge.net [10] the Ohloh data has no apparent bias towards any particular category of open source project. The only bias we could see is a focus on active projects with engaged user communities, as the Ohloh service requires community participation to have a project listed (self-reporting bias), as well as an English-language bias, given that the Ohloh website is written in English.

Our snapshot contains 11,143 open source projects. In September 2007, Daffara estimated that there are 18,000 active open source projects in the world [5]. (The total number of projects is much larger, but most open source projects are not active and by our activity definition have to be excluded.) Using the same definition of "active project" as Daffara our database snapshot contains 5,117 active open source projects. Thus, we estimate that our database contains about 30% of all open source projects active in March 2008.

The Ohloh database contains the complete commit history of all of its projects to the extent that it is available on the web, going back as far as 1991. Using the definitions from Section 2, we measure the commit size distribution of our open source sample population. Iterating over all 8,705,118 commits in the database, we compute the commit sizes and determine the commit size distribution.

Figure 7 shows the commit size distribution of our open source sample. The distribution is shown as a probability density function (PDF) for visualization and comprehension purposes. Integrating the interval [0, 1] provides the probability of a commit of size 0 or 1 LoC, [0, 2] provides the probability of a commit of size 0, 1, or 2 LoC, etc. The PDF is strictly falling with 1-3 LoC being most frequent.

The PDF presented here is empirical: It does not present an analytically closed model; all that is being presented is measured data. In other work we show that a generalized Pareto distribution fits the empirical data well, but this discussion is omitted here for reasons of space. As a Pareto distribution, the shape of the distribution follows a power law.

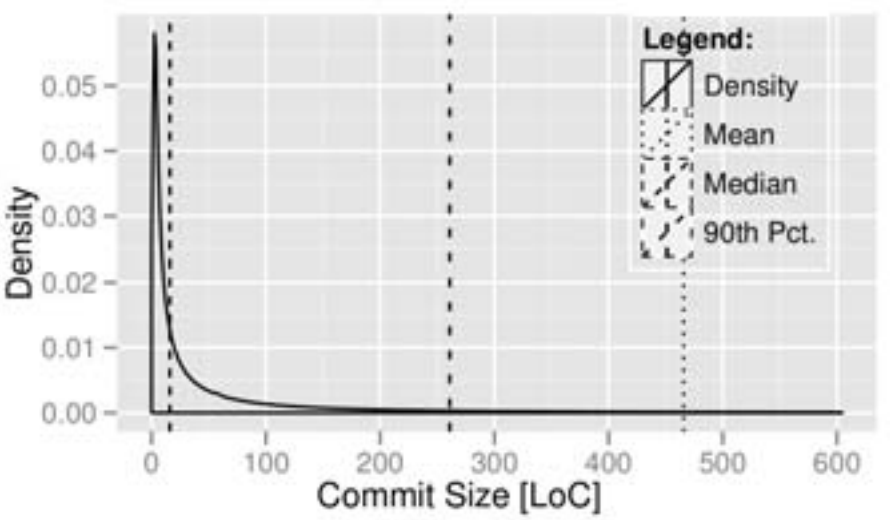


Figure 7: The (empirical) PDF of the commit size distribution of about 30% of all active open source projects (March 2008). The graphs in Figures 7-9 have been calculated using R and ggplot2, which uses a Gaussian smoothing kernel with the standard deviation as bandwidth.

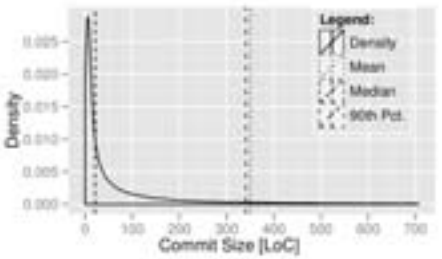


Figure 8: The PDF of the commit size distribution of the closed source BAS project at age 10-19 years

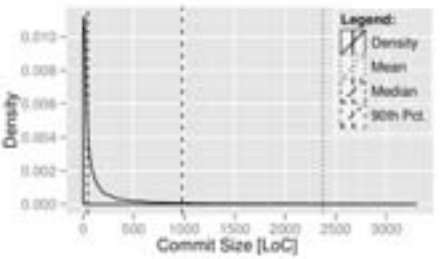


Figure 9: The PDF of the commit size distribution of 122 closed source projects at age 0-3 years

The mode of the distribution is 1.5 LoC. This real number is an artifact of measuring commit sizes using the estimation algorithm discussed in Section 2. An added and a removed line of code maps as often on one changed line of code as on one added and one separately removed line of code, averaging out to 1.5 LoC.

4.2 Closed Source Distributions

This section presents an analysis of selected projects of SAP AG, one of the world's largest software producers. We retrieved data for two types of projects:

- SAP's core virtual machine and libraries, the BAS project, at age 10-19 years, and
- 122 of SAP's research projects during their first three years of life.

We chose these projects due to the stark discrepancy between the two types of projects. The core virtual machine and libraries (the "BAS" project, short for base) is perhaps the most robust and thoroughly tested code SAP has ever developed and it forms the base of its software stack. It is written in C. The commit count of the BAS project is 56,840 commits and the configuration management system is Perforce.

The 122 research projects are young projects with a high mortality rate. The code base is undergoing wild changes and rapid development. They are written in C-style languages like C, C++, and Java. The commit count for the projects is 23,271 commits and the configuration management system is Subversion.

Figure 8 shows the PDF of the distribution of SAP's BAS project and Figure 9 shows the PDF of the distribution of the 122 research projects. The research projects all had similar distributions and like for open source have been grouped as one.

The commit size distribution of the BAS project is similar to the one of the research projects. This may come as a surprise given that a hypothesis might have been that mature projects develop more incrementally and using smaller code changes than young projects, which might move forward in (code) leaps and bounds. However, even young research projects apparently are moving forward mostly in small incremental commits.

Thus, the commit size distributions of (a) about 30% of open source (at its time), of (b) a mature closed source project, and of (c) 122 young research projects undergoing rapid change all have similarily-shaped strictly falling distributions that follow a power law, with 1-3 lines of code being most common. The actual parameters of the distributions differ, however, between open source and our closed source sample.

5. Belief vs. Reality

Table 4 summarizes the differences between reality and what developers said about commit size distributions. As to the mode of the distribution, the most frequent commit size, survey respondents are off by a wide margin. In the case of open source, the predicted value is 20 LoC with a measured value of 1.5 LoC and a relative error of 12. In the case of closed source, the predicted value is is 50 LoC with a real value of 3 LoC and a relative error of 16. Thus, respondents overestimate the mode significantly.

In general, the average developer believes that the commit size distribution has its maximum at around 20 LoC (open source) and 50 LoC (closed source) and then falls off rapidly as commit sizes increase. In reality, the commit size distributions of open source and closed source are strictly falling but have a longer tail than expected, with the mean value and 90th percentile farther out than what developers are expecting.

The long tail of the distribution, both open source in general and selected closed source projects, becomes evident by the 90th percentile of commit sizes being lower than the mean value of the commit sizes: This implies that a significant number of large commits are regularly being contributed, more than anticipated by survey respondents. The mode

of the open and closed source distributions are close to each other, suggesting little practical difference between open and closed source projects. The mean and the 90th percentile are different between open and closed source. Also, survey respondents underestimated the mean significantly.

One might argue that it is hard even for technically schooled people to distinguish mode from mean from median. However, in our survey [14] we did not use these words, but visualized their meaning. We believe that respondents had a sufficiently good understanding of these terms when making their choices.

Table 4: Belief, reality, absolute, and relative error for open and closed source distributions

	1. Open Source (Large Sample)				2. Closed Source (Selected Projects)				3. Difference Between Open & Closed Source			
	Belief (median)	Reality	Absolute Error	Relative Error	Belief (median)	Reality	Absolute Error	Relative Error	Belief (median)	Reality	Absolute Error	Relative Error
Mode [LoC]	20	1.5	18.5	12.33	50	3	47	15.67	30	1.5	28.5	19
Mean [LoC]	50	465.7	415.7	0.8926	80	942.0	862.0	0.9151	55	476.3	421.3	0.8845
90th Pctle [LoC]	200	261	61	0.2337	200	461.5	261.5	0.5666	100	200.5	100.5	0.5012

Table 5: Sampling error in survey respondents' answers when viewed as a random sample of the total developer population

	1. Open Source (Large Sample)				2. Closed Source (Sel. Projects)			
	25th Percentile		50th Percentile		25th Percentile		50th Percentile	
	Absolute Error	Relative Error	Absolute Error	Relative Error	Absolute Error	Relative Error	Absolute Error	Relative Error
Mode [LoC]	8.5	5.667	18.5	12.33	12.00	4.000	47.00	15.67
Mean [LoC]	365.7	0.7853	415.7	0.8926	742.0	0.7877	864.5	0.9177
90th Percentile [LoC]	161.0	0.6169	211.0	0.8084	261.5	0.5666	361.5	0.7833
Sampling Error [%]	10.00 (9.333)		11.55 (11.47)		10.00		11.55	

6. Threats to Validity

We address a possible sampling error of survey responses, age of open and closed source data, issues with measuring commit sizes and representativeness of project data.

6.1 Survey Responses

With 73 valid and complete survey responses, the response rate is on the low side. However, from the demographics gathered in the survey, there was no apparent bias. Thus, the question of the representativeness of the survey becomes one of sampling error.

Table 5 above presents the difference between the medians of the predicted and the real commit sizes. To show that the results of Table 5 are representative we first calculate the difference between the predicted and the real value for each survey respondent. Table 5 shows the 25th percentile and the 50th percentile of that difference.

For the 50th percentile, the sampling error for the survey responses is 11.55% at a confidence level of 95%. Thus, even in the worst case scenario more than 38% of all developers have an error above the 50th percentile. We also calculate the sampling error for the 25th percentile which is 10.00% at a confidence level of 95%. Thus, at least 65% of all developers are off by this error. This shows that even using conservative estimates we can consider the survey responses as representative.

6.2 Age of Analysis Data

Our Ohloh database snapshot is dated March 2008. However, for this analysis the age doesn't matter much. We have no indication that open source changed significantly from 2008 to today (2011). Hence we believe that adding three years to an analysis history of 15+ years will not change the results in any significant way.

6.3 Measuring Commit Sizes

We spent considerable effort on developing a well-performing algorithm for determining commit sizes (Section 2). The core equation can be computed fast but only provides an estimate of the size of a given commit. Thus, in any given instance it may be off, but when measured over large commit size populations, it will be accurate [9]. This is the situation we were facing in this paper. The commit size population is greater than 8 million commits and is sufficiently large to allow for the probabilistic estimation of commit sizes by the simple heuristic of Section 2 that we use throughout this paper.

6.4 Representativeness of Project Data

Our open source sample population is close to being representative of open source. With about 30% of all active open source projects at the time of the database snapshot, we have captured a significant chunk of the total population. As discussed, except for self-reporting and English-language bias, we find no apparent bias in our sample. Moreover, there is no apparent connection between this potential bias and the measure of interest, commit sizes. Thus, we believe that the commit size distribution of open source is close or identical to the one presented in this paper.

Our closed source sample population is from a single vendor only and hence not representative. The question becomes how far off from representative values are our closed source measurements? First, there is no significant difference between a mature and many young projects. Moreover, there is no significant difference between the general open source distribution and the commit size distribution of the closed source project sample. All of this is contrary to developer belief and our own prior expectations.

While we cannot overcome the fundamental problem the surprising similarity between the open source and closed source distributions suggests that to the extent that they are similar, the degree of representativeness of our open source sample applies to the representativeness of our closed source sample.

7. Related Work

We previously presented an analysis of the open source commit size distribution [2]. In comparison to that analysis, the work presented in this paper adds the survey, closed source data, and a more thorough investigation of the open source commit size distribution including using the more precise heuristic for estimating commit sizes of Section 2.

Alali et al. present an analysis of "a typical commit" using the version history of 9 open source projects [1]. They mostly focus on the number of files changed (and how), but also provide chunk and line-size data. They compute line size changes by adding lines added and removed, thus overestimating sizes by ignoring changed lines of code. Still, they find "quite small" commit sizes without giving more details. Interestingly, they find a strong correlation between diff chunk and size. Alali et al.'s 9 projects are large well-known open source projects. In contrast to Alali we focus solely on commit sizes, use a more precise measure and compute a derived function, the commit size distribution, on a close-to-representative sample rather than 9 selected projects.

Hattori and Lanza discuss “the nature of commits” in which they look at 9 different open source projects [7]. They measure commit sizes by number of files changed and find a distribution similar to ours. Beyond the distribution, their work is mostly about classifying commits while we focus on the difference between developer belief and reality. Similarly, Hindle et al. analyze 2000 large commits from 9 selected open source projects and find that small commits are more corrective while large commits are more perfective [8].

Purushothaman and Perry analyze the impact that small changes have on quality attributes of software under consideration [13]. Their data is derived from a single large closed source project. They find that one-line changes represent the majority of changes during maintenance, which is in line with our results. Implicit in the choice of research topics by Purushothama and Perry as well as Hindle et al. may be an assumption that commit sizes are smaller in maintenance than in development mode. Our closed source data does not immediately support such a hypothesis but it is worth investigating further.

Weißgerber et al. look at the patch submission and acceptance process of two open source projects [16]. They find that small patches are more likely to get accepted into the code base than large patches. An obvious reason may be that small patches are easier to review than large patches which, if not handled quickly, get harder to review and accept with time. While not representative, Weißgerber's observation is interesting to us, as it might explain why the commit size distribution of open source is falling more quickly than those of the closed source projects we analyzed.

8. Conclusions

The paper shows that the commit size distribution of open source runs counter to the expectations of respondents from a software developer survey we undertook. The only exception was the 90th percentile, which respondents were able to predict reasonably well. Survey respondents could not agree on the form of the distribution and expected a much higher mode for commit sizes in open source as well as closed source software development. They also expected the distribution to fall of more quickly than it does. Thus, developers underestimate the significance of small commits and don't realize the long tail of commit sizes. Moreover, respondents expected to see significant differences between open and closed source software development, and our data shows these differences aren't there.

The survey findings are particularly significant, because we found no difference of opinion between tool developers and regular developers. Thus, a reasonable assumption is that the conceptual model of software tool developers may not be well aligned with reality when it comes to the commit size distribution. Our work provides that reality and can be used to improve the design of software development tools. We can now rethink the use of screen real estate in merge tools, or the design and implementation of configuration management systems, or how to improve change impact analysis using the commit size statistics that our paper presents.

It is left for us to speculate why survey respondents were off to the observable extent. One possible answer is that commit sizes may have become smaller over time so that past experiences unduly affected respondents' judgment. Such cognitive bias is supported by psychology research that reports that humans hesitate to go to the extremes but rather remain conservative estimators [11]. Other explanations are possible as well, but we leave them to psychologists and future work.

Acknowledgements

We would like to thank York Thomas of HPI for providing us with the SAP data. We would also like to thank Manuel Klimek of Google for helping us with the survey distribution. We would like to thank Wolfgang Mauerer of Siemens and Lutz Prechelt of FU Berlin for feedback on the paper. We would like to thank Oscar Nierstrasz and Niko Schwarz for helping us improve the paper and for contributing the references about psychology research on cognitive biases. We would like to thank all other reviewers for their help as well.

References

[1] Alali, A., Kagdi, H., Maletic, J. I. (2008). What's a typical commit? A characterization of open source software repositories. In *Proceedings of the 16th IEEE International Conference on Program Comprehension* (ICPC '08), 182-191.

[2] Arafat, O., Riehle, D. (2009). The commit size distribution of open source software. In *Proceedings of the 42nd Hawaiian International Conference on System Sciences* (HICSS 42), 1-8.

[3] Canfora, G., Cerulo, L., Di Penta, M. (2009). Ldiff: An enhanced line differencing tool. In *Proceedings of the 31st International Conference on Software Engineering* (ICSE '09), 595–598.

[4] Cosmides, L., Tooby, J. (1996). Are humans good statisticians after all? Rethinking some conclusions from the literature on judgment under uncertainty. *Cognition* 58 (1996), 1-73.

[5] Daffara, C. (2007). Estimating the number of active and stable FLOSS projects. 2011-11-24. URL: http://robertogaloppini.net/2007/08/23/estimating-the-number-of-active-and-stable-floss-projects/. Accessed: 2011-11-24. Archived: http://www.webcitation.org/63QrHacPr.

[6] Gilovich, T., Griffin, D., Kahneman, D. (2002). Heuristics and biases: The psychology of intuitive judgment. Cambridge University Press.

[7] Hattori, L.P., Lanza, M. (2008). On the nature of commits. In *Proceedings of Workshops of the ASE 2008*, the 23rd IEEE/ACM International Conference on Automated Software Engineering, 63-71.

[8] Hindle A., German, D.M., Holt, R.C. (2008). What do large commits tell us? A taxonomical study of large commits. In *Proceedings of the 5th International Working Conference on Mining Software Repositories* (MSR 2008), 99-108.

[9] Hofmann, P., Riehle, D. Estimating commit sizes efficiently. In *Proceedings of the 5th International Conference on Open Source Systems* (OSS 2009), 105-115.

[10] Howison, J., Crowston, K. (2004). The perils and pitfalls of mining SourceForge. In *Proceedings of the 1st International Workshop on Mining Software Repositories* (MSR 2004), 7-11.

[11] Peterson, C., Beach L.R. (1967). Man as an intuitive statistician. *Psychological Bulletin* 68, 29-46.

[12] Peterson, C., Miller, A. (1968). Mode, median, and mean as optimal strategies. *Journal of Experimental Psychology* 68, 363-367.

[13] Purushothaman, R., Perry, D. (2004). Towards understanding the rhetoric of small changes. In *Proceedings of the International Workshop on Mining Software Repositories* (MSR 2004), 90-94.

[14] Riehle, D. (2009). Commits in Software Development (Survey). Accessed: 2011-11-29. Archived: http://www.webcitation.org/63Z72ULbu.

[15] Tversky, A. Kahneman, D (1974). Judgement under uncertainty: heuristics and biases. *Science* 185, 1124-1131.

[16] Weißgerber, P., Neu, D., Diehl, S. (2008). Small patches get in! In *Proceedings of the Fift*h Int'l Workshop on Mining Software Repositories (MSR 2008), 67pp.

Deriving Quality-based Architecture Alternatives with Patterns*

Marco Konersmann, Azadeh Alebrahim, Maritta Heisel, Michael Goedicke, Benjamin Kersten
paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen, Germany
{marco.konersmann, azadeh.alebrahim, maritta.heisel, michael.goedicke,
benjamin.kersten}@paluno.uni-due.de

Abstract: We propose in this paper an iterative method composed of three steps to derive architecture alternatives from quality requirements using a catalogue of patterns and styles. The solution candidates are chosen by answering a set of questions which reflects the requirements. We instantiate then the solution candidates using a UML-based enhancement of the problem frame approach. To ensure that the instantiated architectures fulfill the quality requirements, we evaluate them in the next step. A desired refinement of the software architectures is then achieved by iterating over the described steps.

1 Introduction

The development of software architecture is a challenging task, even when the requirements of a software system are clear. For building upon common knowledge and best practices, the use of catalogues containing architectural patterns and styles (e.g. [TMD09]) has shown to be valuable. These catalogues are used in architectural design methods, that aim to give guidance for deriving architectures from requirements (e.g. [BCK03, HNS99]). In these methods, catalogues are used as a reference to find solutions for an architectural problem by choosing appropriate patterns and styles (called *solution candidates* in this document) from the catalogue. Trying to find appropriate solution candidates in a catalogue is, however, not trivial. Usually more than one candidate is appropriate, but they differ in their quality attributes. The existing approaches are imprecise or do not provide any aid for finding good candidates from catalogues [BR10]. Sequentially evaluating the candidates of a catalogue is not efficient.

In this paper we propose an iterative method to derive alternative architectures that differ in their quality attributes. The method integrates our approaches to this problem in [AHH11] and [MKG11]. We first obtain alternative solution candidates proposed from catalogues by relating questions to these candidates. Answers to these questions rate the solution candidates with respect to the requirements of the system to develop. In the second step, we derive architecture alternatives from the best-rated candidates. For the derivation we

*The work reported herein was funded by the German Research Foundation under the grants no. GO774/5-1 and HE3322/4-1.

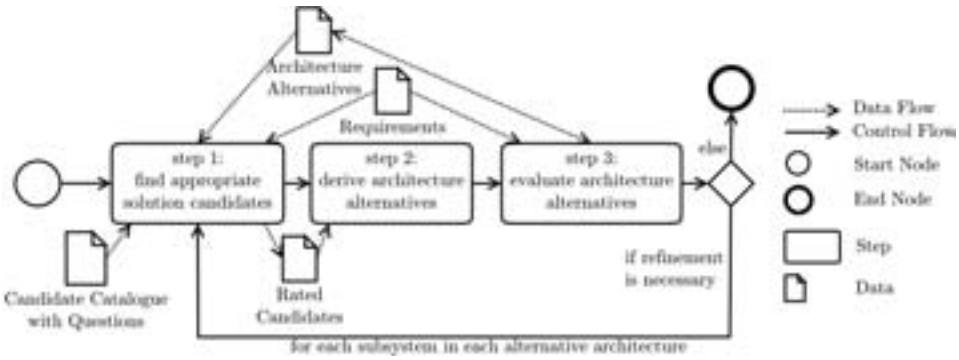


Figure 1: An overview of the proposed method for architecture derivation

make use of an enhancement of the problem frame approach with the focus on quality requirements. In the third step we evaluate the derived architectures against the quality requirements. To refine the architectures we iterate over these three steps. In each further iteration, subsystems of the architecture alternatives are refined.

We illustrate our approach with a chat application, which allows a text-message-based communication via private I/O devices. Users should be able to communicate with other chat participants in a same chat room. We focus on the functional *Communicate* requirement with the description "*Users can send text messages to a chat room. The text messages will be shown to the users in that chat room in the current chat session in the correct temporal order on the users' displays*" and its corresponding quality requirement *Response Time* with the description "*The sent text message should be shown on the user's display in 1500 ms maximum*". Note that in order to specify performance requirements properly, more details have to be given. We use the MARTE profile [UML] for this purpose.

The remainder of this paper is structured as follows: Section 2 describes the approach using the chat application as running example. The approach is then discussed in section 3. In section 4 we differentiate the approach from related work, before we conclude and present future work in section 5.

2 Architecture Alternative Derivation Method

The method for deriving architecture alternatives is separated into three steps. In step 1 (section 2.1), solution candidates are rated with respect to the system's requirements. In step 2 (section 2.2) alternative architectures are derived from the best-rated solution candidates. The alternatives are functionally equivalent to an external observer. Their internal structure and behaviour differs. The alternatives are evaluated against the quality requirements in step 3 (section 2.3). These three steps have to be executed manually. We currently develop tool support for the process. To refine the alternatives we iterate over these three steps (section 2.4). In each iteration, subsystems of the derived architecture alternatives

are refined. By refining each subsystem of each positively evaluated alternative, a tree of alternative architectures is spanned. Figure 1 gives an overview of the method.

2.1 Step 1: Find Appropriate Solution Candidates

Figure 2 describes the step for finding promising candidates. This step is described in detail in [MKG11]. The inputs are the system's requirements, and a catalogue of solution candidates. The catalogue includes candidates, that reference questions. The questions are targeted at software quality, e.g. scalability or security. Answers to those questions impact the rating for a candidate. These ratings vary between *-1 (probably not appropriate)* and *1 (probably appropriate)* or *excludes*. The latter means that the candidate cannot be applied.

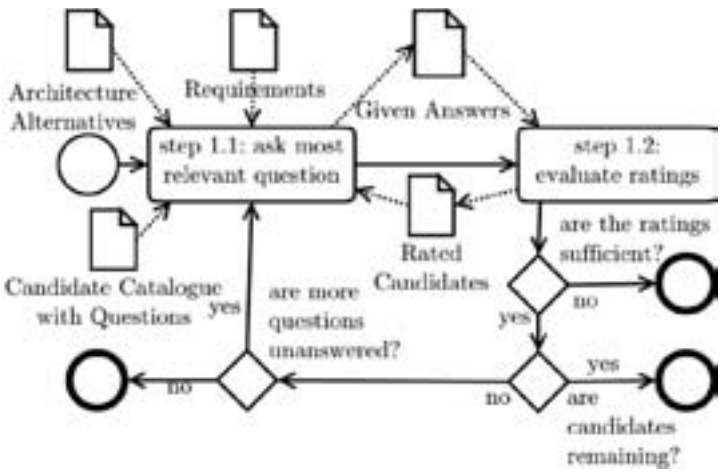


Figure 2: An overview of the method's step 1

Step 1.1: Ask Most-Relevant Question

In step 1.1, the questions are sorted in decreasing order by the number of references from candidates. Then, the most referenced question is asked. That question has an impact on the most candidates. Due to the order, the impact of the answers decrease with every asked question. When a question is only referenced by excluded candidates, it will not be asked.

Step 1.2: Evaluate Ratings

In step 1.2 the candidates are sorted in decreasing order by their ratings. A candidate's rating is calculated by each of its answer's rating. If any answer's rating is *excludes*, then the candidate is excluded. Else the result is the arithmetic mean of each answer's

Candidate	Q1	Q2	Q3	...	Arithmetic Mean
Client/Server	1	0.5	0.9		0.19
Simple Peer-to-Peer	1	0	-0.1		0.22
Standalone	excludes	x	x		0.03
Pipes & Filters	0.2	0	x		0.0
Publish-Subscribe	0.2	0.4	-0.1		0.14
Layered	x	x	0.3	...	0.11
...	...	...	...	...	

Q1: Is the system necessarily distributed? → Yes
Q2: Are high request-peaks expected? → No
Q3: Does the system conduct sensible / confidential data? → Yes

Table 1: Rated answers for the first iteration of the chat application. The columns 2 to 4 show the ratings for answers given to the questions Q1 to Q3. The questions and answers are shown below the table. The last column shows the candidate rating.

ratings. The ratings give guidance to find out which candidates are probably appropriate. If more candidates are remaining, the ratings are not perceived detailed enough, and more questions remain unanswered, step 1.1 will be repeated with the next-most referenced question.

In our example, a pattern catalogue with 10 patterns referencing 25 questions was used. The ratings were defined by experience. An excerpt of the questions and given answers in the first method iteration is shown in table 1. The table shows only the ratings for the given answers. As a result of this step, the candidates *Client/Server* and *Simple Peer-to-Peer* were identified to be the most-promising candidates. The other candidates were excluded or had a lower rating.

2.2 Step 2: Derive Architecture Alternatives

In this step we derive the structural view of the architecture by instantiating architecture alternatives with all solution candidates that we obtained from the previous step. In order to derive a component-based architecture we need to decompose the overall problem in subproblems. For this purpose we use a requirements engineering process based on the problem frames approach [Jac01], which we describe briefly in this section. Then we instantiate the solution candidates by using the results of applying the problem frames approach. Figure 3 shows an overview of the second step.

Requirements Engineering using Problem Frames

We describe briefly an extension of Jackson's problem frames [Jac01], which we apply to instantiate the solution candidates. This approach uses a UML profile [CHH11] that extends the UML meta-model to support problem-frame-based requirements analysis. We

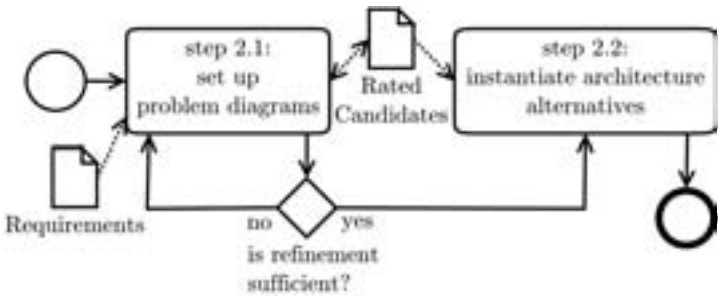


Figure 3: An overview of the method's step 2

use this profile to create the diagrams for the problem frames approach.

Problem frames are a means to describe software development problems. A problem frame is described by a *frame diagram*, which basically consists of *domains*, *interfaces* between them, and a *requirement*. The task is to construct a *machine* (i.e., software) that improves the behavior of the environment (in which it is integrated) in accordance with the requirements.

Requirements analysis with problem frames proceeds as follows: first the environment in which the machine will operate is represented in a *context diagram*. A context diagram consists of a machine, domains and interfaces. In the UML profile the context diagram is represented by the stereotype $\ll$`contextDiagram`$\gg$ [1]. Then, the problem is decomposed into subproblems, which are represented by *problem diagrams*. A problem diagram consists of a submachine of the machine given in the context diagram, the relevant domains, the interfaces between these domains, and a requirement.

Jackson distinguishes the domain types biddable domains that are usually people, causal domains that comply with some physical laws, and lexical domains that are data representations. In problem diagrams a connection domain establishes a connection between two other domains by means of technical devices. Examples are video cameras, sensors, or networks. A display domain represent a special case of a causal domain (introduced in [CHH$^+$08]). In problem diagrams *interfaces* connect domains, and they contain *shared phenomena*. Shared phenomena may e.g. be events, operation calls or messages. They are observable by at least two domains, but controlled by only one domain, as indicated by "!". In Fig. 4 the notation $U!\{sendTM\}$ (between *CA_communicate* and *User*) means that the phenomenon *sendTM* is controlled by the domain *User*.

Each functional requirement constrains at least one domain to show the desire of the change of something in the world. A requirement may refer to several domains in the environment of the machine. Functional requirements are complemented by quality requirements. In the UML profile these relations are shown by corresponding dependencies. To provide support for annotating problem descriptions with performance requirements, we use the UML profile MARTE (Modeling and Analysis of Real-time and Embedded

[1] In the UML profile each model element is represented by the corresponding stereotype

Systems) [UML].

The problem diagram in Figure 4 considers the chat application that was introduced in section 1. It describes the functional requirement *Communicate*. E.g., it states that the *CA_communicate* machine can show to the User the *CurrentChatSession* on its Display ($CAC!\{displayCCS\}$). The requirement constrains the *CurrentChatSession* of the *User* and its *Display*. The requirement refers to the users and the text messages. The requirement *Communicate_RT* describes the response time requirement, which complements the functional requirement *Communicate*. The requirement *Communicate_Conf* describes the confidentiality requirement complementing the functional requirement *Communicate*. In this paper we focus on the response time requirement.

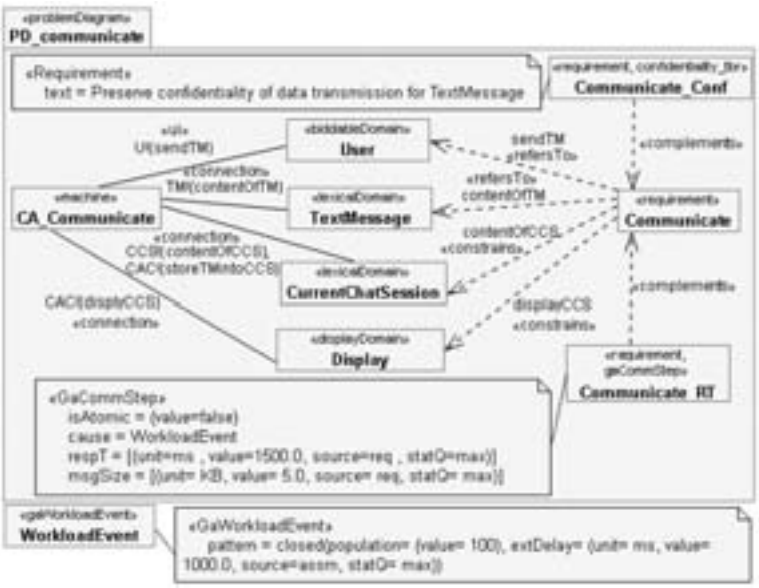


Figure 4: Problem diagram for the requirement *Communicate*

Step 2.1: Set Up Problem Diagrams

In step 2.1 we first set up the problem diagrams to prepare for the instantiation step. We decompose the overall problem into subproblems. Each problem diagram describes one subproblem with the corresponding requirement. Then we annotate each subproblem by complementing functional requirements with related quality requirements. Decomposing the overall problem into subproblems using problem frames is described in detail in [AHH11].

Then we take into account the architectural styles and patterns we obtained by answering the questions. After having chosen the appropriate candidates, we go back to the requirements descriptions and decompose the problem diagrams with respect to the chosen candidates for the architecture. This may lead us to introduce connection domains,

e.g., networks. Analogously to decomposing the problem diagrams and so decomposing the functional requirements, we also have to decompose the corresponding quality requirements.

In order to apply this step on the chat example, we first need to decompose the overall chat problem into its subproblems. We focus on the functional requirement *Communicate* (see figure 4) as one subproblem. Then we address quality requirements by annotating the functional requirements with complementing quality requirements. E.g., the requirement *Communicate* is complemented by the response time requirement *Communicate_RT*. The quality requirement in this example is modeled using the MARTE profile for performance annotations. The *respT* attribute states that the required response time for sending text messages should be 1500 ms as maximum. The *cause* attribute represents the triggering event, which is in our case a *ClosedPattern* with 100 concurrent users (*population*), each of which needs a think time of 1000 ms (*extDelay*). The *msgSize* attribute states that the sending text messages should be 5 KB maximum.

In order to decompose problem diagrams properly we take into account the solution candidates we obtained from step 1. We proceed the example with the *Client/Server* alternative and describe each step of its instantiation in detail. After having chosen the *Client/Server* architecture style we decompose the problem diagrams so that each subproblem is allocated to only one of the distributed components.

In our example, we decompose the problem diagram depicted in figure 4 into three subproblem diagrams *Send* (Client), *Forward* (Server), and *Receive* (Client). *Send* addresses the problem of sending text messages to the server (not shown). *Forward* addresses the forwarding of text messages from the server to the receivers (see figure 5). *Receive* addresses the receiving of text messages (not shown). For each of these three subproblems, we introduced the connection domain *Network* to achieve the distribution.

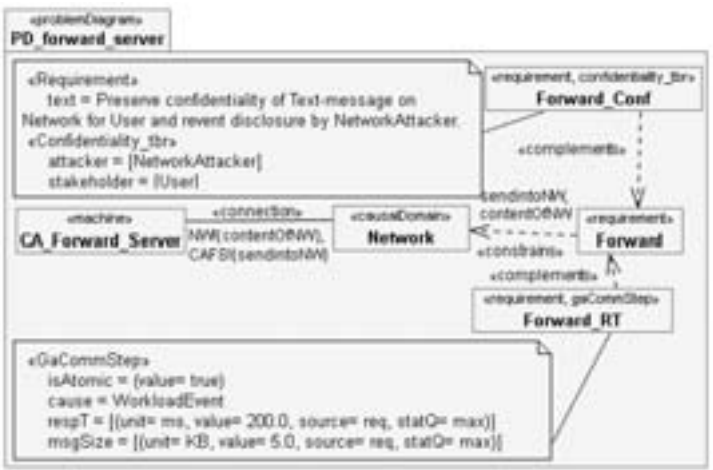


Figure 5: Problem diagram for the subproblem *Forward*, annotated with quality requirements

To fulfill the response time requirement, the response time should be divided so that all subproblems together satisfy the desired response time. The *Communicate* requirement states a response time of 1500 ms maximum. This must be achieved through the three subproblems. We must also consider the time that the data needs to be transported over the network. In our example, each of the machines *CA_send* and *CA_forward* is required to send a text message to the server or to forward the text message to the receivers, respectively, within 200 ms. The machine *CA_receive* may take 300 ms to process the received text message and display it. This leaves 800 ms to transmit data from the client to the server and back. Note that knowledge about the real circumstances in the environment e.g. about the network and the computational power of clients and server is needed to meet performance and specifically response time requirements.

Step 2.2: Instantiate Architecture Alternatives

In step 2.2 we derive the architecture by instantiating the solution candidates through subproblems. The initial architecture that we have to refine consists of one component for the overall machine. Each submachine, which belongs to one decomposed problem diagram becomes a component in the architecture. In further iterations we refine each component by introducing domains reflecting specific solution approaches we obtained from the decomposed problem diagrams.

For the chat example we derive the architecture in the first iteration by establishing one component for the overall machine *chat application*. To indicate the distribution we add the stereotype ≪`distributed`≫ from the UML profile for problem frames to the architecture component. For the *Client/Server* architecture style, there are two components representing the clients and the server, respectively, inside the overall machine. Then we make use of the subproblem diagrams. Each submachine in the subproblem diagrams becomes a component either in the client or in the server. The submachines *CA_send* and *CA_receive* belong to the client component, whereas *CA_forward* belongs to the server.

2.3 Step 3: Evaluate Architecture Alternatives

In the evaluation step of our method, the derived architecture alternatives have to be evaluated against the system's quality requirements. Typically, a trade-off analysis will be part of the evaluation, because system qualities are often contradictory. This method does not constrain the choice of quality evaluation methods. As a result of the evaluation, some of the derived architecture alternatives can be excluded when they show to perform worse than other alternatives regarding the quality requirements. Other alternatives will be considered for refinement in the method's next iterations.

As the evaluation method is not in focus of this paper, we will not go into details here. In our example both candidates passed the evaluation. Thus in the next iterations, each candidate is considered further when the architecture is refined.

2.4 Further Iterations: Architecture Refinement

Software architectures can be specified at different levels of details. Our method reflects this by allowing for arbitrary method iterations. In the first iteration, alternatives for the overall system are derived. In each further iteration, subsystems of these alternatives are refined by executing the method with the focus on a subsystem, instead of the overall system. Consequently, only the set of requirements are considered, that are relevant to the specific subsystem in focus. The candidates ratings are reset, so that the questions need to be answered again, though against the focused subsystem's requirements.

In our example, we first focus on the *Client/Server* alternative. Within this alternative we refine the subsystem *Client*. By answering the questions in step 1, *Load Balancer* is rated as an appropriate solution candidate. In step 2, we elaborate the problem diagrams by introducing domains reflecting load balancing as a solution candidates. To specify the quality problem diagram given in figure 6 we introduce a new machine *LoadBalancer* that distributes the load from the network across several server components, each of which contains one machine for solving the *Forward* problem.

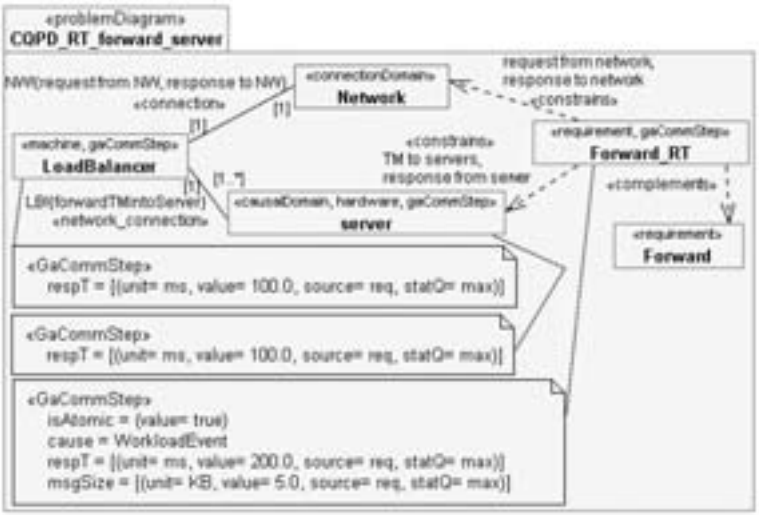


Figure 6: Problem diagram for the quality requirement *Forward_RT*

After elaborating all problem diagrams with solution candidates and instantiating the software architecture with them, we obtain the software architecture shown in figure 7 as one alternative with the *Client/Server* architectural style. The *LoadBalancer* is placed before the servers. Its port multiplicity [1..*] means that it can be connected with several server components. The Peer-to-Peer alternative and the refinement of the Client subsystem are handled analogously. This is not shown in this paper.

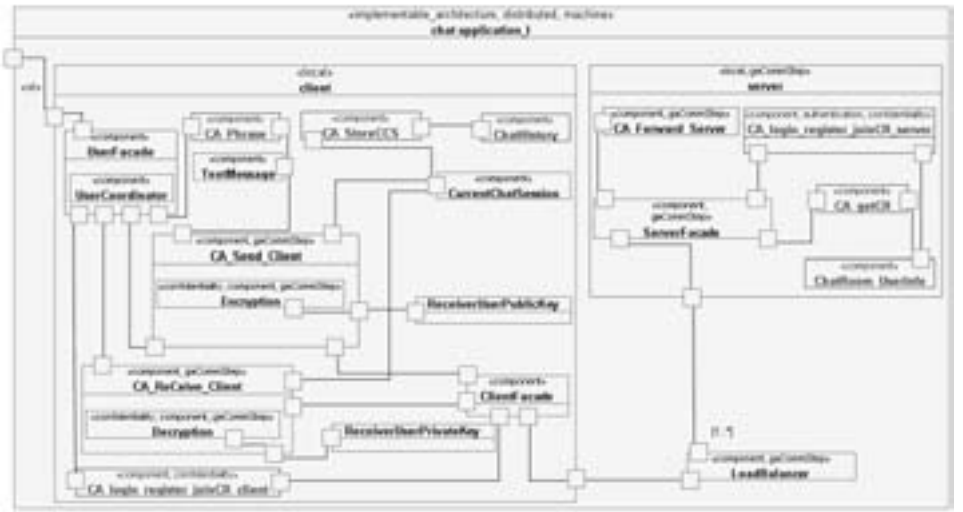


Figure 7: Implementable architecture *Communicate*

3 Discussion

The task of architectural design cannot be completely automated. Thus the approach presented here can also only give guidance. The architect has to interpret the results of the candidate ratings and choose which candidate to consider. While the ratings suggest that a candidate with a rating (arithmetic mean of all single ratings) of 0.7 is more appropriate than a candidate with the rating 0.6, this is not necessarily true. The step of instantiation also leaves choices to the architect. Thus the method is non-deterministic.

The success of the method is highly dependent on the amount and quality of data it is based on, i.e. questions, ratings, and candidates. The current set of data is small and provides only a few candidates and questions. More data should be provided by a community of experienced architects. We plan on extending the data base.

The proposed method aims at systematically deriving architecture alternatives that differ in their quality attributes, taking the quality requirements into account. Our experiments (on a distributed, highly configurable load generator and on smaller information systems, all developed in one of our working groups) indicate that the method is usable and helps to systematically explore the design space of a system, while reducing the development effort by highlighting appropriate alternatives. A systematic evaluation is still to be performed.

4 Related Work

The systematic derivation of architectures from requirements has been subject to research in related work for many years [SB82, Nus01]. The Twin Peaks Model [Nus01] integrates

requirements elicitation with architectural design. In this method, requirements and architecture are refined concurrently, while providing feedback to each other. However, the details and mechanisms of the feedback is not specified. Also, in contrast to our approach, no alternatives are developed, that can be considered in an evaluation.

In [vL03] van Lamsweerde proposes a goal-oriented approach to architectural design. Van Lamsweerde first considers functional requirements, before adapting the architecture to meet quality requirements. In our approach, we focus on quality requirements first by choosing the solution candidates based on the quality requirements. We then add the required functionality in a systematic manner.

Attribute Driven Design (ADD) is a method that emphasizes the use of patterns and styles while deriving an architecture from requirements. However, ADD does not describe how to elicit a matching pattern from a large pattern catalogue, except for sequentially evaluating each element in the catalogue (cf. [WBB+07]). There are more architecture methods that are imprecise at this point such as Siemens 4 Views [HNS99]. Therefore, our approach bridges the gap of pattern elicitation found in related work.

Zdun uses questions to select architecture patterns in [Zdu07]. In this approach, questions are directed to key characteristics of a group of patterns (e.g. "*How to realize asynchronous result handling?*"). The answers are patterns, which are related to criteria supporting the decision process. Other approaches (e.g. [HA07]) relate patterns to coarse-grained quality attributes. We believe that our approach is more suitable for less experienced teams, because they are more fine-grained and related to the system's requirements and context. This is, however, subject to validation.

Bode and Riebisch [BR10] relate solutions to quality goals. Their work focuses on rating the impact of patterns on quality goals. Bode and Riebisch develop context-independent ratings, aiming at fulfilling abstract quality goals. In contrast, our approach also takes the system's context and environmental constraints into account by explicitly considering the requirements, e.g. insecure networks that have to be used. We are confident that this allows to rate solution candidates more precisely, because such details can have a strong impact on the choice of patterns and styles.

5 Conclusion and Future Work

In this paper, we presented a method for deriving architectural alternatives for software systems. The method allows for systematically exploring the design space and aims at highlighting appropriate alternatives. It is iterative, so that it can be performed until the desired level of detail is reached. Our experiments have shown that the method is usable and helpful.

As future work, we plan to find mechanisms or a community to provide data, as the quantity and quality of ratings and patterns is very important to render the method helpful. In addition, a case study is work in progress.

References

[AHH11] Azadeh Alebrahim, Denis Hatebur, and Maritta Heisel. Towards Systematic Integration of Quality Requirements into Software Architecture. In Ivica Crnkovic, Volker Gruhn, and Matthias Book, editors, *Proceedings of the 5th European Conference on Software Architecture (ECSA)*, LNCS 6903, pages 17–25. Springer, 2011.

[BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice (SEI Series in Software Engineering)*. Addison-Wesley, 2. a. 2003. edition, 4 2003.

[BR10] Stephan Bode and Matthias Riebisch. Impact Evaluation for Quality-Oriented Architectural Decisions regarding Evolvability. In Muhammad Babar and Ian Gorton, editors, *Software Architecture*, LNCS 6285, pages 182–197. Springer, 2010.

[CHH+08] Isabelle Côté, Denis Hatebur, Maritta Heisel, Holger Schmidt, and Ina Wentzlaff. A Systematic Account of Problem Frames. In *Proc. of the European Conf. on Pattern Languages of Programs (EuroPLoP)*, pages 749–767. Universitätsverlag Konstanz, 2008.

[CHH11] C. Choppy, D. Hatebur, and M. Heisel. Systematic Architectural Design based on Problem Patterns. In P. Avgeriou, J. Grundy, J. Hall, P. Lago, and I. Mistrik, editors, *Relating Software Requirements and Architectures*, pages 133–160. Springer, 2011.

[HA07] Neil Harrison and Paris Avgeriou. Leveraging Architecture Patterns to Satisfy Quality Attributes. In Flavio Oquendo, editor, *Software Architecture*, volume 4758 of *Lecture Notes in Computer Science*, pages 263–270. Springer Berlin / Heidelberg, 2007.

[HNS99] Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied Software Architecture: A Practical Guide for Software Designers*. Addison-Wesley, 11 1999.

[Jac01] M. Jackson. *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley, 2001.

[MKG11] Marco Müller, Benjamin Kersten, and Michael Goedicke. A Question-Based Method for Deriving Software Architectures. In Ivica Crnkovic, Volker Gruhn, and Matthias Book, editors, *Proceedings of the 5th European Conference on Software Architecture (ECSA)*, LNCS 6903, pages 35–42. Springer, 2011.

[Nus01] Bashar Nuseibeh. Weaving Together Requirements and Architectures. *Computer*, 34:115–117, March 2001.

[SB82] William Swartout and Robert Balzer. On the inevitable intertwining of specification and implementation. *Commun. ACM*, 25:438–440, July 1982.

[TMD09] Richard N. Taylor, Nenad Medvidovic, and Eric Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 1. auflage edition, 2 2009.

[UML] UML Revision Task Force. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*. http://www.omg.org/spec/MARTE/1.0/PDF.

[vL03] Axel van Lamsweerde. From System Goals to Software Architecture. In Marco Bernardo and Paola Inverardi, editors, *Formal Methods for Software Architectures*, LNCS 2804, pages 25–43. Springer, 2003.

[WBB+07] Rob Wojcik, Felix Bachmann, Len Bass, Paul Clements, Paulo Merson, Robert Nord, and Bill Wood. Attribute-Driven Design (ADD), Version 2.0. Technical report, Software Engineering Institute, 2007.

[Zdu07] U. Zdun. Systematic Pattern Selection using Pattern Language Grammars and Design Space Analysis. *Software: Practice and Experience*, 37(9):983–1016, 2007.

Improving the software architecture design process by reusing technology-specific experience

Glib Kutepov

Information Systems Development
Fraunhofer IESE
Fraunhofer-Platz 1
67663 Kaiserslautern
glib.kutepov@iese.fraunhofer.de

Abstract: Experience with particular technologies such as SOA, Cloud Computing, or Mobile Apps plays a crucial role when designing the architecture of a software system. Being aware of the challenges usually encountered when using a technology and knowing in advance how to resolve these challenges can dramatically increase the quality of the software system architecture and decrease the design effort. However, it is not always a straightforward process to collect the necessary architectural experience, persist it on the organizational level, and reuse it in the right way, especially if the technology is new. This paper describes how architecture design processes can be improved by supplementing them with architectural experience related to a particular technology. The way architectural experience can be described using architectural scenarios and solution patterns is explained and persisted in the architecture design process. The efficiency of the approach is validated with the help of a case study.

1. Introduction

Consider a modern software development organization that develops its products with a strong focus on the architecture. During the software architecture design process, the architect identifies key challenges that influence the current system architecture and finds appropriate solutions. This task is carried out successfully because the architect bases his system on familiar, wide-spread IT technologies whose caveats are well known to him or her. Fortunately, the world of technology does not stand still: Every day, some new technology appears on the IT scene, which opens up new opportunities: maybe an alternative or an improvement to an existing technology, or a completely new technological paradigm, such as SOA, Cloud Computing, or Mobile Applications. The organization may have to incorporate the new technology in its architectural practices in order to stay competitive. Initially, the identification of challenges and solutions will be less effective than before due to a lack of knowledge on the architect's part regarding the new technology. This may increase the time needed for architecture design and decrease quality. However, with time the process will become more effective and the quality of the product will improve. This will be mostly due to the tacit experience of the software

architect. Having one person as the only source of this specific knowledge may jeopardize all the benefits of adopting the new technology. Thus, a way to systematically collect and persist the architect's experience in an organization has to be found.

Hofmeister et al. [HN99] suggest performing similar activities before designing every architectural view and refer to these activities as "Global Analysis". The core of "Global Analysis" is the identification of factor-strategy pairs, which are then applied in the actual view design phase. "Factors" represent the challenges or problems that may influence architecture design and "strategies" represent solutions to them. We see such pairs as an effective tool for describing and persisting technology-specific experience.

In this paper, we propose an approach for systematically collecting, describing, and reusing such pairs in the context of the architecture design process. The pairs are collected with the help of project post-mortems, while the factors are described in the form of architectural scenarios and strategies with solution patterns. Furthermore, several scenarios are provided for using such pairs in the architecture design phase, thus operationalizing the experience.

The paper is structured as follows: After an introductory part, the pros and cons of well-known techniques for persisting architectural experience are described in section 2. In section 3, the details of the approach are given. Our explanation is based on Fraunhofer ACES - the architecture design process [KK11] used for software architecture design at Fraunhofer IESE. Section 4 shows how we instantiated our idea for the actively evolving technology of mobile "apps" and particularly its application in the business domain. The "Initial Validation" chapter features a case study that was performed in order to take first steps towards the validation of the approach. The section "Conclusion" concludes the paper.

2. Related Work

The aim of this paper is to show the reader how new technology can be efficiently adopted in the architectural practices of software engineering companies. The main challenge in adopting a new technology is the lack of structured organization-wide architectural experience that can be reused in an operational manner. Thus, our goal is to find a way to collect and persist this experience inside the architecture design process.

Well-known processes such as ADD [BK01] or BAPO/CAFCR [Sm03] provide solid guidance for architecture design but, unfortunately, do not offer any facilities for collecting and persisting architectural experience for further reuse. This means that such facilities have to be either invented or created by combining prior works. We examined related work on existing solutions to the following challenges: ways to collect architectural experience and ways to persist it within the architecture design process. One of the prominent techniques for collecting experience is "Project Postmortems" [Ti90]. How this technique was used for our purpose will be explained in section 3.2. A popular approach for persisting architectural experience are domain-specific software architectures [Tr95]. Researchers have developed a body of various methodologies for

approaching such architectures, ranging from guidelines and recommendations for creating domain-specific architectures [AA07] to proposals of ready-to-use reference architectures [ZL00], [DN08]. Unfortunately, the applicability of these approaches is limited to particular well-scoped business domains, which is not our case. Our target is a technology that can have multiple application areas; therefore it would be too effort-consuming and inflexible to model it with a set of software components.

Alternative approach for technology-specific experience persistence are pattern languages, such as [BM00], [GH94], or the factor-strategy pairs of [HN99]. Although pattern languages are a very effective approach, we feel that they lack precision in describing architectural problems: In [HN99], the description of the factor (problem) is rather unstructured, and in the "Gang of Four" work [GH94], the problems are described in terms of object-oriented programming rather than architecture. In this paper, these drawbacks are mitigated with the help of architectural scenarios [KB94], [DF99].

It is always a challenge to find the right detail level for describing the solution part of a pattern. For example, the detail level of [GH94] patterns differs from ours: Unlike the implementation level used by [GH94], we describe patterns on the architectural level, which is more abstract and can lead to multiple implementations; hence, it is a complex task to describe a pattern in such a way that it gives the architect solid guidance without prescribing any particular implementation. Therefore we propose our own generic template for solution descriptions. The next section describes how these approaches were combined in order to mitigate the drawbacks of each one and develop the needed technique.

3. Approach

This section contains a step-by-step description of our approach for persisting technology-specific architectural experience in the architecture design process.

3.1 Baseline Architecture Method

As a basis for the technology-specific architecture design process, we take Fraunhofer ACES [KK11] – the architecture design process actively used by Fraunhofer IESE. The input to the process are system requirements obtained in previous phases of the software system engineering process, and the output is, among others, a set of documented architectural views of the system. ACES is a comprehensive process that guides architects in all their activities, starting from the identification of stakeholders and the understanding of architectural drivers via architecture realization all the way to the documentation and validation of the system architecture produced. The approach consists of two parts: *core competence* – the main part, which includes domain-independent sub-processes for architecture design, and *domain competencies* – an optional part, which contains domain-specific static experience artifacts related to the architecture design. The structure of Fraunhofer ACES is shown in Figure 1.

The “domain competencies” (in our case rather “technology competencies”) part of Fraunhofer ACES foresees three crucial experience collection areas - *challenges*, *solutions*, and *technologies*, which represent a similar concept as the factors and strategy pairs of the “Global Analysis” of [HN99]. Challenges correspond to factors and solutions correspond to strategies. The technologies area is used for persisting information about COTS solutions that are relevant for the considered domain. Solutions can address technologies in their description. This paper focuses on challenges and solutions in architecture design; therefore the technology part of ACES is not addressed here. ACES provides experience areas, but gives no concrete guidelines on how to collect this experience or how to represent it.

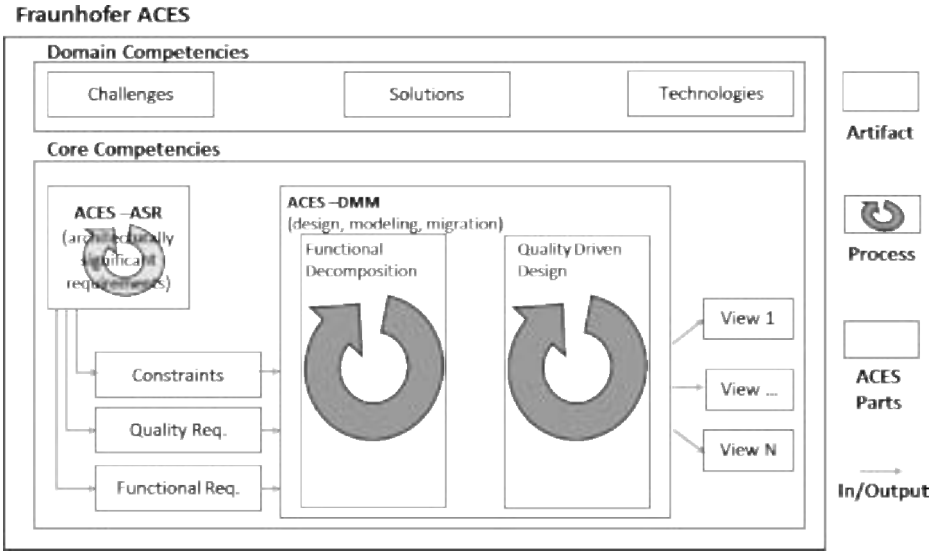


Figure 1: Fraunhofer ACES

In order to reuse architectural experience during the architecture design process, we examined two sub-processes of the Fraunhofer ACES core competencies: ASR [KK11] (Architecturally Significant Requirements) – the process analyzing stakeholders and eliciting requirements specifically related to the architecture, and DMM [KK11] (Design, Modeling, Migration) – the process for designing the actual system architecture based on the elicited requirements. In sections 3.2 – 3.5, we describe how one can fill out the “domain competencies” part of ACES with the necessary experience and then reuse it in ASR and DMM processes.

3.2 Eliciting Challenges and Solutions

The idea of collecting experiences by identifying challenges of a particular area is not new and has already been proposed by [DF99]. However, the problem is that [DF99] did not specify where to get these challenges from and how to describe them. Knowledge about challenges regarding a particular technology comes from experience; therefore, an

experience source is required. We consider the best experience source to be pilot projects implemented with the current technology. Therefore, several initial projects have to be performed during which the experience base is filled. After each project, a project postmortem [DD05], [Ti90] has to be performed in order to elicit tacit architectural experience residing in the mind of the architect, or in the artifacts of the project, such as documentation, code etc. Postmortems are techniques that allow for systematically examining the completed project for the purpose of eliciting pitfalls to avoid in the future, and best practices to be reused. We use postmortems for eliciting challenges that recur when applying a specific technology. We perform project postmortems in the form of interviews with architects. These interviews consist of the following steps:

1. *Find out challenging architectural scenarios*. Our experience in designing software architecture shows that most of the challenges are usually related to the quality (non-functional) requirements of the architecture. In Fraunhofer ACES [KK11], quality requirements are described in the form of architecture scenarios [DF99]. Thus, all that architects need to do during the post-mortem meeting is to point out architectural scenarios that in their minds do not only represent a unique challenge met in one project, but that reflect the crosscutting challenge of the current technology. More details on describing challenges with architectural scenarios can be found in section 3.3.

2. *Check for duplicates*. Check if this architectural scenario already exists in the experience base. If not, add it; then check if the related solution has been refined or improved in the current project. If yes, supplement the existing solution with the new details.

3. *Elicit and describe the solution*. An architect needs to elicit the solution for the identified challenge from the project documentation and describe it according to the solution pattern template given in section 3.4. A solution described with the solution pattern template is then added to the experience base.

4. *Retain traceability*. We trace relationship between architectural scenarios and solution patterns with the help of a traceability matrix; therefore this matrix has to be updated every time a change is made to the experience base.

The artifacts elicited in this phase are: a set of architectural challenges described using the architectural scenario template presented in section 3.3, a set of solution patterns described using the template presented in section 3.4, and a traceability matrix, which establishes the relationship between both.

3.3 Describing the Challenges with Architectural Scenarios

Quality requirements for software architectures are described in ACES with architectural scenarios [KB94]. According to [RW05] - "*An architectural scenario is a crisp, concise description of a situation that the system is likely to face, along with a definition of the response required of the system*". Initially, architectural scenarios were used as a tool for software architecture evaluation [KB94], but later this approach has also turned out to be

suitable for describing quality requirements for software architectures [DF99]. The template shown in Figure 2 can be used for describing architectural scenarios. In order to clarify this issue better, an example of an architectural scenario is given in section 4.

Scenario	*Name of scenario*
Quality	*Related quality attribute*
Environment	*Context applying to this scenario*
Stimulus	*The event or condition arising from this scenario*
Response	*The expected reaction of the system to the scenario event*
Response Measure	*The measurable effects showing if the scenario is fulfilled by the architecture*

Figure 2: Architectural scenario template

3.4 Describing Solutions

Following the idea of pattern languages [BM00], [GH94], we use "Solution Patterns" for the description of our solutions. The crucial point in describing solution patterns is the level of detail. On the one hand, the more details the solution contains, the better. On the other hand, each additional detail induces a certain assumption regarding the context in which the system is developed, which makes the solution less applicable. Developers of pattern languages use templates for their descriptions [GH94]. It is good to use strict templates when the context of an application is well known, as in the case of pattern languages for specific domains. In our case, we consider a whole technology that can be applied in various ways. The template for describing the solution must therefore be sufficiently detailed to be understandable and implementable, and at the same time remain applicable when used in different contexts. Therefore we keep our solution pattern template simple and rather base it on examples than on principles, resulting in more freedom during description and subsequent application. The template consists of the following parts:

1. *Textual description*. A clear description of the way the current pattern resolves the challenge described in the architectural scenario. The description must include pros, cons, restrictions, and known uses of the solution. Should the challenge be resolved by employing COTS components (frameworks, platforms etc.), a link to this component has to be specified.

2. *Structural Diagram*. A structural view of the system that supports the current architectural scenario.

3. *Behavioral Diagram*. An instance(s) of interaction among the components of the system, which supports the current architectural scenario(s).

3.5 Enriching the Process

Once the static experience artifacts (challenges and solutions) have been collected and described appropriately, they must be included in the architecture design process and

provide the users of the process with usage guide. How artifacts are included into Fraunhofer ACES and then reused is sketched in Figure 3.

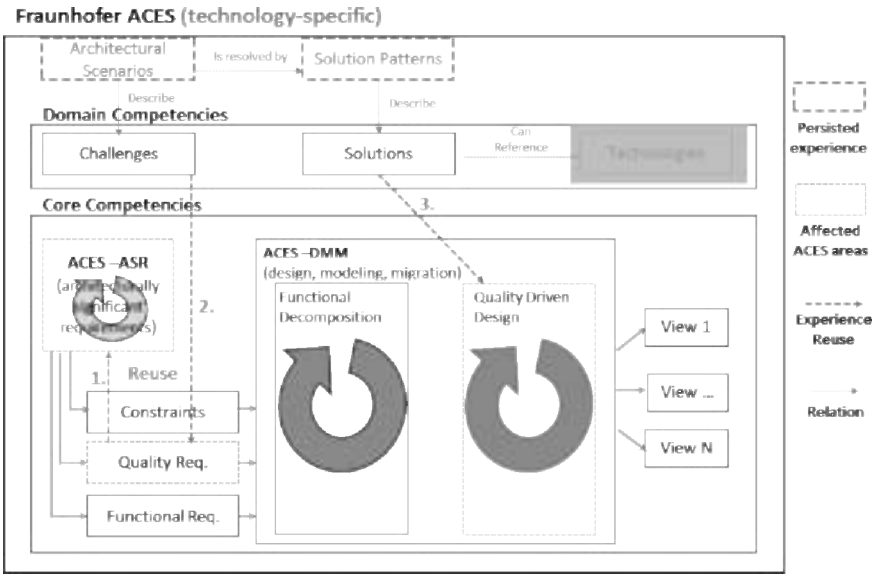


Figure 3: Technology-specific Fraunhofer ACES

We guide users with the help of reuse scenarios that describe how experience artifacts shall be applied. Three key reuse scenarios (the numbers of reuse scenarios can be matched to the numbers in Figure 3) that may occur when designing a software architecture with the technology-specific Fraunhofer ACES are:

1. **Using architectural scenarios for finding missing requirements.** There are certain quality requirements that are very likely to appear in the context of a particular technology. The customer might first overlook these requirements and come back to them only in a later phase of the project when any change is costly. Therefore, the architect will find it handy to use typical architectural scenarios of the technology as a checklist for finding missing requirements.

2. **Using architectural scenarios as input for the architecture design process.** Architectural scenarios accurately describe the challenge that needs to be resolved by the architecture of a particular software product. Ideally, the architecture scenario will be linked directly to the solution pattern (see point 3) that resolves it; otherwise, its precise description will help the architect to significantly narrow the solution search domain and will later on serve as a basis for assessing the selected solution.

3. **Applying solution patterns for quality driven design.** Based on the architectural scenarios to be satisfied, an architect selects solution patterns to be implemented. Using proven solution patterns will make the architecture design process more efficient and will increase the quality of the resulting product.

4. Application Example

In order to collect initial evidence regarding the effectiveness of our approach, we apply it to one of the technologies currently under research at Fraunhofer IESE. Due to the current trend towards ubiquitous computing and the growing need for mobile support for enterprises, we decided to choose mobile apps as our target technology and scope it to the business domain of the application (excluding, e.g., the gaming domain).

The challenges encountered here are mostly related to the fact that business-oriented mobile applications have the same quality requirements as common desktop applications, but run in a totally different environment. These applications suffer from Internet connectivity problems, high power dependency, and other challenges brought on, for example, by the specifics of the operating system or by a particular application store. Therefore it is absolutely necessary to be aware of the typical challenges of this technology and make sure they are covered with the system architecture.

4.1 Identified Challenges

After conducting several pilot projects at Fraunhofer IESE and performing their post-mortems (section 3.2), we identified a set of challenges related to mobile business applications. Some of these challenges are presented in the table in Figure 4. The second and third columns of the table represent the name of the challenge and architectural scenario that describes it. The first column classifies the challenges into challenge areas.

Challenge Area	Challenge (Quality Requirement)	Architectural Scenario
Unreliable Connectivity	Application must operate with bad internet connection.	Seamless Connectivity
	Application must operate when there is no internet connection .	Connection Loss Tolerance
	Remote communication must be fast.	Reduced Network Latency
	Remote communication must be reliable.	Consistent Communication
Limited Energy Supply	Application must be energy efficient.	Reduced Power Consumption
Deployment	It must be possible to deploy application update within one hour	Rapid Application Deployment

Figure 4: Challenges of Mobile Business Applications

One prominent challenge for mobile apps is deployment. In contrast to desktop or web applications, the standard facility for deploying apps is an “appstore”. An “appstore” is completely proprietary entity and lies beyond the developers’ control: There is no guarantee that the application will pass the internal approval process and that deployment will be allowed. Furthermore, the duration of the approval process cannot be foreseen and deployment time can therefore not be guaranteed to the customer. Controlled deployment is clearly a crucial requirement for the mobile application development organization, which is legally obliged to guarantee software defect removal within a certain timeframe. The architectural scenario “Rapid Application Deployment” in Figure 5 describes a concrete instance of a controlled deployment challenge. Having such precise description allows an architect to easily evaluate the suitability of the proposed solution by simply playing the scenario.

Scenario	*Rapid application deployment*
Quality	*Deployment*
Environment	*Mobile application is deployed on the user's device*
Stimulus	*Customer discovers a defect or requests a feature which requires rapid application modification and re-deployment*
Response	*It is possible to deploy the new version of the application on the user's device within one hour*
Response Measure	*Deployment time is less than one hour (60 minutes)*

Figure 5: Example of an Architectural Scenario ("Rapid Application Deployment")

4.2 Identified Solutions

Based on the experience obtained during the pilot projects, we were also able to find appropriate solution patterns for the identified challenges. In Figure 6, the reader can find a traceability matrix that establishes the relationship between solution patterns and the architectural scenarios they resolve. It can be clearly seen that one pattern can resolve various scenarios and one scenario can be resolved by multiple patterns.

Scenario/Pattern, Technology	Caching	Offline Mode	Reduced Payloads	Transactional Messaging	Message Queuing	Independent Lightweight Data Model	Deployment bypassing appstore
Seamless Connectivity	x			x	x		
Connection Loss Tolerance	x	x					
Reduced Network Latency	x		x		x	x	
Consistent Communication				x			
Reduced Power Consumption					x		
Rapid Application Deployment							x

Figure 6: Identified Solution Patterns

In Figure 7, Figure 8, and Figure 9, an example of a solution pattern described according to the template given in section 3.4 is given. This pattern is named "Deployment bypassing appstore" and resolves the architectural scenario given in Figure 5.

General Information:
In order to avoid problems with deployment through the appstore, a company has to establish a private app distribution server. The server contains the repository where apps reside, and takes care of app distribution through mobile clients. The mobile client is an app which resides on the mobile device of the company worker (user). This app has access to the repository and can download and install custom apps on the user's device without app store involvement. Furthermore, the app listens for push notifications from the server, which inform the user about recent app updates.

Pros:
- Time independence from appstore
- No app evaluation process at appstore
- Possibility to use deploy functionality prohibited by app stores

Cons:
- If required, billing concept has to be developed from scratch

Restrictions:
- iPhone: device must be jail broken
- Android: security exception has to be added

Known Uses:
- "MobileIron Virtual Smartphone Management Platform" [Mi11] implements similar concept

Figure 7: Textual Description ("Deployment bypassing appstore")

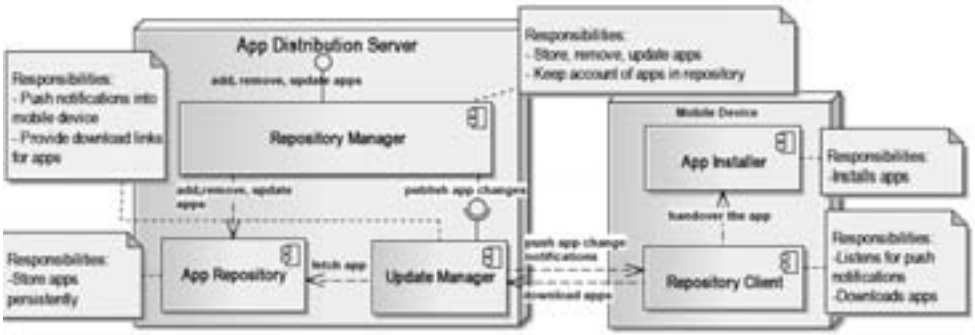


Figure 8: Structural Diagram ("Deployment bypassing appstore")

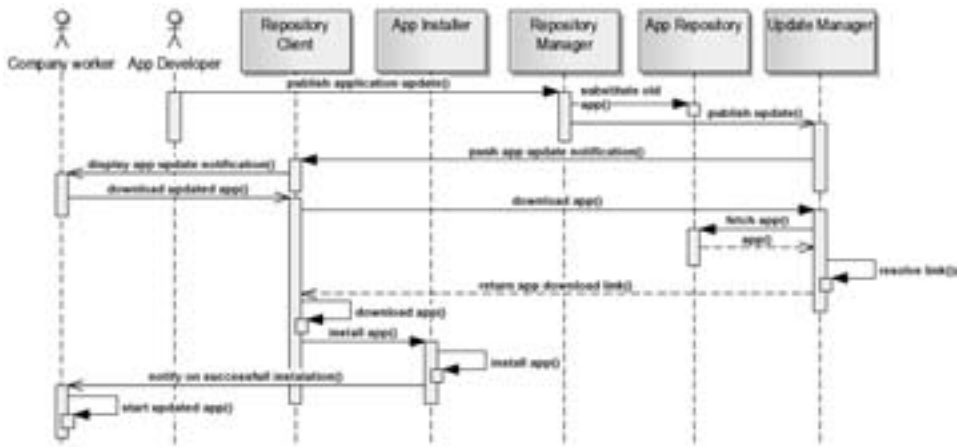


Figure 9: Behavioral Diagram (Architectural Scenario: "Rapid Application Deployment")

4.3 Initial Validation

This section features an initial validation of our assumption regarding the improved efficiency of the architecture design process and better quality of the end product due to the reuse of technology-specific experience during the architecture design process. Both points are hard to assess without a real project setting. However, a coarse validation of the efficiency aspect can be carried out with the help of a case study.

The case study was designed as follows: We took an architecture document of one of the mobile business applications developed at our institute, designed a change request for it, and asked three Fraunhofer IESE employees with different levels of knowledge in software engineering and only general knowledge in mobile systems to perform several tasks related to this change request with the help of the mobile technology experience persisted in Fraunhofer ACES. The experience base included a catalog with 21 architectural scenarios and 22 solution patterns (partially shown in Figure 6) identified at Fraunhofer IESE using the approach described in section 3. The participants had to

imagine having to implement the change request and thus had to redesign some of the system structures accordingly. They were asked to record the time they spent on:

1. Finding the relevant architectural scenario and solution pattern in the catalog;

2. Reading the pattern, understanding how they would apply it for the given system architecture, and sketching modified structural and behavioral views of the system.

According to the results, the participants reusing experience artifacts needed five minutes on average to find the matching architectural scenario and solution pattern in the catalog. In order to understand the application of the chosen pattern to the current system architecture and sketch modified views, the participants needed 18 minutes on average.

The participants were asked to compare their experiences using the extended ACES approach with using the original approach. Their assessment was that given Fraunhofer ACES without technology-specific experience artifacts, they would require 2.5 hours on average to come up with their own solution for the given problem. The use of the extended approach thus reduced the time spent to less than 20%.

However, these results are based only on one case and on the subjective judgment of the case study participants. Obviously, a more thorough validation needs to be done regarding the proposed approach in order to enable drawing more concrete conclusions about its efficiency. It will be necessary to perform similar case studies or full-size experiments while varying factors such as number of participants, level of participants' experience, size of the catalog, experiment context, and type of tasks for the participants. Also, ways to validate the quality aspect of the resulting product have to be found.

5. Conclusion

In this paper, we have shown a way to collect and persist technology-specific architectural experience in an organization. Reuse of this experience during follow-up projects is supposed to increase the efficiency of the software architecture design process and the quality of the resulting software product. Furthermore, stored experience will lower an organization's dependence on human resources.

We described how technology-specific experience can be persisted and reused beneficially within the architecture design process. For experience persistence we used challenge-solution pairs. We described a method for collecting these pairs using project postmortems and gave templates for their precise description with architectural scenarios and solution patterns. Finally, we gave an example of a typical challenge-solution pair for the technology of mobile apps and described it using given templates. A case study served as coarse validation and allowed us to draw first conclusions regarding the efficiency of a software architecture design process supplemented with technology-specific experience artifacts. Although this case study showed a noticeable increase in

efficiency, it is too early to draw final conclusions in this regard. A more thorough validation of the approach has to be performed.

References

[AA07] Eduardo Santana de Almeida, Alexandre Alvaro, Vinicius Cardoso Garcia, Leandro Nascimento, Silvio Lemos Meira, Daniel Lucredio: Designing Domain-Specific Software Architecture (DSSA): Towards a New Approach. WICSA'07 (2007)

[BK01] Len Bass, Mark Klein, and Felix Bachmann, "Quality Attribute Design Primitives and the Attribute Driven Design Method," the 4th International Workshop on Product Family Engineering. Bilbao, Spain, 3-5 October, 2001.

[DD05] Kevin C. Desouza, Torgeir Dingsøyr, Yukika Awazu: Experiences with Conducting Project Postmortems: Reports vs. Stories and Practitioner Perspective. Proceedings of the 38th Hawaii International Conference on System Sciences (2005)

[DF99] Jean-Marc DeBaud, Oliver Flege, Peter Knauber: PULSE-DSSA - A Method for the Development of Software Reference Architectures. Internal publication: Fraunhofer Institute for Experimental Software Engineering (IESE); 1999

[DN08] Serhan Dagtas, Yuri Natchetoi, Huaigu Wu, Louenas Hamdi: An Integrated Lightweight Software Architecture for Mobile Business Applications. Seventh Working IEEE/IFIP Conference on Software Architecture (2008)

[GH94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Massachusetts, 1994.

[HN99] Christine Hofmeister , Robert Nord , Dilip Soni. Applied Software Architecture 1999, ISBN-10: 0-201-32571-3

[KB94] Kazman, R. ; Bass, L. ; Abowd, G. ; Webb, M. SAAM: a method for analyzing the properties of software architectures. Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on.

[KK11] Thorsten Keuler, Jens Knodel, Matthias Naab; Architecture-Centric Software and Systems Engineering; 2011; White Paper; Fraunhofer Publica

[Mi11] MobileIron Virtual Smartphone Management Platform; http://www.mobileiron.com/en/smartphone-management-products/smartphone-management-for-enterprise-mobility

[RW05] N. Rozanski, E. Woods. Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. 1995. ISBN: 978-0-3217-1833-4

[Sm03] Jürgen K. Smith “The Building Block Method”, Philips Research Laboratories, Eindhoven. ISBN 90-74445-58-6; 2003

[Ti90] Mark J. Tiedeman: Post-Mortems-Methodology and Experiences; IEEE journal on selected areas in communications, vol. 8. no. 2 . February 1990

[Tr95] W. Tracz, Domain-Specific Software Architecture (DSSA) Pedagogical Example, *ACM SIGSOFT Software Engineering Notes*, Vol. 20, No. 03, July, 1995, pp. 49-62.

[ZL00] Deming Zhong, Bin Liu, Lian Ruan: The Domain-Specific Software Architecture of Avionics Testing System. 24th Digital Avionics Systems Conference (2000)

Controlled Generation of Models with Defined Properties

Pit Pietsch, Hamed Shariat Yazdi, Udo Kelter

Software Engineering Group
University of Siegen
pietsch, shariatyazdi, kelter@informatik.uni-siegen.de

Abstract: Test models are required to evaluate and benchmark algorithms and tools which support model driven development. In many cases, test models are not readily available from real projects and they must be generated. Using existing model generators leads to test models of poor quality because they randomly apply graph operations on graph representations of models. Some approaches do not even guarantee the basic syntactic correctness of the created models. This paper presents the SiDiff Model Generator, which can generate models and model histories and which can modify existing models. The resulting models are syntactically correct, contain complex structures, and have specified statistical properties, e.g. the frequencies of model element types.

1 Introduction

Model-Driven Development (MDD) has become a central development paradigm in many application domains. As a consequence, new tools and methods specifically tailored for MDD are being developed and need to be evaluated with regard to aspects like quality, efficiency and scalability. Examples of such tools are model transformation tools, testing, validation and verification tools, model repositories, and difference and evolution analysis tools.

Finding real test models for evaluation purposes is difficult because in many domains real models are only scarcely available. Even when models from real projects are available their properties are often unknown, or they are not adequate for a particular testing purpose. Different types of tools and MDD methods require significantly different test models. Model comparison and history analysis algorithms need pairs, or sequences, of models, where the evolution is precisely documented. Other algorithms just need very large "realistic" models for scalability tests. As a solution to this challenge we present the *SiDiff Model Generator* (SMG) [PSYK11].

The SMG can create or modify arbitrary model instances complying to EMF Ecore based meta models. It has been developed in the context of the ***SiDiff*** system (s. [KWN05, TBWK07]), a highly configurable framework for model comparison. While originally intended for creating synthetic benchmarks for model comparison algorithms, the SMG evolved into a highly configurable tool for test model generation.

The paper is structured as follows. Section 2 discusses in greater detail the requirements

on a highly versatile model generator. Section 3 presents an overview of the SiDiff Model Generator and explains the main steps in the process of generating synthetic models. The Stochastic Controller, which is responsible for implementing statistical properties of models, is explained in greater detail in Section 4. In Section 5 the runtime of the SMG and the quality of the produced models are discussed. Section 6 discusses other approaches to generate test models and the paper closes in Section 7 with a summary and an outlook.

2 Background and Requirements

Before discussing the requirements on a model generator some definitions that are used throughout the paper are introduced.

We assume that models are typed and that each model type has a set of *edit operations* which can be used to alter a model of this type. For example, state machines should have an edit operation `createState(name)` which creates a new state with a given name. An *Invocation of an Edit Operation* is an edit operation with concrete arguments.

An *asymmetric difference*[1] (or *Patch*) is a sequence of invocations of edit operations. It can be regarded as a linear sequential program, each statement being an edit operation. A difference can be *applied* to, or *executed* on, a model which is called the *base model* in this context. By executing the sequence of invocations on the base model, a new model is created which can be regarded as a successor version.

Invocations of edit operations can refer to existing model elements and/or positions of elements in a given model; this is one reason why an error can occur when an invocation is executed on a model. A difference is *applicable* on a model if no error occurs when the sequence of invocations is executed on this model.

Main Usage Scenarios. A model generators should be able to handle three main use cases: (a) *creating new models* from scratch, (b) *modifying existing models*, i.e. create a difference which is applicable to a given model, and apply this difference, and (c) *creating model histories*. One might think that (c) is just the repeated application of (b) where the last created model is the base model for the next iteration. However, this is not the case: A model history in this context has properties that are connected to multiple revisions, e.g. the life time of model elements.

Correctness of Generated Models. The generated models should be correct according to their complete meta model, i.e. not only conform to the basic abstract syntax, but also to additional constraints which are typically specified as OCL expressions.

Properties of Generated Differences. One should be able to specify properties of the generated models and differences, for example the *size* of the difference(s), i.e. the number of contained operation invocations. Another important property is the *frequency* of edit operations which occur in the generated differences.

[1] There are also symmetric differences, but they are not relevant in this paper.

A model generator should support two interpretations of frequencies: literal and stochastic. With *literal interpretation*, the generated differences will contain exactly the specified number of edit operations. The literal interpretation imposes restrictions on the specified values for the size of the difference and the frequencies: they must be nonnegative integers.

With *stochastic interpretation*, the specified frequencies of edit operations are interpreted as the probabilities of each operation. The generated differences will contain only approximately the specified frequencies. Stochastic interpretation is useful when large models are required for random testing and when the set of edit operations is large and the distribution of frequencies is skewed.

Complex Edit Operations. A model generator should be able to create all correct models, including those containing complicated structures which are not accidentally created by elementary operations. As a consequence, a model generator should offer means to define such complex non-atomic edit operations.

All existing approaches to generate models do not meet several of the above requirements, s. Section 6.

3 The SiDiff Model Generator

Figure 1 gives an overview of the four different main components of the SiDiff Model Generator (SMG), as well as the input and output documents. The meta model and the operation set are dependent on the model type. Further OCL constraints can be specified to disallow unwanted properties in the output model.

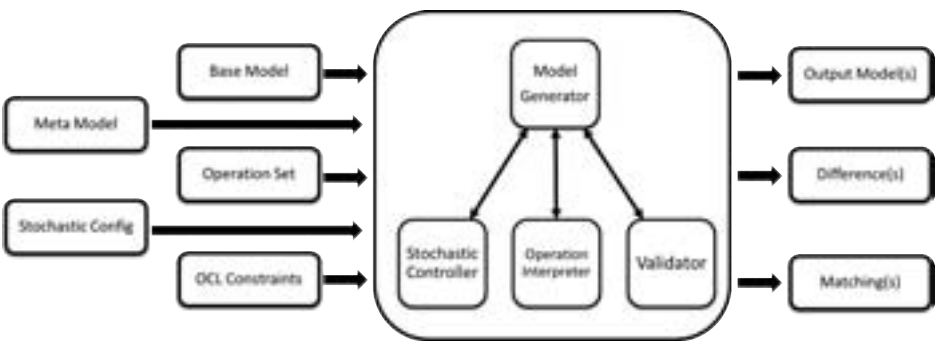


Figure 1: The SiDiff Model Generator - Overview

The *Operation Interpreter* applies selected edit operations to the base model which is provided as input. The *Stochastic Controller* (SC) is configured by an XML-file and ensures that operations are generated in a controlled manner. The *Validator* checks the correctness of the modified model. The *Model Generator* is the central communication interface between the different components.

For each pair of a base and an output model two additional documents are generated: the difference, i.e. the applied operations, and a matching, i.e. a table with corresponding elements of the two models.

The SMG generates only differences which are applicable to the base model without errors. If such a difference cannot be created the whole creation process fails and no output (other than error messages) is created. Any generated differences are, of course, not guaranteed to a applicable to other models which differ from the specified base model.

The SMG covers the all three main usage scenarios mentioned above. Use case (b) is implemented by repeatedly generating an operation invocation which is applicable to the current state of the model being constructed. Use case (a), creating a new model, is implemented by generating a difference which is applicable to an empty model. Use case (c), creating a history, is implemented by creating one large difference which is partitioned into smaller, consecutive parts. In all three cases, the primary product of the SMG is a difference; the new or modified model is a by-product, although this may be the most important result from a user's point of view. The two different interpretation modes for frequencies of operations always refer to this difference; they are further refined as follows.

Literal Interpretation Mode: Here the user specifies exactly the number of invocations for each edit operation. For example, if the edit operation *createClass* is specified to be executed five times, the difference will contain five invocations of *createClass* and the modified model will have five classes more than the base model. The configuration is regarded as a literal specification of both the number and the kind of invoked edit operations, thus allowing the user to define quantitative properties of the generated models in great detail. All other aspects, e.g. where an operation is applied, are still selected with the help of the Stochastic Controller. This mode is, for instance, suited to recreate an observed modification process, e.g. based on data gathered from real model repositories.

Stochastic Interpretation Mode: In the stochastic interpretation mode *Probability Mass Functions* (PMF) of edit operations are used to configure the SMG. In this mode, the frequencies of edit operations are interpreted as probabilities. In case the specified probabilities do not sum up to 100%, the SC internally normalizes them by dividing them to the sum. The size of the difference has to be explicitly specified (whereas in literal interpretation mode, the size is implicitly defined by the sum of the number of edit operation invocations). The behavior of this mode is highly dependent on the size of the difference and on the shape of the PMFs: the generated difference will only approximately exhibit these frequencies. For skewed distributions with heavy tails a higher number of invocations will give better approximations. The stochastic interpretation mode can be used, for example, to create differences of varying size, but similar properties, with very low specification effort.

3.1 Model Modification Process and Index Maps

The model modification process has several consecutive steps: (1) A ContextType, i.e. a meta class which represents a model element type from the given meta model, is selected.

(2) A concrete instance of this meta class and the edit operation are selected. (3) The parameters required for the execution of the operation are selected or created. (4) The created operation is executed.

In the described model modification process situations may arise where it is not possible to complete the creation of an operation: For example, after the selection of ContextType, Context and edit operation, a parameter value which is a reference to a model element might be required, but no suitable model element exists in the model. To avoid expensive backtracking algorithms in these cases, we devised the concept of index maps: Based on the meta model, the operation set and the base model a set of maps is initialized. These maps essentially preserve the knowledge which operations can be successfully created at the current state of the model modification process. In each of the four selection steps mentioned above, the Stochastic Controller filters choices for which it would be impossible to create an applicable operation. The maps are automatically updated after each successful creation of an operation invocation.

The SMG comes with a custom tailored operation model (for more details see [PSYK11]) that allows users to specify simple operations, i.e. creating, deleting, moving and editing attribute values of single model elements, as well as to define complex operations that consist of two or more simple ones.

The SMG is open to the integration of other model transformation engines integrated with EMF Refactor[2]. However, the index maps mentioned above are not supported when EMF Refactor is used since they are tightly coupled to the operation model of the SMG.

4 Stochastic Controller

One very important requirement for a model generator is that the generated differences resemble real, user driven modification processes as much as possible. Common examples of such modification processes may be adding new model elements in close proximity to each other to resemble the implementation of a new subsystem or applying few scattered changes mimicking a debug process. As briefly discussed in Section 3, the *Stochastic Controller* (SC) is responsible for the controlled stochastic selection and application of edit operations to a given base model. In this section we first introduce qualitative properties of models. After this other concepts used in the SC, such as Decision Tables and Selection Policies, are discussed in detail.

4.1 Qualitative Properties of Differences and Fitness Values

In addition to the measurable properties of differences, i.e. size and frequency of edit operations (s. Section 2), qualitative properties affecting the resulting models exist, too. As mentioned in Section 3.1, the invocation of an edit operation consists of four selec-

[2] http://www.eclipse.org/modeling/emft/refactor/

tion steps. The properties of an individual that are used in selection processes are called *effective properties*. These properties mainly depend on the model type and the application domain. For instance, effective properties for the selection of UML classes could be the metric *Number of Methods* or the count of applied modifications. Because effective properties are often not precisely known and complex correlations between them exist, it is necessary to discuss two cases: When enough knowledge of effective properties is available and when they are mostly unknown. For both cases solutions are outlined after the concept of Fitness Values is introduced.

A *Fitness Value* (FV) is a value that describes the aptness of an individual for a definite goal, i.e. selection in our case. For instance, the number of modifications already applied to a model element can be used as a FV to decide whether it is more or less likely that the element in question is selected for further modification. Generally any function that maps all (or some) of the effective properties of individuals to their selection likelihood can be considered as a generator of FVs and we may obtain or construct a cumulative distribution function (CDF) based on them. The case that the range of FVs is finite is a special case in which we may obtain the PMF of individuals by calculating their frequencies. All model elements are annotated with their FVs. Additionally, FVs of individuals can be kept fixed during the model modification process or they can be updated after each successful execution of an edit operation (s. Section 4.2). Since the selection process is altered by this decision, it has a direct effect on the quality of the produced models.

For example, when benchmarking model comparison algorithms, FVs are used to explicitly prevent the application of redundant operations, e.g. renaming an element twice, or operations that cancel each other out, e.g. deleting a newly created element. Since such differences cannot be traced by model comparison algorithms, they would skew the evaluation results.

When the effective properties for selection and their corresponding PMFs are known the SC can simply be adjusted accordingly. In cases where such data is not available, a domain expert still has various options to configure the SC to her best knowledge. One solution is to perform the selection process randomly based on an uniform distribution. Because this is an oversimplification in most cases, more complex methods based on FVs, called *Selection Policies*, were devised. These policies give users the ability to control the selection process quite precisely (s. Section 4.2).

In the case of creating a model history additional effective properties connected to the life cycle of model elements exist, which cannot be considered when simply applying n consecutive differences to a base model. One example of effective properties connected to a history of models are the number of modifications a model element undergoes through its life cycle and the distributions of these modifications over different revisions of the history. Another example are different life time expectations of model elements based on their type. The SC is also capable of handling such complex behaviors.

4.2 Decision Tables and Selection Policies

Suppose that we are going to model the evolution of a given model M to M'. A *Decision Table* (DT) is, roughly speaking, a mapping that tells the system how to modify M in order to eventuate in M'. This concept can be extended to handle model histories: Let m be the number of revisions and $1 \leq i < m$, DT_i is a map between the model in revision i and its successor revision $i + 1$. DT_is are highly configurable and are defined in a configuration file for the system.

Selection Policies (SP) are statistical tools that can be used in the DTs. They select individuals in a controlled stochastic manner based on FVs.

Roughly each DT_i contains the followings:

(C-I) A list of ContextTypes each accompanied with predefined probabilities of selection as well as a SP for selecting one of them.

(C-II) SPs that choose a Context (instance) for each ContextType.

(C-III) A list of defined Operations on each ContextType that are also accompanied with selection probabilities as well as a SP for selecting one operation out of the list. These probabilities can be given from the observed frequencies of edit operations or from PMFs of edit operations. They are only effective in the stochastic interpretation mode.

(C-IV) Each Operation is additionally accompanied with the Number of Frequency which states how often each one will be applied to the model. This configuration data is only relevant when the literal interpretation mode is used.

(C-V) SPs for selecting Parameter values.

Currently five different types of SPs known from Genetic Algorithms [Mic96, Poh06, RR02, SD08] are implemented in the SMG. From now on we suppose that the set of FVs is bounded and their values are nonnegative. This will be a consistent assumption throughout our application scenarios.

As its name implies, the *Random Selection Policy* (RSP) is the obvious case of an uniform random selection method. In *Roulette Wheel Selection Policy* (RWSP) the selection probability of an element is proportional to its FV. This causes better fitted elements to have a higher chance of being selected. If the FVs of selected elements are increased after each operation execution there is a risk that the most fitted elements eventually overwhelm the selection procedure. To avoid this situation, elements can be sorted according to their FVs in a non-decreasing order and the indexes can be used as new FVs. This selection policy is called *Simple Ranking Selection Policy* (SRSP). When there are many individuals, the ones with the lowest FVs will only have a very low chance of being selected. To address this shortcoming either the *Linear Ranking Selection Policy* (LRSP) or the *Non-Linear Ranking Selection Policy* (NLRSP) can be used. Both work based on the extended concept of the SRSP and the rankings are done in a configurable way. Let n be the number of

individuals participating in the selection process, and also let the individuals be already sorted according to their FVs in non-decreasing form, then according to LRSP the new FV for the i th individual is defined as:

$$r_l(i,n) = \frac{1}{n}\big(\alpha + 2\,(1-\alpha)\frac{i-1}{n-1}\big)$$

where $1 \leq i \leq n$ and $\alpha \in [0,2], \alpha \in \mathbb{R}$ is the selection pressure. When α is less than 1 the slope is positive, if $\alpha = 1$ then the line will be horizontal causing all individuals to have equal probability for being selected and when α is greater than 1 we have a negative slope in which most fitted individuals have a lower chance of selection (s. Figure 2).

With previous assumptions, the NLRSP can be obtained by:

$$r_{nl}(i,n) = \frac{n\,x^{i-1}}{\sum_{j=1}^{n} x^{j-1}}$$

where x is the positive root of the following equation which can be solved using Bisection or Newton-Raphson methods [BF00], [KK08]:

$$(\beta - n)\,x^{n-1} + \beta\,x^{n-2} + \cdots + \beta\,x + \beta = 0$$

In the above equation, $\beta \in [1, n-2]$ is the selection pressure and we suppose that $n \geq 3$. By the linear transform $\beta = (n-3)\,\alpha + 1$ where $\alpha \in [0,1], \alpha \in \mathbb{R}$, then we have a suitable handy representation of NLRSP in which α will be the selection pressure. When α tends to zero the slope of the line will tend to 0 as well so we will have a uniform random selection. When α tends to 1 then the selection pressure increases in a non-linear manner (s. Figure 2). Additionally, due to non-linear characteristic of the NLRSP, when the number of participating individuals are big, even moderate selection pressures have strong impacts, i.e. the very few of individuals at the end of the sorted list will get most of

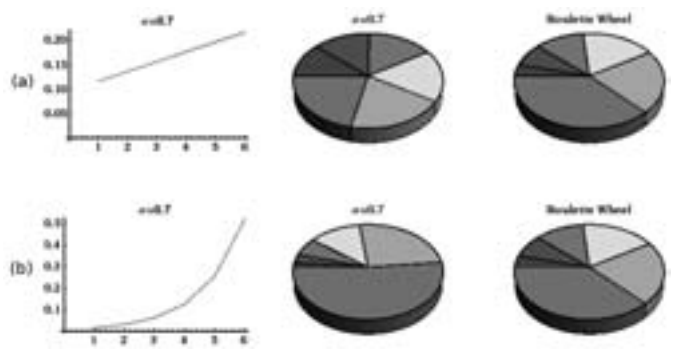


Figure 2: (a) LRSP vs RWSP (b) NLRSP vs RWSP. For a sample of $n = 6$ individuals and selection pressure $\alpha = 0.7$, X-Axis shows the individuals and Y-Axis shows the selection probabilities. Pie charts show the corresponding chance of selection in these two methods.

selection probabilities. Therefore in such situations smaller selection pressures might be more favorable.

Since during the modification process the size and structure of a model is changed dynamically, the SPs can be used to control this process in a subtle way and they have qualitative effects on the results (s. Section 4.1). A small part of a simplified conceptual decision table[3] in the XML format is shown here:

```
<DT>
    <SPCT Name="RWSP" UpdateMode="Dynamic" />
    <CT Name="UMLPackage" p̂="20">
        <SPC Name="NLRSP" α ="0.7" />
        <SPO Name="RWSP" UpdateMode="Fix" />
        <O Name="createClass" p̂="20" f̃="3" />
        <O Name="createInterface" p̂="1" f̃="2" />
    ⋮
</DT>
```

Considering the DTs specifications (s. 4.2), the above XML file is interpreted as: C-I corresponds to the <**SPCT**> tag as well as <**CT**> tags. Each <**CT**> tag contains a <**SPC**> (C-II) and a <**SPO**> (C-III) as well as at least one <**O**> (C-III).

Selection probabilities ($\hat{p}$ values), which are defined in C-I and C-III, can also be used as FVs for SPs since they have the required characteristics. To make things more organized, when SP=RWSP then the values of $\hat{p}$ are used as they are for selecting the individuals (they are automatically normalized), when SP=SRSP, LRSP, NLRSP these values are used as FVs and the selection is done based on the specified SP (in this case normalization is not necessary).

The parameter *UpdateMode* decides whether or not the effective FVs in the selection process are dynamically updated after each successful edit operation execution or stay fixed through the whole the modification process. In the end, the $\tilde{f}$ values are frequencies that are used in the literal interpretation mode (C-IV).

5 Evaluation

To evaluate the SMG the runtime for creating and modifying models was measured. The results of five executions were averaged and show a good performance of the tool. Using a configuration file containing just create edit operations, 4 models (class diagrams) with the sizes of 100, 1000, 5000 and 10000 were first created and then modified. The execution times on an MacBook Pro 2010 with an 2.66 GHz Intel Core i7 processor and 4 GB RAM are shown in Table 1.

In order to evaluate the runtime for modifications of models, a configuration that randomly

[3] In this DT we use the following abbreviation: SP=Selection Policy, CT=ContextType, SPCT=SP for ContextType, SPC=SP for Contexts, SPO=SP for Operations, O=Operation, $\hat{p}$=Selection Probabilities (PMFs), $\tilde{f}$=Frequency of an Edit Operation, α=Selection Pressure for a SP.

No. Elements in the base model (NEBM)		100	1000	5000	10000
Creation times		0.03s	0.52s	10.77s	55.47s
Modification times, with the No. Edit Opr. in % of NEBM	25%	0.02s	0.36s	9.43s	50.31s
	50%	0.03s	0.70s	19.95s	117.21s
	75%	0.02s	1.04s	35.76s	249.59s

Table 1: Runtime for creation and modification of models. The times do not include the times for loading and serialization of the models.

selects and executes edit operations was used. The size of the difference was chosen as 25%, 50%, and 75% of the number of elements in the base model. For example, for the model with 1000 elements and the case of 50%, 500 edit operations were created and executed. Obviously, the runtimes required by the SMG are good and acceptable in practice.

Edit Operation	Spec.freq.	100	1000	5000	10000
create Package	0.64%	3.33%	0.42%	0.84%	0.50%
create Class	4.20%	5.56%	3.85%	4.27%	4.19%
create Interface	0.60%	0.00%	0.73%	0.49%	0.59%
create Attribute	11.50%	12.22%	12.50%	11.64%	11.25%
create Method	28.83%	11.11%	25.46%	27.96%	29.54%
create Parameter	51.83%	62.22%	55.00%	51.97%	51.68%
create Association	2.40%	5.56 %	2.08%	2.83%	2.25%

Table 2: Observed frequencies of elements in the created models vs the specified frequencies.

We also checked how well specified frequencies of operations are actually implemented in the newly generated models for one of the five executions. The frequencies used in our evaluation are shown in column "Spec.freq." in Table 2; they were observed on the software repository of a real project[4].

Not surprisingly, actual frequencies in small differences can diverge substantially from the specified frequencies. Generally such behaviors are intrinsic to probabilities and statistics due to their stochastic nature. Specially in this case, it is caused by the uneven skewed shape of the specified distribution. The actual frequencies of larger difference tend to show better approximations. In our example with the size of 1000 and above our observed frequencies are reasonably close to the specified ones.

6 Related Work

Brottier et al. [BFS+06] present a generator for models, which is used in the context of model transformation testing. The input of the tool consists of an arbitrary meta model and a set of fragments, which are manually or automatically created object structures of inter-

[4] the ASM project, s. http://asm.ow2.org/.

est. [BFS+06] does not discuss how correct fragments are created. The algorithm creates a model by randomly choosing and connecting fragments. None of our requirements is fully met by this approach.

Mougenot et al. [MDBS09] propose a generator specifically aimed at creating large model instances. The algorithm works on a tree representation of models. Therefore the meta model of the model type must be transformed into a tree specification; however, this transformation does not preserve all information. The generation process randomly performs tree edit operations with a uniform distribution. An adaption of the algorithm to realistic statistical distributions is labeled as work in progress. None of our requirements is met by this approach.

Ehrig et al. [EKTW06] use graph grammars to systematically generate instances of arbitrary meta models. The set of rules is organized in three layers: Layer 1 rules create instances of meta model classes, Layer 2 and 3 rules establish relationships between created elements. The rules are automatically deduced from the meta model and applied randomly. This approach meets our correctness requirement, but cannot control properties of the generated models.

The Ecore Mutator developed in the AMOR[5] project is a tool which can modify Ecore-based models. The generator offers a set basic of operations, called mutations, to modify a given base model; additional operations can be implemented if necessary. The operations are performed randomly; the created output model is not checked for validity. None of our requirements are fully met by this approach.

To sum up, the main weakness of existing tools for test model generation is that the resulting models might be syntactically incorrect, might not contain typical constructs, and might have characteristics which differ substantially from realistic models or sets of models.

7 Summary

The SiDiff Model Generator presented in this paper is a versatile tool for creating test models for various purposes and in various usage scenarios. It overcomes several limitations of other current state-of-the-art approaches for test model generation. Most notably, it creates only correct models, it can modify existing models, and it enables users to control the size and many other properties of the created models and differences.

The SMG is generic in the sense that it can create models of arbitrary type. The primary requirement to support a new model type is a meta model and an associated set of edit operation, most of which can be generated from the meta model. Complex operations such as refactorings can be added manually with limited effort.

The SMG is being used in our own projects, a large set of generated models is available from our web site[6].

[5] http://www.modelversioning.org/

[6] http://pi.informatik.uni-siegen.de/qudimo/smg

Acknowledgments

This work was supported by Deutsche Forschungsgemeinschaft under grant KE499/5-1. The authors would like to thank Tim Sollbach and Michaela Rindt as student members of the SMG development team for their help.

References

[BF00] Richard L. Burden and J. Douglas Faires. Numerical Analysis. Brooks Cole, 7th edition, 2000.

[BFS+06] Erwan Brottier, Franck Fleurey, Jim Steel, Benoit Baudry, and Yves Le Traon. Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool. In Proceedings of ISSRE'06, Raleigh, NC, USA, 2006.

[EKTW06] Karsten Ehrig, Jochen Malte Küster, Gabriele Taentzer, and Jessica Winkelmann. Generating Instance Models from Meta Models. In FMOODS, pages 156–170, 2006.

[KK08] Autar Kaw and Egwu Eric Kalu. Numerical Methods with Applications. 1st edition, 2008.

[KWN05] Udo Kelter, Juergen Wehren, and Joerg Niere. A Generic Difference Algorithm for UML Models. In Software Engineering 2005. Fachtagung des GI-Fachbereichs Softwaretechnik, 2005.

[MDBS09] Alix Mougenot, Alexis Darrasse, Xavier Blanc, and Michèle Soria. Uniform Random Generation of Huge Metamodel Instances. In Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications, ECMDA-FA '09, pages 130–145, Berlin, Heidelberg, 2009. Springer-Verlag.

[Mic96] Zbigniew Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, 3 edition, 1996.

[Poh06] Hartmut Pohlheim. Geatbx: Genetic and evolutionary algorithm toolbox for use with matlab documentation, version 3.80 edition, December 2006.

[PSYK11] Pit Pietsch, Hamed Shariat Yazdi, and Udo Kelter. Generating Realistic Test Models for Model Processing Tools. In Proceedings of the 26th IEEE and ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KA, USA, Nov 2011.

[RR02] Colin R. Reeves and Jonathan E. Rowe. Genetic Algorithms Principles and Presentation, A Guide to GA Theory. Kluwer Academic Publisher, 2002.

[SD08] S. N. Sivanandam and S. N. Deepa. Introduction to genetic algorithms. Springer, 2008.

[TBWK07] Christoph Treude, Stefan Berlik, Sven Wenzel, and Udo Kelter. Difference Computation of Large Models. In ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pages 295–304, New York, NY, USA, 2007. ACM.

Hypermodelling for Drag and Drop Concern Queries

Tim Frey

Institut für Technische und Betriebliche Informationssysteme
Otto-von-Guericke University
Magdeburg, Germany
tim.frey@tim-frey.com

Abstract: Imagine a developer, who wants to alter the service layer of an application. Even though the principle of separation of concerns is widespread not all elements belonging to the service layer are clearly separated. Thus, a programmer faces the challenge to manually collect all classes that belong to the service layer. In this paper we present the Hypermodelling approach that can be used to query the necessary code fragments that belong to the service layer. We utilize methods and ideas from Data Warehousing to enable this functionality. Slicing and dicing the code in this new way overcomes the common limitation of having only one viewpoint on a program. The service layer can now be uncovered just via a query.

1 Introduction

Programmers spend most of their time reading and navigating source code [1]. Often developers alter an application layer. Usually the developer tediously gathers the artifacts that belong to the desired layer. This is a typical instance of the concern retrieval problem that programmers face commonly. Locating concerns in software is indicated to be a challenge for software developers [2,3,4]. Probably the main reason why locating concerns in source code is so difficult is because of their intertwining and the difficulty to apply separation of concerns (SOC) [5]. This principle addresses the goal to encode functionality into modules that have one primary responsibility. Normally, not all concerns are packed into separate modules and therefore a module encodes multiple functionalities at the same time [6].

Modern development environments allow building working sets, consisting of one or more fragments corresponding to the various interests for a developer [7]. Such working sets need to be built manually by a developer and can be used to filter the navigation just for the elements belonging to the working set. Typical for working sets is to split the application into their distinct layers and organize those as different working sets. This concurrent view of elements belonging to a working set is beneficial for developers by allowing them to navigate between the various fragments. But, still, the challenge of code investigation to build the working sets exists. For instance, when new classes are added to a layer, they need to be associated with the corresponding set.

To support programmers in investigating source code beyond working sets several tools are available for concern analysis or query operations [1,8,9]. The query tools offer complex and advanced query languages. This leads to the fact that composing a query is quite complicated [8]. It is indicated that a few query tools seem not to be enhancing productivity. This could result from the complexity of the tools [8].

In order to overcome the above mentioned limitations of manually creating working sets and complex queries, we present the Hypermodelling approach for the development environment. Hypermodelling enables developers to design concern queries without the need to learn a complex query language like it is used in other tools. Developers can use known views and elements via drag and drop in their queries. The same view wherein the working sets are shown is used to present the result of a query. This way, we avoid dealing with a custom query result representation view, like other tools use it. Developers can now do queries for the various layers of an application without the need to organize them into working sets.

Our contribution consists of two points: First, we show a tool that enables developers to drag and drop a query from well-known views and concerns. The result is also presented in a well-known view to enable a good usability for developers. And second, we address the complex query problem by utilizing the Hypermodelling approach for the development environment. This allows developers to design queries for code slices and their combinations.

In prior work, we described the Hypermodelling approach [10,11]. In this paper, we transfer the idea to the area of Integrated Development Environments (IDE). We present the Hypermodelling approach on an abstract level and use it to build a prototype. Also, a video is available, showing the tool in action.[1]

The paper is organized as follows. First, we present the Hypermodelling approach. Then, we describe a prototype and its architecture. Afterwards, we present a preliminary evaluation by showing a few sample use cases. Finally, related work is described and compared to the prototype, conclusions are done and future work paths are explained.

2 Hypermodelling

In this section we recap the most important facts of Hypermodelling. A more detailed description can be found in our earlier work [10,11]. Hypermodelling is founded on the idea to combine mechanisms to separate concerns and Data Warehouse (DW) like queries. Several mechanisms exist to separate concerns at the source code level. In this paper we focus on Metadata annotations [12] and normal object-oriented programming features. First, we describe an example of source code how concerns are separated. We use annotations as example for other programming mechanisms, because they are straightforward to understand. Then, we show the similarities to DW data structures and present the Hypermodelling approach.

[1] http://eclipse.hypermodelling.com

We present sample source code in Listing 1. There, a *Customer* class and a Data-Access-Object (*CustomerDAO*) are shown. The *CustomerDAO* class extends a helper class (*DaoSupport*), for table access. This is commonly done when frameworks are used. Both classes make heavy use of annotations. We show exemplary concern associations through the boxes. The elements are associated with various concerns at the same time. For instance, the *Customer* class is associated with the entity concern that belongs to the persistency part of Java. At the same time the *Customer* class is marked as deprecated. Hence, every programmer associates it with the concern that it should not be used any more by other code. Another example is the *createCustomer* method where compile errors are suppressed. The *createCustomer* method is consuming the deprecated *Customer* and, thus, a deprecated element consumer.

```
1: @Entity                                          Entity concern association
2: @Deprecated                  Deprecated concern association
3: class Customer{
4:         @Deprecated                              Deprecated concern association
5:         Customer(){
6:         …}
7: …}
8: class CustomerDAO extends DaoSupport             DAO concern association
9:
10:        @SuppressWarnings("deprecation")  SupressWarnings  concern association
11:        Customer createCustomer(){        Deprecated element consume
12:                return new Customer();            concern association
13:        }     ...}
```

Listing 1. Example for annotated source code and associations with concerns

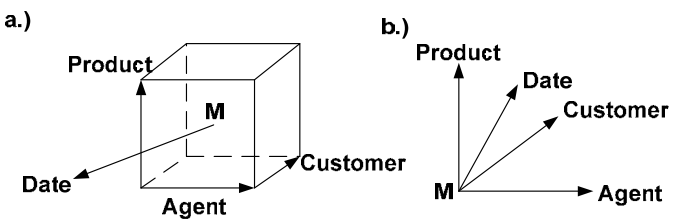


Figure 1: Multi-dimensional data within a Data Warehouse

DWs and associated techniques are used to analyze multi-dimensional data. Normally, an Online Analytical Processor (OLAP) is used to query multi-dimensional data that is stored within a DW [10,11]. We show an OLAP data structure in Figure 1a. There, the measure revenue (M) is associated with different dimensions at the same time. Revenues can be viewed by different associations with dimensions. Such associations are a product, date, a customer and an agent. Each one of those dimensions is describing an aspect of a product sale. The agent that made the deal, the bought product, the customer

that acquired it and the date of the deal. Out of these associations queries can determine the revenues for the diverse dimensions. This way, the revenues for a specific product or the revenues for a specific customer can be determined. Likewise, it is possible to discriminate through the dimensions. A sample for such discrimination can be the computation for revenues at specific date in combination with a specific product.

Figure 1b shows the cube abstracted to a pure vector model. This can be compared to the concern associations of a code fragment in Listing 1. Thus, in Hypermodelling a source code fragment is associated with multiple concerns at the same time like a measure is associated with dimensions in a DW. Hence, the main idea is to leverage principles and technologies from the area of DW to advance source code analysis.

Table 1. Concerns of Listing 1

Element / Dimensions	extends DaoSupport	@Entity	@Deprecated	@SupressWarnings
Customer		X	X	
Customer.Customer()			X	
CustomerDAO	X			
CustomerDAO. createCustomer()				X

We present a more detailed concern association example in Table 1. The table shows an alternative representation of Listing 1 and its associations. The rows are representing source code fragments and columns representing concerns. The columns show that the concerns are source code fragments themselves. Hence, the table shows the relations in the source code. The "X" indicates that a fragment is associated to a concern. For example the constructor of the class *Customer* is marked *@Deprecated*. Likewise the rest of the table-listing associations can be done.

We define: All the code associated to a specific concern is forming the corresponding concern slice. A concern is, like described before, similar to a dimension in a DW. Our approach is that similar queries to the ones used within a DW can be executed on source code. Such queries can define slices of various concerns.

Furthermore, combinations of slices can be used to discriminate results. This is needed because a slice is containing all the corresponding code fragments that belong to that concern. But, code fragments are often encoding multiple concerns at the same time. Thus, code fragments belong to multiple slices at the same time. This can also be seen in Table 1. For instance, Customer belongs to the slice Entity and also to the deprecated

slice. Therefore, queries can address combinations of slices. A sample query for Table 1, resulting in the Customer class can be: All fragments that belong to @Entity “and” to @Deprecated at the same time.

3 Implementation

The Hypermodelling tool is implemented by a custom query processing engine. This was done to show the application of Hypermodelling approach within the IDE before implementing DW techniques within the IDE. We inspect source code files in real time within a query. This has the advantage that new concerns in altered source code files are detected when a new query is executed. Nevertheless, the real time detection and the custom query engine have several limitations. Within a DW queries and analysis for slowly changing dimensions are possible. These dimensions are hard to indentify in source code. Our tool shows the first application of multi-dimensional queries within the IDE to investigate such scenarios at a later point. Additionally, the query speed slows down for larger code bases, because every file is inspected in real time. However, DW technology offers plenty of functionally to process queries faster. For instance, DW technology is capable to do pre calculations and aggregations or query analyzers to speed up queries [22]. Our earlier work [11] shows an example using actual DW technology to perform queries on code. This can be used as a way to speed up the tool with DW technology in the future. In this paper, we focus on how the Hypermodelling approach can be applied within the IDE to ease the work of a developer.

Next, the reasons to choose the different views for query composition and result presenting are described. Finally, we explain how a query is processed.

3.1 Views

As described before; programmers spend most of their time in exploring source code and it is indicated that structured code investigation is effective [13]. In the case of Eclipse the most used tool is the package explorer [14], which shows a hierarchical structure of projects, packages, classes and methods. Hence, we assume that it is the tool most used for source code exploration. However, composing a query in current query tools is difficult [15]. Therefore, we put some effort into developing an innovative way of presentation for our implementation. The goal was to respect the way programmers use Eclipse and allow structured investigations of code.

We support well known views as drag sources to allow composing a query. This avoids learning. Therefore, a developer can compose a query by dragging elements from known views into the query view. Hence, the concerns defined in the source code, libraries and in the various tools can be added to a query. As addition and as an alternative to this drag source, we realized a view where well known concerns are listed. We encoded a few of those for demonstration issues. The intention for this view is to enable a systematic query creation. Additionally, we can offer combinations of concerns as concern units in this view. We call this view the pre-defined concerns, because standard concerns can be

offered this way.

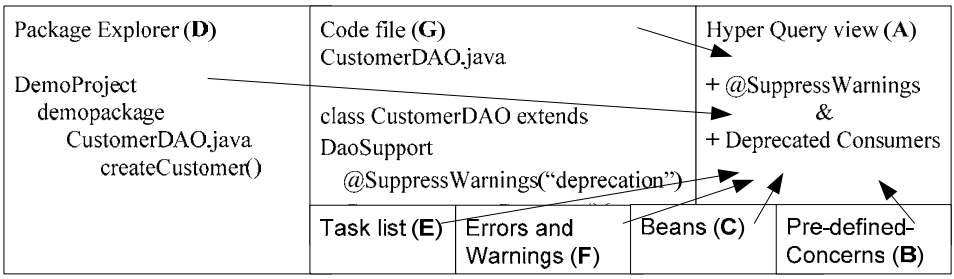


Figure 2: The Hypermodelling Tool

We use the package explorer (D) to present the results of a query to a developer. We made this choice to avoid that programmers need to learn a new view. We can justify this because it is one of the main tools that developers use for code exploration. Hence, it seems a good fit since queries are also used for exploration. For us, information reduction is a central element to inspect query results effectively. Thus, the presentation in the package explorer (D) is based on a filter that just shows the elements belonging to the result. Figure 2 presents distinct common used views in Eclipse as a schematic overview. The arrangement of the package explorer (D) on the left, the code file (G) in the middle and other views (E,F,C,B) at the bottom and right (A) is quite common. The important fact is that the Hypermodelling view can be placed anywhere. The various views serve as drag sources (B,C,D,E,F,G) to drop concerns into a Hypermodelling query (A). The arrows visualizing that concerns can be dragged and dropped from the various views into a query. For a better comprehension, we uploaded a screenshot of the Eclipse that shows the different views.[2]

3.2 Tool Usage

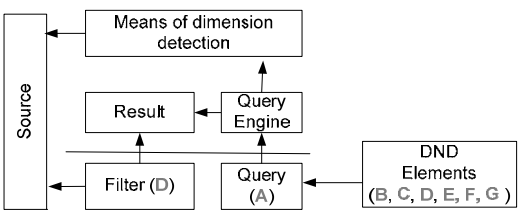


Figure 3 shows how the tool works. The letters (A-G) correspond to the ones Figure 2. Concerns can be selected via dragging and dropping from the various views (B,C,D,E,F,G), into the query view (A).

Figure 3: Tool Operation

In B we offer well-known concerns, like various annotations or compiler outputs, to a developer. Furthermore, we offer popular concerns like inheritance. A programmer can just add the extenders and implementers of classes or interfaces as concern slices to a

[2]http://eclipse.hypermodelling.com

query. A generic template for inheritance and also commonly used super classes are offered in this view. Furthermore, we support other well-known concern kinds in this view. For instance, fragments that contain unsafe type casts can be added to a query. Even a template for a string in a class or method name can be added to a query. Generally, the idea of the pre-defined concerns view (B) is to enable a fast access to often used concerns.

The other views (C,D,E,F,G) show normal views of the IDE from which concerns can directly be dragged and dropped to a query. The tool supports various code elements as concerns on a generic base. This means that, e.g. new annotations just can be dragged and dropped out of a code fragment into a query. This way, we enable the addition of annotations that are defined in programming libraries and in the code itself to a query. This is possible, because annotations in Java are defined through a language mechanism. This has the advantage to present a subsidiary possibility to add concerns to a query by using familiar views. This way, it is possible to populate concern data of an actual investigated program into queries.

View C shows a bean view of the IDE where classes that are defined as beans are shown. E shows Mylyn Tasks with which different fragments may be associated. D shows the normal package explorer. From that, Java Elements or groups of Java elements can be added to the query. This makes it very easy to include or exclude elements in the view from a query. F shows the error and warnings view as drag source. From G, the editor, annotations can be added to a query.

The query view (A) shows the Concern slices that are included("+") and excluded("-") in the query. For example, from Listing 1, the @Entity slice would consist of the class Customer. The @Deprecated slice contains createCustomer(). Another result for the query of "+" DeprecatedConsumer is createCustomer(). So, if "-" SuppressWarnings is added an empty result would be the outcome. For this reason it is possible to tie slices with an "AND" together which we visualize with the "&" in the figure. The query +DeprecatedConsumer & + SuppressWarnings determines all consumers of deprecated marked elements that show no compiler warnings. This way, developers can reveal deprecated element consuming code fragments which do not show compiler warnings. This is quite useful, to determine not updated spots in a program that do not excel at compile time or in the errors and warnings view.

At execution the query engine uses different means to determine fragments in the source code, belonging to the claimed dimensions. When all fragments that correspond to the query are found, they are stored in a result. This result is used to apply a filter to the package explorer (D). Thereby elements are excluded that are not in the result set.

A specialty of the tool is the support for drill downs. A package can be dragged and dropped into a query and be the source for a drill-down. All the consumers of the package and its types are belonging to the result slice. The types of the package (classes, enums, interfaces) can be expanded and (de)selected from the query. Likewise, the types themselves that are in a package can be extended to methods which can also be (de)selected. This way, systematic top-down investigation is possible.

4. Evaluative Use Cases

Here we present example applications for the tool to evaluate possible use cases.

4.1 Resolving the service layer

A possible application scenario of our tool is to uncover code that belongs to a certain application layer. A developer can compose a query to uncover elements that are belonging to the service layer. Therefore, he composes code slices that belong to this layer together in a query. When he executes the query he can see results like working sets in the package explorer. The slices in the query can be for example: (1) Classes that are annotated by typically in the service layer appearing annotations, like @Service; (2) Classes that have parent classes or interfaces that belong to the service layer, like controller classes; (3) Classes that contain strings that are typical for service layer, like "service" or "controller". This way a query would reveal all source code that belongs to the different concern slices. Additionally, concern slices that belong to a layer can be offered as a group in the pre-defined concerns view. This way, a developer could just drag and drop a layer query from the pre-defined concerns view without having the need to assemble the query himself. Generally, the query for a layer avoids building working sets manually.

In order to use another tool, developers need to learn the complex query language of that tool, e.g., JQuery [9], first. The Hypermodelling idea of code fragments that belong to multiple concerns is simpler. Furthermore, all the elements that belong to the service layer have to be defined by hand in a query in other tools. The Hypermodelling tool in contrast offers pre-defined concerns and offers the possibility to drag and drop concerns into a query. Lastly, the other query tools present the result of their queries within new views, where our tool enables to work with the package explorer.

4.2 Unsecured Methods

Another challenge for developers is to uncover unsecured publicly accessible methods to identify potential security leaks. A developer can compose a query for those with the Hypermodelling tool. For instance can a developer query for classes that inherit a framework class that makes them publicly available. Additionally, he can exclude the secured classes in the query. Excluding secured classes is possible, because applications are normally secured via Aspects or annotations. This structural information can be added as excluded slice to the query. Hence, all publicly accessible methods that are not secured can be uncovered easily. With other tools a complex query building process and result representation in standard views is not possible.

4.4 Deprecated Update

When software evolves, often elements are deprecated. A typical task for developers is to upgrade code not to use deprecated elements any more. Currently, the warnings and

error view lists all elements that consume deprecated elements. A programmer has to click through the whole list to get to the desired elements that need to be upgraded. With the Hypermodelling tool, a developer can add the slice of deprecated consumers to a query. This way, all the methods and classes consuming deprecated elements are revealed. Now, the developer investigates the first class and recognizes that it is extending a deprecated class. The documentation is checked to determine how the code should be updated. The developer applies the new knowledge how to update and changes that code. In that very moment the knowledge how to update this kind of case is at hand. Hence, the developer has the desire to update all elements of the same kind to avoid checking the documentation again. It would take quite a while to find the elements with the same problem in the errors and warnings view. Hence, he uses the Hypermodelling tool and adds just the slice of extenders of the deprecated classes to the query. The result of the query shows all elements using this deprecated class instantly. He can update all classes without having the need to use the errors and warnings view.

Other query tools are designed to query for code. Hypermodelling is built to query for plenty of code and other associated facts. Thus, concerns like deprecated consumers that are from the error and warnings view can only be used within the Hypermodelling tool. Moreover, this example shows that elements from code can be added to a query. Also, any kind of code slices can be combined with the error and warnings concern. Other tools are not that flexible.

5 Related Work

ConcernMapper is a tool for associating source code fragments with concerns manually [1]. The Concerns can be used to filter the package explorer and only show elements belonging to the selected concerns. Hypermodelling on the contrary has persuasive query capabilities that unleash queries with combinations of concerns far beyond the simple filtering. It can be interesting to extend the Hypermodelling tool with the capability to add concerns of ConcernMapper to a query.

Flat^3 [16] offers the possibility to search for code fragments via text strings and presents the search results in a result view. Also, concerns can be discovered via the execution of a program. Flat^3 offers rudimentary visualizations. Compared to Hypermodelling the multi-dimensional concern structure is ignored and just one concern kind is supported. Again, the detected concerns could be used to extend the Hypermodelling tool and be used within queries.

Mylyn is a tool that can be used to track down elements that are relevant for a task [15]. Hypermodelling is in contrast to Mylyn not a tool to track down concerns automatically. But tasks of Mylyn can be used within the queries. Mylyn itself lacks the possibility to calculate elements that belong to one task and not to another. Through the support of Hypermodelling this limitation can be overcome. Now it is possible to compute elements that belong to one and (not) another task or to combine queries for tasks with other concerns.

JQuery is a source code query tool with a predicate logic language [9]. The main issue is

the complexity and the custom language to do the queries [8]. Additionally, no result filter for the known views is offered. With Hypermodelling, developers have a tool that is integrating into their well known views. There is no necessity to learn a new complex query language and new views.

A multi-dimensional programming approach (HyperJ) is described by Ossher and Tarr [6] which can be seen as related and also as inspiration for the multi-dimensional Hypermodelling. There, an enhancement of object-oriented languages is proposed to overcome the limitation of intertwined concerns. Hypermodelling has in contrast the advantage that it is not a programming paradigm and it can be used together with commonly applied programming languages so no extension is needed.

The Concern Modeling Environment (CME) [17] allows multiple concerns to be associated with a code fragment. CME offers a query interface to query the concern associations and program structures and show them in a result view. The main difference to Hypermodelling is that no inspiration from DW technology is taken. Additionally, the current result representation in the package explorer and the drag and drop capabilities make it simpler to compose queries in Hypermodelling and inspect their results.

The AspectJ Development Tools (AJDT) offer support for programming with Aspects [18]. In contrast to Hypermodelling, AJDT is based on the dominant viewpoint of Aspects. Our presented Hypermodelling tool supports various kinds of concerns like annotations and unsafe type casts. In our approach Aspects are just representing one concern. Our tool could be extended to support the concern kind of Aspects additionally. Furthermore, would it be interesting to adapt the Aspect visualizing view to show not only show Aspects and their files or packages together, but to use other concerns, like annotations or Beans, and see the Aspect distribution for those.

Semmle code[3] is a source analysis and investigation tool. It offers a query interface to investigate source code and its associated elements. The results of a query are presented in a result view. Hypermodelling allows queries to be composed more smoothly through the drag and drop functionalities without having the need to learn a query language. Thereby, a well known view can be used for result investigation. Finally, the explicit multi-dimensionality of source code, where a fragment belongs to multiple dimensions respected by Hypermodelling is targeting investigations to uncover concerns. Semmle is neither leveraged for those and neither nor does it target concern investigations.

Orthographic Software Modeling (OSM) [19] is a multi-dimensional viewpoint and navigation through the various software models which seems similar to the idea to use DW like methods. To support the multi-dimensional navigation the various characteristics of dimensions are selected and through their combination the desired model is shown. In contrast to Hypermodelling neither SOC nor queries are part of the technique. Generally, OSM targets models where Hypermodelling focuses mainly on source code and the associated elements.

[3] http://semmle.com/semmlecode/

6 Conclusions and future Work

Source code fragments belong to multiple concerns. The Hypermodelling approach utilizes this and allows DW like queries for concerns. We leveraged and transferred the Hypermodelling approach to the IDE. As a result, we presented a tool that enables developers to query code in an OLAP similar manner. The tool takes advantage of the fact that the package explorer is the commonly used view in Eclipse, by offering a filter for the result representation of a query. Various presented usage scenarios indicate already beneficial applications. Related work confirmed: Hypermodelling differs from other approaches and seems to solve some issues more easily than other approaches.

Altogether, we introduced the approach to use well-known information by a programmer, to enable multi-dimensional queries on source code. Now the structural information in the code and of frameworks can be used to reveal the code of a layer. The tool demonstrates that queries and the application of OLAP like queries in the IDE are possible. Therefore, studies should be performed, considering developer productivity with the tool. Currently, we are planning to perform a study with developers for the validation of our tool.

However, although our presented tool already supports plenty of concerns we have the desire for more. Software tests and their results can be an interesting candidate to be supported for queries. Generally, our method is only supporting clear and well defined concerns. Thus, it should to be investigated how fuzzy concerns can be supported within the tool. For further development of the tool, evaluations and discussions should be carried out, to decide which concerns should be supported in the next version. Clearly, the actual application of DW technology should be considered and used in future versions to have plenty of speed up mechanisms at hand.

The package explorer as the only view for result representation needs to be reconsidered. Definitely, the current application has the advantage of providing the possibility of summoning concerns from other views this way, but only one kind of hierarchy can be shown. Maybe, there should be other result views offered to enable further investigation capabilities with other hierarchies. Likewise, programmers often work with models, when they investigate a program. Therefore, further research is needed to investigate how models can be used to represent query results. Also, how a traceability of model elements and source code can be used within queries needs to be investigated.

Furthermore, more application scenarios need to be considered to show what is possible with the Hypermodelling approach. For instance, we believe the application of the tool has benefits in the area of composite business applications (mashups) [20]. These wire various business domains together to derive a new functionality. The various business domains could be recognized as concerns and queries could be built, based on them.

Finally, it might be interesting to rank the importance of concerns with algorithms similar to CodeRank [21]. A usage scenario can be to suppose the most influential concerns, based on metrics, in the pre-defined concerns view.

Acknowledgements

Thanks go to Veit Köppen, Gunter Saake and the anonymous reviewers for their useful comments on earlier drafts of this paper.

References

[1] M. P. Robillard, F.Weigand-Warr. ConcernMapper: simple view-based separation of scattered concerns. In Proceedings of the '05 OOPSLA Workshop on Eclipse technology eXchange. ACM. 2005

[2] S. Letovsky, E. Soloway. Delocalized plans and program comprehension. IEEE Software, 3(3), pages 41–49. 1986

[3] T. Frey, M. Gelhausen, G. Saake. Categorization of Concerns – A Categorical Program Comprehension Model. In Proceedings of the Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) at the ACM Onward! and SPLASH Conferences. USA. 2011.

[4] T. Frey, M. Gelhausen, H.Sorgatz, V. Köppen. On the Role of Human Thought – Towards A Categorial Concern Comprehension. In Proceedings of the Workshop on Free Composition (FREECO) at the ACM Onward! and SPLASH Conferences. USA.2011.

[5] E. W. Dijkstra. Selected Writings on Computing: A Personal Perspective. On the role of scientific thought. Springer.1982

[6] H. Ossher, P. Tarr. Multi-dimensional separation of concerns and the hyperspace approach. In Proceedings of the Symposium on Software Architectures and Component Technology. Kluwer, pages 293-323. 2001

[7] A. J. Ko, B. Myers et al.. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. IEEE Transactions on Software Engineering, Volume 32, pages 971-987. 2006

[8] B. de Alwis, G.C. Murphy, M. P. Robillard. A Comparative Study of Three Program Exploration Tools. 15th IEEE International Conference on Program Comprehension ICPC 07. IEEE. pages 103-112. 2007

[9] K. D. Volder. Jquery: A generic code browser with a declarative configuration language. In P. V. Hentenryck, editor, PADL, volume 3819 of Lecture Notes in Computer Science. Springer, pages 88–102. 2006

[10] T. Frey. Vorschlag Hypermodelling: Data Warehousing für Quelltext. 23rd GI Workshop on Foundations of Databases. CEUR-WS, pages 55-60. 2011.

[11] T. Frey, V. Köppen, G. Saake. Hypermodelling - Introducing Multi-dimensional Concern Reverse Engineering**.** In 2nd International ACM/GI Workshop on Digital Engineering *(IWDE)*, Magdeburg, Germany, 2011.

[12] J. A. Bloch. Metadata Facility for the Java Programming Language. http://jcp.org/en/jsr/detail?id=175. 2004.

[13] M. P. Robillard, W. Coelho and G. C. Murphy How Effective Developers Investigate Source Code: An Exploratory Study. IEEE Transactions on Software Engineering, Volume 30, pages 889-903, 2004

[14] G. C. Murphy, M. Kersten, L. Findlater. How Are Java Software Developers Using the Eclipse IDE? IEEE Software, pages 76-83. 2006

[15] M. Kersten. Focusing knowledge work with task context. PHD Thesis, University of British Columbia, Canada. 2007

[16] T. Savage, M. Revelle, D. Poshyvanyk. FLAT^3: Feature Location and Textual Tracing Tool, in Proc. of 32nd International Conference on Software Engineering (ICSE'10), ACM, pages 255-258. 2010

[17] W. Chung, W. Harrison, V. Kruskal et al.. Working with Implicit Concerns in the Concern Manipulation Environment, AOSD '05 Workshop on Linking Aspect Technology and Evolution (LATE). 2005.

[18] A. Clement, A. Colyer, M. Kersten. Aspect-Oriented Programming with AJDT. Workshop on Analysis of Aspect-Oriented Software at ECOOP '03.Springer, pages 1-6. 2003

[19] C. Atkinson, D. Stoll. Orthographic modeling environment. In Fundamental Approaches to Software Engineering. Springer, pages 93-96. 2008

[20] J. Weilbach, M. Herger. SAP XApps and the Composite Application Framework. SAP Press, 1st edition, 2005

[21] B. Neate, W. Irwin, N.I. Churcher. CodeRank: A New Family of Software Metrics. In ASWEC: Australian Software Engineering Conference, IEEE, pages 369-378. 2006

[22] L. Langit, K. S. Goff, D. Mauri, et al.. Smart Business Intelligence Solutions with Microsoft SQL Server® 2008. O'Reilly. 2009

Entwurf einer domänenspezifischen Sprache für elektronische Stellwerke*

Wolfgang Goerigk[1], Reinhard von Hanxleden[4], Wilhelm Hasselbring[3], Gregor Hennings[1], Reiner Jung[3], Holger Neustock[2], Heiko Schaefer[2], Christian Schneider[4], Elferik Schultz[2], Thomas Stahl[1], Steffen Weik[1] und Stefan Zeug[1]

[1] b+m Informatik AG, 24109 Melsdorf
[2] Funkwerk Information Technologies GmbH, 24145 Kiel
[3] Arbeitsgruppe Software Engineering, Universität Kiel
[4] Arbeitsgruppe Echtzeitsysteme und Eingebettete Systeme, Universität Kiel

Abstract: Die Entwicklung elektronischer Stellwerke für den Bahnbetrieb ist ein aufwändiges Unterfangen, welches sich besonders für die zahlreichen Nebenstrecken und andere kleinere Bahnanlagen häufig als unrentabel erweist. Um in Zukunft einerseits mehr Verkehr auf die Schiene zu bringen und zudem die Kosten für den Betrieb der Infrastruktur zu senken, müssen die Hardware-Komponenten günstiger werden, aber auch die Entwicklung der darauf laufenden Software produktiver erfolgen, ohne Abstriche bei der Sicherheit zu machen.

Bisher werden für elektronische Stellwerke Prozessrechner eingesetzt, welche speziell auf das jeweilige Stellwerk zugeschnitten sind. Ebenso wird die Software speziell für die jeweilige Anlage entwickelt. Beide Komonenten müssen für den Betrieb zugelassen werden. Unser Ansatz zur Produktivitätssteigerung setzt einerseits auf den Einsatz standardisierter Hardware-Komponenten aus der Industrieautomation, hier konkret speicherprogrammierbarer Steuerungen, und andererseits auf eine Verbesserung des Softwareentwicklungsprozesses durch den Einsatz modellgetriebener Softwareentwicklung mit domänenspezifischen Sprachen und dazu passenden Werkzeugen.

In diesem Beitrag stellen wir aus dem Verbundprojekt MENGES (Modellbasierte Entwurfsmethoden für eine neue Generation elektronischer Stellwerke) den Entwurf und die Implementierung einer domänenspezifischen Sprache für die Programmierung elektronischer Stellwerke vor. Die MENGES-Sprache besteht aus einer Menge textueller Teilsprachen, für deren effiziente Benutzung eine leistungsfähige Werkzeugumgebung zur Analyse der Spezifikationen, zur Code-Generierung und zur zweckgerichteten grafischen Repräsentation von Spezifikationsteilen entwickelt wird.

1 Einleitung

Der Verkehrsträger Schiene soll in der Zukunft ein wichtigere Rolle in der Verkehrslandschaft einnehmen. Um den heutigen Anforderungen bezüglich seiner Wirtschaftlichkeit

*Diese Arbeit wird im Projekt MENGES aus dem Zukunftsprogramm Wirtschaft des Landes Schleswig-Holstein, kofinanziert mit Mitteln aus dem Europäischen Fonds für regionale Entwicklung (EFRE), gefördert.

gerecht zu werden müssen die bestehenden Bahnanlagen modernisiert werden. Wesentliche Bedeutung kommt dabei der Modernisierung der Nebenstrecken zu, die zurzeit noch mechanisch oder elektromechanisch betrieben werden. Sie sollen zukünftig elektronisch gesteuert werden.

Die Entwicklung elektronischer Stellwerke ist bisher ein sehr aufwändiges Unterfangen. Die Hardware wird speziell entwickelt, die Software wird für diese Anlagen in der Regel re-implementiert bzw. angepasst. Alle Komponenten müssen darüber hinaus verifiziert und durch staatliche Behörden zugelassen werden. Für Nebenstrecken ist dieses Vorgehen häufig unrentabel. Im Rahmen des Forschungsprojekts „Entwicklung von modellbasierten Entwurfsmethoden für eine neue Generation elektronischer Stellwerke“ (MENGES) entwickeln wir eine domänenspezifische Sprache (DSL) für die Beschreibung der Steuerungssoftware, sowie die passende Werkzeuginfrastruktur für diese Sprache.

In Abschnitt 2 stellen wir die Domäne und das Anwendungsszenario Lindaunis vor. Abschnitt 3 beschreibt die konkrete Architektur einer Steuerungseinheit in Hard- und Software. Den Kernbeitrag dieser Publikation stellen die in Abschnitt 4 beschriebene DSL und die in Abschnitt 5 beschriebene Werkzeuginfrastruktur für diese DSL dar. Verwandte Arbeiten werden in Abschnitt 6 diskutiert. Abschließend wird in Abschnitt 7 eine Zusammenfassung und ein Ausblick für das Projekt gegeben.

2 Die Domäne „elektronische Stellwerke für Regionalstellwerke“

Elektronische Stellwerke sind komplexe technische Systeme bestehend aus einer Vielzahl von bahntechnischen Anlagen wie Weichen oder Signalen und aus einer komplexen Logik zur Steuerung und Überwachung dieser Anlagen. In MENGES konzentrieren wir uns auf elektronische Regionalstellwerke (ESTW-R). Als Evaluationsszenario wird das Pilotprojekt Lindaunis genutzt, welches die Umstellung der Strecke Kiel–Flensburg von mechanischem auf elektronischen Betrieb verwirklicht.

Die fachliche Domäne Die Domäne der Regionalstellwerke beinhaltet viele Fachtermini, wovon hier einige zentrale Begriffe kurz eingeführt werden (siehe auch [Pac11]). Das Gesamtkonstrukt aus Bahnanlagen und der gesamten Steuerungs- und Überwachungstechnik, welche sich über mehrere Bahnhöfe und Streckenabschnitte erstrecken kann, wird in MENGES *Stellwerk* genannt. Es gliedert sich in *Lokalstellwerke*, welche nur einen Bahnhof bedienen, und die *Bedienzentrale*, dem Arbeitsplatz des Fahrdienstleiters. Die Lokalstellwerke werden in weitere Teilbereiche, sogenannte *Freimeldeabschnitte* (FMA), unterteilt. Diese FMA bestehen aus einer Weiche oder einem Stück Gleis und dienen der Unterteilung der Strecke in diskrete Bereiche, welche, so sie befahren werden sollen, frei sein müssen. FMA werden durch *Achszähler* voneinander abgegrenzt. Achzähler registrieren in welche Richtung ein Rad bzw. eine Achse an der FMA-Grenze rollt. Für eine Zugbewegung muss ein Streckenabschnitt exklusiv reserviert und gesichert sein. Diese reservierte Strecke setzt sich aus FMA zusammen und wird *Fahrstraße* genannt. Fahrstraßen werden durch ein Hauptsignal und einen Zielpunkt begrenzt.

Die Kontrolle eines Lokalstellwerks übernimmt ein lokaler Rechner, den wir *Stellwerks-*

kern nennen. Die einzelnen Stellwerkskerne werden über die Bedienzentrale gesteuert. Die Kerne ihrerseits überwachen die Signale, Weichen, Achszähler sowie weitere Aktuatoren und Sensoren des Stellwerks. Diese überwachten Elemente werden unter dem Begriff *Stellwerkselemente* zusammengefasst.

Beispiel-Kommando „Weiche umstellen“ In diesem Papier wird das Kommando *Weiche umstellen* (WU) als durchgängiges Beispiel genutzt, um zu zeigen, wie ein Kommando von der Bedienzentrale kommend in Steuerimpulse für die Aktuatoren umgesetzt wird, und wie die Sensordaten für die Bedienzentrale aufbereitet werden. Nehmen wir an die Weiche mit der Bezeichnung 83W1 läge links und sei gegen ein Umstellen gesichert (bahntechnisch *verschlossen*). Die Weiche soll nun nach rechts umgestellt werden. Dazu sendet die Bedienzentrale ein Kommando WU(83W1) an den verantwortlichen Stellwerkskern. Dieser dekodiert das Kommando und übergibt es der Steuerungskomponente für die Weiche 83W1. Abhängig vom internen Zustand der Komponente führt sie den Befehl aus oder, so die Weiche von einem anderen Vorgang genutzt wird, weist ihn ab. Ist die Weiche nicht anderweitig reserviert oder beansprucht akzeptiert die Komponente das Kommando und leitet den Umstellvorgang ein. Dazu instruiert sie den Weichentreiber den Verschluss zu lösen und den Stellantrieb zu aktivieren. Sobald der Treiber erkennt, dass die Weiche rechts liegt oder nach einer maximalen Laufzeit wird der Antrieb ausgeschaltet und dies der Steuerungskomponente mitgeteilt. Diese wiederum liefert der Bedienzentrale das Ergebnis der Aktion.

Speicherprogrammierbare Steuerungen Der Stellwerkskern wird bei großen Projekten durch spezielle Prozessrechner realisiert. Für die eingeschränkte Domäne der elektronischen Regionalstellwerke (ESTW-R) können jedoch einfachere Produkte genutzt werden. Im Lindaunisprojekt werden hierfür speicherprogrammierbare Steuerungen (SPS) verwendet. SPS sind Steuerungscomputer aus der Industrieautomation, die dort zum Steuern von Produktionsanlagen eingesetzt werden. SPS arbeiten zyklisch. Sie lesen zu Beginn eines jeden Zyklus Werte von den Eingängen ein, verarbeiten diese und geben dann die Ergebnisse an den Ausgängen am Ende des Zyklus aus. Als Befehlssatz kommen Operatoren, Zähler, Timer, Schleifen und Verzweigungen vor, welche mit Hilfe von verschiedenen im Standard IEC 61131-3 definierten Sprachen beschrieben werden [JT09]. Die meisten SPS kennen keine parallele Verarbeitung. Selbst jene, die mehrere Threads unterstützen, behandeln die einzelnen Pfade wie einzelne separate SPS. Dies erlaubt es SPS-Programme als streng sequenziell anzusehen.

3 Architektur eines Stellwerkskerns

Die Architektur eines Stellwerkskerns besteht aus einer Reihe von Hardwarekomponenten, welche das Steuerungssystem und dessen Anbindung an die Stellwerkselemente realisieren, sowie einer Software, welche die Steuerungslogik und eine Laufzeitumgebung umfasst.

Die Hardware des Kerns besteht aus einer Reihe von speicherprogrammierbaren Steuerungen (SPS) und digitalen Ein- und Ausgabekomponenten (DIO), wobei die SPS die Steue-

rungssoftware beherbergen und die verteilten DIO zur Ansteuerung der Stellwerkselemente dienen. Die Software umfasst Laufzeitkomponenten und generierten Code, welche in Abbildung 1 als Schichtenmodell dargestellt sind.

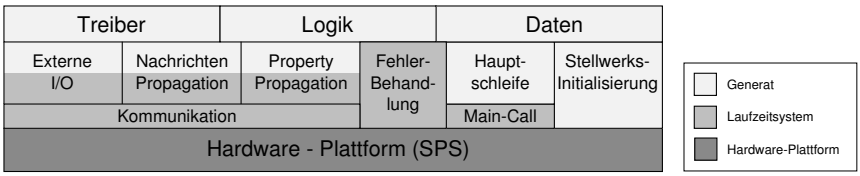


Abbildung 1: Schichtenarchitektur für die Menges Plattform

Treiber, Logik und Daten bilden die oberste Schicht, welche Struktur und Verhalten des Kerns implementiert. Darunter folgt die Anbindung der Struktur- und Verhaltensschicht an die Hardware. Diese Schicht besteht aus modellunabhängigen Komponenten (Laufzeitsystem) und aus projektspezifischen Komponenten, die generiert werden müssen.

4 Die MENGES-DSL für elektronische Stellwerke

Die DSL für Stellwerke ist in mehrere Teilsprachen unterteilt, welche im rechten Teil von Abbildung 2 dargestellt sind. Eine weitere DSL wurde für die Beschreibung von Topologie- und Projektierungsinformationen entwickelt, die jedoch austauschbar ist. Auf diese Weise kann der Heterogenität der Topologie-Domänen, die je nach Land und Anwendungsgebiet durch unterschiedliche Begrifflichkeiten geprägt sind, Rechnung getragen werden. Die Kernsprache selbst trennt die Deklaration von Schnittstellen und Typen von der Implementierung (Logik). Die Instantiierung wird mittels einer separaten Sprache beschrieben, da diese einerseits für unsere sicherheitskritische Anwendung statisch erfolgt und andererseits auf konkreten topologischen Daten beruht. Die letztendliche Abbildung auf reale Hardware wird in der Hardware-Aufbau-Sprache spezifiziert.

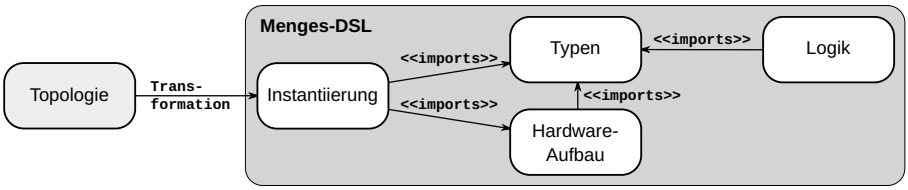


Abbildung 2: Teilspachen und Metamodelle der DSL für elektronische Stellwerke

Im Folgenden stellen wir die Beschreibung der Topologie (Abschnitt 4.1), der Typen für Stellwerkselemente (Abschnitt 4.2), der Stellwerkslogik (Abschnitt 4.3), der Hardware-Konfiguration (Abschnitt 4.4) und der Instantiierung (Abschnitt 4.5) vor.

4.1 Topologie der Gleise

Die Topologie- und Projektierungssprache erlaubt die Beschreibung von Gleistopologien und weiterer betrieblicher Strukturen. Die Basis der Beschreibung sind Gleise und Weichen, welche auf ein Knoten-Kanten-Modell abgebildet werden. Diesem Gleisgraph werden die weiteren Stellwerkselemente einbeschrieben.

Die Anzahl der Kanten pro Basisknoten hängt vom konkreten Knotentyp ab und wird durch eine Menge vordefinierter Kindstrukturen, den *Ports*, bestimmt. Ports tragen Namen und implizieren eine bestimmte Traversierungs-Semantik der Knoten. Eine Weiche ist ein spezieller Basisknoten mit den Ports head, diversion und main. Erlaubte Routen über den Basisknoten Weiche führen von head zu main bzw. head zu diversion. Diese internen Traversierungen erlauben es verschiedene Prüfungen auf dem Graphen durchzuführen, wie z.B. die Zulässigkeit der Route einer Fahrstraße. Neben Weichen kennt die Sprache noch Stumpfgleisknoten und Einbruchsstellen, die jeweils nur einen Port besitzen. Gleiskanten haben keine räumliche metrische Ausdehnung. Die Einbeschreibung von Stellwerkselementen wie Signalen oder Achszählern erfolgt als ordinale Liste. Diese Elemente können eine Richtung aufweisen (Signale) oder ohne Richtung sein (Achszähler). Basierend auf den Achszählern werden Freimeldeabschnitte definiert, welche dann zur Beschreibung der Fahrstraßen genutzt werden. Eine Weiche (mit Traversierungskanten), Stumpfgleise, und Gleiskanten sind schematisch in Abbildung 3 dargestellt, Signale und Achszähler durch schmale Pfeile bzw. Rechtecke auf den Gleiskanten angedeutet.

Abbildung 3: Knoten-Kanten-Darstellung einer mit Stumpfgleisen verbundenen Weiche.

4.2 Typen der Stellwerkselemente

Klassen-Schnittstellen, Konnektoren, Aufzählungstypen und Zustandsmengen für Stellwerkselemente werden in der Typensprache beschrieben. Sie ist das Bindeglied für die anderen Teilsprachen. Die Klassen-Schnittstellen heißen **interlocking element**. Sie kömmem Strukturen wie Datenfelder, Zustandsvariablen, Timer, Prädikate und Rollen enthalten. Datenfelder können von einem einen primitiven Datentyp oder einer Klasse sein. Zustandsvariablen dagegen sind immer vom Typ Zustandsmenge (nominale Werte und eine Zustandsübergangsrelation). Andere nominale Begriffe werden in der DSL mittels Aufzählungstypen realisiert. Für Timer müssen nur Namen angegeben werden. Sie erhalten zur Laufzeit ihren Startwert. Prädikate dienen dazu, komplexere logische Ausdrücke auf verständliche Begriffe abzubilden um so die Logik lesbar zu halten. Listing 1 zeigt eine einfache **interlocking element**-Definition.

```
interlocking element GaWeiche {
    properties
        Weichenlage lage;
    statevars
        WeichenumstellungStates wuMaschine;
    roles
        WeichenKommando.WeichenLogik interpreter;
        WeichenTreiberCom.WeichenLogik treiber;
}
```

Listing 1: Typdeklaration der Weichen-Logik

Für die Kommunikation zwischen Komponenten wird ein Rollenmodell genutzt. Klassen füllen diese Rollen aus. Die Kommunikation zwischen den einzelnen Rollen wird mit dem Konstrukt **communication description** beschrieben. Es ermöglicht den zyklischen Austausch von Eigenschaftswerten, wie auch das ereignisbasierte Übertragen von Nachrichten gemäß vorgegebenen Protokollen. Das Propagieren erfolgt gerichtet. Das heißt, nur eine Rolle darf schreiben, während die andere Rolle nur lesen darf. Die Richtung wird, wie in Listing 2 gezeigt, durch einen Pfeil zwischen den Rollen angegeben.

```
communication description WeichenTreiberCom : WeichenLogik, WeichenTreiber {
    properties WeichenTreiber -> WeichenLogik:
        boolean li;
    properties WeichenLogik -> WeichenTreiber:
        boolean motor_li;
}
```

Listing 2: Kommunikation zwischen Logik und Treiber für die Weiche (Auszug)

Das Protokollmodell basiert auf Nachrichten, die in unserem Fall durch einen Bezeichner benannt werden. Ferner können sie über ein Payload, beschrieben durch Parameter, verfügen. Aus den Rollen und Nachrichten lassen sich Protokolle zusammenstellen, was in Listing 3 dargestellt ist. Das Beispiel definiert, dass auf eine Nachricht WU der Rolle KommandoInterpreter die Rolle WeichenLogik mit einem akzeptiert oder abgelehnt antworten muss. Mit Hilfe dieser Protokolle soll es ermöglicht werden, Verklemmungen zwischen den Automaten und Prozessen automatisch zu ermitteln.

```
KommandoInterpreter WU -> WeichenLogik (akzeptiert,(erfolgreich|fehler))|abgelehnt -> KommandoInterpreter ;
```

Listing 3: Kommunikation zwischen Kommandointerpreter und Weichen-Logik im ESTW-R

Die Laufzeitumgebung stellt für beide Kommunikationsarten sicher, dass die Daten ihr Ziel in einem bestimmten Zeitfenster erreichen. So dies nicht gelingt, greift die Fehlerbehandlung und das System wechselt in einen Fehlerzustand.

4.3 Logik und Verhalten

Das Verhalten der Komponenten wird in der Logik-Sprache spezifiziert. Sie umfasst Konstrukte für die Beschreibung von betrieblichen und technischen Prozessen, Zustandsautomaten und speziellen bedingten und unbedingten Aktionen. Da die Beschreibung des Verhaltens eines **interlocking element** sehr groß werden kann, kann die Logik über mehrere Spezifikationsdokumente verteilt werden.

Die einfachsten Verhaltensbeschreibungkonstrukte sind *Aktionen*, welche Datenfelder verändern, Nachrichten versenden und Timer starten, und *Entscheidungen*, welche Datenfelder auslesen, Nachrichten empfangen und Timer abfragen (siehe Listing 4). Entscheidungen können geschachtelt werden, wobei Aktionen nur in den Blättern der so entstehenden Entscheidungsbäume stehen dürfen. Die Semantik von Entscheidungen ist wie folgt definiert: Die einzelnen Fälle werden der Reihe nach evaluiert bis eine Bedingung erfüllt ist. Existiert ein solcher Fall, wird dessen Aktion ausgeführt und die Evaluation beendet.

```
action weichenMotorEinschalten {
    this.motor = LINKS;
    start timerMotor(7000);
}
conditional prüfeVerschlossen() {
    case this.verschlossen:
        verschlussLösen();
    default:
        weichenMotorEinschalten();
}
```

Listing 4: Implementierung des Verhaltens mit Aktionen und Entscheidungen

Zustandsautomaten erlauben es komplexeres zustandsbehaftetes Verhalten zu beschreiben. Jeder Automat referenziert hierzu eine Zustandsvariable, für die er Transitionen definiert. So ist es möglich für verschiedene Zustandsvariablen verschiedene Automaten zu spezifizieren, welche dieselbe Zustandsmenge nutzen. Die Transitionen bestehen aus einem Guard und einer Aktion. Ist der Guard wahr, wird die Aktion ausgeführt. Die Priorität der Transitionen wird explizit durch eine Präzedenznummer angegeben. Des Weiteren können Transitionen wahlweise entweder direkt nach dem Betreten ihres Ursprungsszustandes oder erst in künftigen Ausführungszyklen evaluiert werden. Dieses Prinzip ist Andrés synchronem StateChart-Formalismus *SyncCharts* entnommen [And03].

Neben Automaten lassen sich betriebliche und technische Vorgänge auch mit an UML-Aktivitätsdiagrammen angelehnten Prozessen ausdrücken (siehe Listing 5). Diese Prozesse kennen nur Entscheidungen und Aktionen. Sie werden entweder durch Nachrichten angestoßen oder von einer Aktion aufgerufen. Die Semantik dieser *Prozessaktionen* ist weitestgehend gleich der oben beschriebenen Aktionen. Zusätzlich können sie noch einen Verweis (**continue**) auf die nächste Aktion oder eine Entscheidung enthalten. Wird dieser Verweis weggelassen, terminiert der Prozess in der Aktion. Entscheidungen haben, anders als oben beschrieben, ein blockierendes Verhalten, falls kein Fall wahr ist. Soll der Prozess nicht blockieren, muss ein **default**-Fall angegeben werden.

4.4 Hardware-Konfiguration

Die Hardware-Konfiguration umfasst die Beschreibung der Hardware bzgl. ihrer verfügbaren Anschlüsse. Daneben werden in dieser Beschreibung die *Typen*, welche auf das entsprechende Gerät ausgebracht werden, und die maximale Anzahl ihrer Instanzen benannt.

```
process WU { start prüfeVerschlossen;

    condition prüfeVerschlossen {
        case this.verschlossen: verschlussLösen();
        default: weichenMotorEinschalten();
    }

    action weichenMotorEinschalten {
        ...
        continue prüfeUmlauf;
    }
}
```

Listing 5: Prozess für das Kommando Weiche umstellen (Auszug)

4.5 Instantiierung

In sicherheitskritischen Systemen ist es notwendig, alle benötigten Ressourcen statisch zu allozieren [Hol06]. Dem Rechnung tragend führt die Instantiierungssprache die Beschreibungen der Soft- und Hardware zusammen. Spezifikationen in dieser Sprache instantiieren Geräte (**devices**) und Klassen (**deployment**), und legen die Verbindungen der Hard- und Softwarekomponenten sowie der externen Anschlüsse fest. Darüber hinaus belegen sie die Parameter der verwendeten Softwarekomponenten, definieren konkrete Kommunikationspartner und bestimmen die zyklisch auszuführenden Verhaltenskonstrukte (vgl. Listing 6).

```
interlocking unit AEC {
    devices
        HIMatrix−F30−S aec−s;

    deployment
        aec−s.GaWeiche = { 83W1 { lage = links }, 83W2 }
        aec−s.GpWeiche = { T83W1, T83W2 }

    connections
        WeichenTreiberCom<GaWeiche, GpWeiche>:
            81W1, T81W1; 81W2, T81W2;

    tasks
        GALogik = { GaWeiche.weichenMotorEinschalten();
                    GaWeiche.prüfeVerschlossen(); }
}
```

Listing 6: Auszug aus der Instantiierungsbeschreibung für das Beispielstellwerk Lindaunis

5 Integrierte Entwicklungsumgebung

Die Eclipse-RCP[1] stellt die Basis für unsere domänenspezifische Entwicklungsumgebung bereit. Editoren für die oben beschrieben textuellen Teilsprachen wurden mit Hilfe des Xtext-Frameworks[2] erzeugt. Ausgehend von einer formalen Beschreibung der konkreten Syntax generiert Xtext entsprechende Parser, Serialisierer und zugeschnittene Editoren, welche das Spezifizieren auf vielfältige Weise unterstützen.

[1] http://www.eclipse.org

[2] http://www.xtext.org

5.1 Textuelle vs. Grafische Darstellungen

Textuelle, konkrete Syntaxen können sehr kompakt und präzise sein, wie sich anhand der Teilsprache *Typen* (vgl. Abschnitt 4.2, Listings 1 bis 3) beobachten lässt. Zudem lassen sich textuelle Spezifikationen leicht versionieren und effizient bearbeiten. Werden diese Spezifikationen jedoch umfangreicher, kann die gewonnene Kompaktheit zu Lasten der Übersichtlichkeit und der Verständlichkeit gehen. Dies betrifft besonders Beschreibungen von Verhalten (Dynamik), wie anhand von Listing 7 deutlich wird. Grafische Repräsentationen sind in solchen Fällen oft das geeignetere Darstellungsmittel, vgl. Abbildung 4. Sie sind durch die geeignete Wahl von Symbolen zumeist sprechender und einfacher zu erfassen. Ihre Erstellung und Bearbeitung ist häufig jedoch sehr zeitaufwändig [FvH10].

```
state machine Gleisabschnitt_beansprucht_SM {
 start nicht        −> DWeg, FWZ, FLR, FWR;
 DWeg               −> nicht, DWeg_FLR, FWZ_DWeg;
 FLR                −> nicht, DWeg_FLR, FLR_FLR,
                       FWR_FLR, FWZ_FLR;
 FWR                −> nicht, FWR_FLR;
 FWZ                −> nicht, FWZ_FLR, FWZ_DWeg;
 DWeg_FLR           −> DWeg, FLR, DWeg_FLR_FLR;
 FLR_FLR            −> FLR, DWeg_FLR_FLR,
                       FWR_FLR_FLR,
                       FWZ_FLR_FLR;
 FWR_FLR            −> FLR, FWR, FWR_FLR_FLR;
 FWZ_DWeg           −> DWeg, FWZ;
 FWZ_FLR            −> FLR, FWZ, FWZ_FLR_FLR;
 DWeg_FLR_FLR       −> DWeg_FLR, FLR_FLR;
 FWR_FLR_FLR        −> FWR_FLR, FLR_FLR;
 FWZ_FLR_FLR        −> FWZ_FLR, FLR_FLR; }
```

Listing 7: Zustände und gültige Übergänge

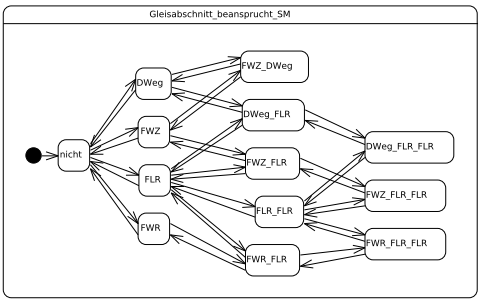


Abbildung 4: Grafische Darstellung zu Listing 7

Aus obigen Überlegungen leitet sich das Ziel ab, Vorteile beider Paradigmen zu vereinigen. Nachteile wie der Bedarf nach händischem Arrangieren von Symbolen können dabei durch Werkzeuge ausgeglichen werden. Fuhrmann und von Hanxleden fassen solche Aspekte unter dem Begriff der *Pragmatik von Modellierungssprachen* zusammen [FvH10]. Der Begriff Pragmatik ist dabei angelehnt an Morris' Auffassung von der Pragmatik natürlicher Sprache [Mor38]. Gemeinsam mit der Syntax und der Semantik bestimmt sie, wie Bedeutung in Sprache ausgedrückt und verstanden wird.

Als Basis für die Entwicklung effizienter Spezifikations- und Browsingtechniken dient das KIELER-Projekt,[3] dessen Kernkomponenten das automatische Positionieren von Diagrammelementen (KIELER Meta Layout) und das Management verschiedenartiger Sichten auf Modelle (KIELER View Management) sind [FvH10].

5.2 Transiente grafische Darstellungen

In MENGES liegen die Spezifikationen stets in textueller Form vor. Um diese grafisch darzustellen, wird der im Rahmen des KIELER-Projektes entwickelte Mechanismus der transi-

[3] http://www.informatik.uni-kiel.de/rtsys/kieler

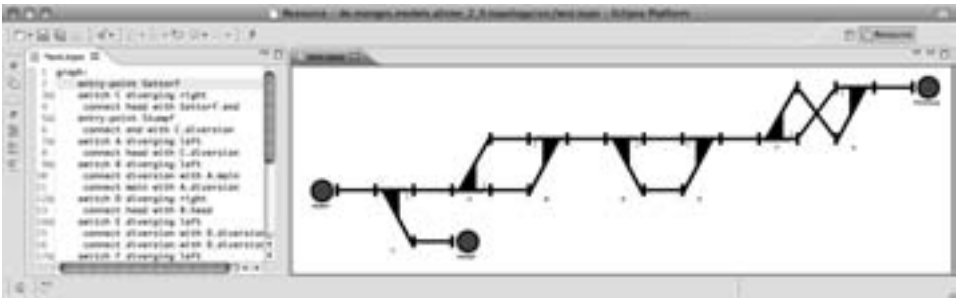


Abbildung 5: Textuelle Topologie-Spezifkation (links) und grafische Darstellung (rechts)

enten Sichten genutzt. Im Gegensatz zur händischen Erzeugung von Diagrammen ermöglicht er die Synthese sprechender Darstellungen. Dies kann sowohl automatisch wie auch *on-demand* geschehen. So wurde die grafische Darstellung in Abbildung 5 ohne das explizite Anfordern durch den Modellierer aus der im linken Bereich gezeigten Spezifikation erzeugt. Etwaige Änderungen wie die Korrektur der Lage der rechts oben positionierten Weiche werden unmittelbar in die Grafik übernommen. Der Inhalt der Grafik muss dabei nicht auf eine Teilspezifikation beschränkt sein. Dies ermöglicht unter anderem das Expandieren von referenzierten und in anderen Dokumenten definierten Elementen. Zudem kann die grafische Darstellung als Navigationshilfe genutzt werden. Die Selektion eines Diagrammelements rückt automatisch das zugehörige Element in der Quelle in den Fokus.

5.3 Code-Generierung und Simulation

Aus Spezifikationen in der textuellen DSL wird mittels Xpand/Xtend[4] Code entsprechend dem Standard IEC 61131-3 [JT09] generiert, welcher zur Simulation mit CoDeSys[5] ausgeführt und mit dem Monitoring-Framework Kieker[6] [vHRH+09] analysiert wird.

6 Verwandte Arbeiten

Unser Ansatz zielt darauf die Entwicklung der Software für Stellwerkskerne mit einem generativen Ansatz und einer zugeschnittenen DSL effizienter zu gestalten. Der Ansatz in [SOE+08] verfolgt das gleiche Ziel, ist aber weitestgehend komplementär. Er geht davon aus, dass es zu einzelnen Elementen der Topologie vorgefertigte Softwarekomponenten gibt. Die Gesamtkomposition erfolgt hier durch die Beschreibung der Topologie, woraus ein Generator Zielcode für ein Lokalstellwerk erzeugt. Die verwendete Topologiesprache *Train Control Language* (TCL) könnte so auch mit der MENGES-DSL genutzt werden. Der

[4] http://eclipse.org/modeling/m2t/?project=xpand

[5] http://www.3s-software.com/

[6] http://kieker.sourceforge.net/

Zielcode wäre in diesem Fall die Instantiierungssprache. Aufbauend auf der TCL wird in [SZLT+11] eine Fallstudie beschrieben, welche mit Produktlinien und Topologiemustern arbeitet, um so die Komplexität von Topologien zu reduzieren. Dieses Vorgehen ist in MENGES ebenso möglich, steht aber nicht im Fokus dieser Arbeit.

Haxthausen und Peleska beschreiben in [HP02] eine DSL mit dem Fokus auf Routenplanung und sicheren Zuglenkungsbetrieb. ESTW-R hingegen enthalten keine automatische Zuglenkung, Fahrwege werden nach wie vor durch Fahrdienstleiter eingestellt.

Das Anliegen der Arbeit von Thomas et al. [T+05] ist vergleichbar mit unserer Situation: Die Aufgabe des vielfachen Erstellens von SPS-Code soll mit einem generativen Ansatz bewältigt werden. Die verwendeten Technologien unterscheiden sich jedoch grundlegend. Neben der reinen Entwicklung von DSL & Code-Generierung steht in MENGES jedoch auch Technologietransfer von der Wissenschaft in die mittelständische Wirtschaft im Fokus.

7 Zusammenfassung und Ausblick

In dieser Arbeit wird der Entwurf einer domänenspezifischen Sprache als Menge textueller Teilsprachen für die Programmierung elektronischer Stellwerke vorgestellt. Diese wurden in einer etablierten Entwicklungsumgebung implementiert und um weitere pragmatische Funktionen (Modellsichten, Navigationshilfen) ergänzt. Mit dem so entstandenen Werkzeug kann modellgetrieben SPS-Code [JT09] generiert werden.

Die bisherigen Anwendungen auf Testszenarien haben gezeigt, dass die Sprache gut zur Beschreibung von Steuerungsprozessen geeignet ist. Im Vergleich zu den bisherigen Spezifikationen der Funkwerk Information Technologies GmbH mit sehr großen Zustandsdiagrammen ist die textuelle Beschreibung, unterstützt durch zweckgerichtete grafischen Modellansichten, wesentlich kompakter und gleichzeitig übersichtlicher und handlicher.

Zurzeit evaluieren wir die Sprache an größeren Beispielen aus dem Projektgeschäft der Funkwerk Information Technologies GmbH systematisch mit Hilfe der GQM-Methode [BCR94, SB99]. Konkret werden wir Verhalten mit Prozessen und Automaten spezifizieren und überprüfen, für welche Problemstellungen sich die jeweiligen Sprachkonstrukte optimal eignen. Ferner werden wir die Werkzeuge mit weiteren grafischen Sichten versehen und Model-Checking-Methoden [BBB+04] für die Entwickler bereitstellen. Für die dynamische Analyse der erstellten Steuerungssysteme wird modellgetriebene Instrumentierung genutzt [BH09, vHKGH11].

Literatur

[And03] Charles André. Semantics of SyncCharts. Bericht ISRN I3S/RR–2003–24–FR, I3S Laboratory, Sophia-Antipolis, France, April 2003.

[BBB+04] R. Buschermöhle, M. Brörkens, I. Brückner, W. Damm, W. Hasselbring, B. Josko, C. Schulte und T. Wolf. Model Checking — Grundlagen und Praxiserfahrungen.

Informatik-Spektrum, 27(2):146–158, April 2004.

[BCR94] Victor R. Basili, Gianluigi Caldiera und H. Dieter Rombach. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[BH09] Marko Boskovic und Wilhelm Hasselbring. Model Driven Performance Measurement and Assessment with MoDePeMART. In *Proceedings of the ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems (MODELS '09)*, Jgg. 5795 of *LNCS*, Seiten 62–76. Springer, Oktober 2009.

[FvH10] Hauke Fuhrmann und Reinhard von Hanxleden. Taming Graphical Modeling. In Dorina C. Petriu, Nicolas Rouquette und Øystein Haugen, Hrsg., *Proceedings of the ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS '10)*, LNCS. Springer, Oktober 2010.

[Hol06] Gerard J. Holzmann. The Power of 10: Rules for developing Safety-Critical Code. *IEEE Computer*, 39(6):95–97, Juni 2006.

[HP02] Anne E. Haxthausen und Jan Peleska. A domain specific language for railway control systems. In *Proceedings of the sixth biennial World Conference on Integrated Design & Process Technology (IDPT '05)*, Juni 2002.

[JT09] Karl Heinz John und Michael Tiegelkamp. *SPS-Programmierung mit IEC 61131-3: Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen*. Springer, 4. Auflage, 2009.

[Mor38] Charles William Morris. *Foundations of the theory of signs*, Jgg. 1 of *International encyclopedia of unified science*. The University of Chicago Press, Chicago, 1938.

[Pac11] Jörn Pachl. *Systemtechnik des Schienenverkehrs: Bahnbetrieb planen, steuern und sichern*. Vieweg+Teubner, 6. Auflage, 2011.

[SB99] R. Solingen und E. Berghout. *The Goal/Question/Metric method: a practical guide for quality improvement of software development*. McGraw-Hill, 1999.

[SOE+08] Andreas Svendsen, Gøran K. Olsen, Jan Endresen, Thomas Moen, Erik Carlson, Al Kjell-Joar und Oystein Haugen. The Future of Train Signaling. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems (MoDELS '08)*, Seiten 128–142, Berlin, Heidelberg, 2008. Springer-Verlag.

[SZLT+11] Andreas Svendsen, Xiaorui Zhang, Roy Lind-Tviberg, Franck Fleurey, Oystein Haugen, Birger Moller-Pedersen und Goran Olsen. Developing a Software Product Line for Train Control: A Case Study of CVL. In Jan Bosch und Jaejoon Lee, Hrsg., *Software Product Lines: Going Beyond*, Jgg. 6287 of *LNCS*, Seiten 106–120. Springer Berlin / Heidelberg, 2011.

[T+05] G. Thomas et al. LHC GCS: A Model-Driven Approach for Automatic PLC and SCADA Code Generation. In *Proceedings of the 10th International Conference on Accelerator and Large Experimental Physics Control Systems (iCALEPCS)*, 2005.

[vHKGH11] André van Hoorn, Holger Knoche, Wolfgang Goerigk und Wilhelm Hasselbring. Model-Driven Instrumentation for Dynamic Analysis of Legacy Software Systems. In *Proceedings of the 13th Workshop Software-Reengineering (WSR '11)*, Mai 2011.

[vHRH+09] André van Hoorn, Matthias Rohr, Wilhelm Hasselbring, Jan Waller, Jens Ehlers, Sören Frey und Dennis Kieselhorst. Continuous Monitoring of Software Services: Design and Application of the Kieker Framework. Bericht TR-0921, Department of Computer Science, University of Kiel, Germany, November 2009.

Generic Roles for Increased Reuseability

Andreas Mertgen

Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin
Ernst-Reuter-Platz 7
D-10587 Berlin
andreas.mertgen@tu-berlin.de

Abstract: Role-based programming, as in the *Object Teams/Java* (OT/J) language, aims to improve object-oriented programming with regard to separation of crosscutting or context-related concerns. Therefore, OT/J introduces class-like modules for roles and context, which connect common classes to build collaborations. However, since role and base objects are directly linked, it implies strong coupling and limited possibilities of reuse. This research aims to create a generic way of expressing connections between a collaboration and its base in order to further decouple modules and enhance their reusability. We introduce a quantification mechanism based on logic meta-programming in *Prolog* that allows using generic references to declaratively defined program elements, which are transformed to build valid OT/J code. We propose that the use of logic meta-variables improves the expressiveness and genericity of role-based programming.

1 Introduction

Separation of concerns is a key principle applied in designing complex, flexible, and reusable software systems. Today's programming languages need to provide features that strongly support modularization. In object-oriented programming (OOP), classes and objects are abstractions of real-world entities with state and behavior and form a well-accepted set of modules to implement functional concerns. However, there are certain dimensions of modularization where the concepts of classes and objects reach their limitations. Our research focuses on two such dimensions: (1) the modularization of crosscutting concerns and (2) modularization of collaboration and context. Both areas of concern cannot be clearly separated in a single class module. Crosscutting concerns are often non-functional concerns and are scattered around multiple classes or tangled together in a single module. Collaborations may involve several classes and instances and depend on a context; all collaboration members together build a strongly coupled composite. Several developments exist that address these concerns and improve modularization support in OOP.

Modularization of crosscutting concerns is addressed in aspect-oriented programming (AOP) [KIL+97]. An aspect is a module that defines *advices* and *pointcuts*. The *advice* construct

is generally similar to a method and defines additional or altered program behavior. *Pointcuts* specify where the advice behavior has to be applied in the influenced base program by describing a set of well-defined points in the program control flow - the so-called *join points*. A pointcut is defined by stating conditions about the structure, state and/or dynamic behavior of a program. The idea of influencing multiple points in a program with a single module is called *quantification*. Quantification is a key technique for modularizing crosscutting concerns [FF00]. Currently, the most prominent apect-oriented language is AspectJ [KHH+01].

Modularization of collaborations is addressed in role-oriented (RO) programming. A role describes the intersection of an entity with a context. Entities may exist outside a context, and their features are defined independently, as in the case of ordinary objects. A context builds a scenario, in which several entities assume specific roles in a collaboration. One example is an entity, `Person`, which plays a role `Employee` within a context, `Company`. Two relationships define a role (e.g., `Employee`): (1) a player relationship to a base (`Person`), which plays the role and (2) a containment relationship to a context (`Company`), which harbors the role. A role uses and extends its base. The context-specific state and behavior are attached to the context and its roles. Thus, all context and role-dependent concerns can be modularized independently and do not get tangled within the base modules. The notion of roles is used in the design and modeling of software, but it is hardly supported in mainstream programming languages. A language with explicit support for roles is Object Teams/Java [Her07], which is the focus of this paper.

In Object Teams, a context is represented by a so-called *team* class. Within a team, there may be several role classes, each of which may be bound to a base class via a *playedBy* relationship. For each base class instance, in each surrounding team, an instance of a bound role class may exist. The *playedBy* relationship between role and base class shows many similarities with the *extends* relationship between sub- and superclass. First, the role may delegate to all methods and fields of its base (so-called *callout* bindings). Second, the role may override any method of the base with specialized behavior (*callin* bindings). Third, variables declared with the role type are substitutes for variables declared with the base type (*translation polymorphism*).

2 Problem Description

Although role orientation in Object Teams shares several ideas with aspect orientation, few possibilities exist for quantification. So far, binding of roles to base classes, including callin bindings, have to be made explicitly. Although a programmer may select the bindings on the basis of specific criteria, which are based on program structure and optionally qualified by program state, since there is no abstraction for describing this selection, the programmer may not express the binding in a reuseable way. Because of this, a programmer encounters numerous problems, which are listed below, and this paper will seek to contribute to the solutions to these problems:

1. **Modularization and reuse.** The missing generalization feature of selection criteria implies that there is no use (nor reuse) of any selection logic for quantifying over binding elements. An abstract selection mechanism would allow quantification of such elements, thus enhancing modularization and reuse options.

2. **Evolution and fragility.** Explicit bindings imply a high coupling between role and base classes, which affects the robustness of the program. Thus, evolution of the base program requires a review of all affected roles. A declarative abstraction of these bindings would loosen the coupling and mitigate evolution issues.

3. **Maintenance.** Lack of quantification support is an inconvenience. Although it is possible to achieve the same results without using quantification, it entails coding overhead, which may be cumbersome and error-prone. Furthermore, a declarative definition of bindings can also contribute to the readability and documentation of the code.

The join point model for callin bindings in Object Teams is limited to *method call interception*, that is, the execution of a base method may be delegated to a corresponding role method, analogous to the *method overriding* between sub- and superclass. At first sight, from an aspect-oriented perspective, this seems to be a hindrance; however, on closer inspection, binding via method call interception fits in well with the ideas of roles and context. Callin bindings may be viewed as *contextual method overriding*. Keeping bindings at the granularity of methods prevents unintended interference with invariants, pre- and post-conditions, and encapsulation. The limitations of bindings reflect a trade-off between expressiveness, safety, and usability in language design.

By contrast, quantification in AOP, like AspectJ, utilizes an extensive join point model. Numerous pointcut designators exist to select join points by kind, scope, or context. In addition, the programmer may use wildcards to describe patterns; e.g., a pointcut `call(void set*(..))` would describe all calls of setter-methods in a single statement. However, the feasibility of such patterns simply relies on naming and structuring conventions. There is no relation with regards to content between the intention of the programmer and the semantics of a wildcard expression. For example, a call to a method named `setup` would also match the example pointcut, which may not have been intended by the programmer. This may lead to unintended mismatches and pose a problem during evolution, e.g., during refactoring, when a programmer renames or moves a method without considering the influence of such changes on the evaluation of pointcuts [KS04]. Thus, wildcards seem to be a "one-size-fits-all" approach, requiring the arranging of patterns to succeed in complex scenarios [GB03].

Nevertheless, the concept of quantification in AOP has proved to be very useful in addressing crosscutting concerns. Therefore, we believe that quantification can improve the power of an RO approach like Object Teams, even though the join point model is less sophisticated.

3 Motivating Example

To illustrate the motivation for this work, we examine a typical example of a non-functional, crosscutting concern: synchronizing UI views with their corresponding data model. The scenario involves objects (playing the role of observers), which have to keep track of the current state of other objects (playing the role of the observed subjects). Such tasks are commonly realized by using the observer pattern, where every subject is responsible for notifying its dependent observers about any relevant changes in its state. Triggering the notification is a crosscutting concern, influencing one or more mutator methods in each involved subject.

As a concrete example, consider a simple application that displays geometrical figures on the screen. If any change occurs in a figure, the displaying object must be notified and updated automatically. An object-oriented approach (see Figure 1) usually includes a (possibly abstract) superclass or interface (e.g., `Figure`), which is the root type for all classes of view-relevant objects (e.g., `Point` and `Line`). These classes represent the subjects and inherit a `Subject` class that serves two purposes: (1) maintaining subject-observer mappings (`add-` and `removeObserver`) and (2) triggering the update logic (`notify`). The latter is performed whenever a subject wants to report a state change, which leads to calling all its registered observers (`update`).

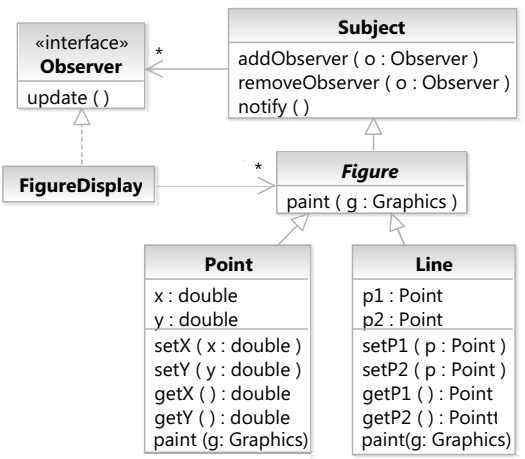


Figure 1: A simple system for displaying figures implementing the observer pattern

Variations of this example scenario are widely used in the field of aspect-orientation; the characteristics of the standard object-oriented pattern implementation in Java as well as the improvements of an aspect-oriented approach in AspectJ are demontrated in [HK02]. In terms of modularization of the crosscutting concern, the RO approach with Object Teams is quite similar to the aspect-oriented solution. The observer functionality is sourced out to an additional module, a *team* class, as shown in Listing 1.

The `FigureDisplay` class contains a list of figures to be displayed on the screen, reflecting their current state. Within this context, classes `Point` and `Line` play the role of subjects; for each, a role class is defined and bound to its base class *(lns. 8, 13)*. A base guard dynamically maintains the subject-observer mapping. An instance of `Point` should play the `PointSubject` role only if it is part of the team's figure list *(ln. 9)*; the same applies to `Line` instances *(ln. 14)*. The updating mechanism is realized with a callin binding to all mutator methods, the execution of one of these base class methods triggers the execution of an `update` in the team *(lns. 10, 15)*. No additional code is required. Any role-related code is defined within the module. Thus, `Figure` instances are completely unaware of their role as a subject and their original implementations are not required to change. The observer may also be implemented as a role. However, in this case, we prefer to implement the observer as a team because it is not only a role within a context; it actually **is** the context.

```
1  public team class FigureDisplay {
2      private List<Figure> figures;
3      ...
4      public void update() {
5          for(Figure fig : figures) {fig.paint(graphics);}
6      }
7
8      protected class PointSubject playedBy Point
9          base when (figures.contains(base)) {
10         update <- after setX, setY;
11     }
12
13     protected class LineSubject playedBy Line
14         base when (figures.contains(base)) {
15         update <- after setP1, setP2;
16     }
17 }
```

Listing 1: FigureDisplay

To avoid unnecessary updates (e.g., those caused by nested mutators), update calls should be buffered and accumulated, instead of being executed immediately. However, for the sake of simplicity, we refrained from optimizing and omitted all parts not relevant to the pattern mechanism.

As stated in [HK02], the structure of a pattern implementation contains common parts essential to all pattern instantiations and individual parts specific to each instantiation. The common parts of the observer pattern are as follows:

1. Roles for subjects and observers
2. Maintenance of subject-observer mapping
3. General updating mechanism (trigger)

The individual parts of the observer pattern are:

4. Binding of roles to certain objects
5. Binding of triggers to certain methods
6. Specific updating procedure of the observer

For the purpose of reuse in further developments, all common parts (1-3) may be extracted into an abstract class, whereas individual parts (4-6) need a concrete extension. This is good practice for the OO, AO, and RO approaches mentioned, but we argue that in this scenario (as in many others) there are additional opportunities for abstraction and reuse.

On closer examination of the observer `FigureDisplay`, we see that information about the individual parts' role and trigger bindings (4 and 5) is inherent to the updating procedure (6). Objects that may be considered for role playing are those accessed during `update` (members of the attribute `figures`); their relevant base classes are `Figure` and all its subclasses (4). Thus, a call to `Figure.paint` updates the display. Hence, all properties of a figure instance read within `paint` seem to be relevant for updating. Changes of exactly these properties should trigger the observer, which may be accomplished by binding all mutator methods of these properties (5).

Parts 4 and 5 pose a quantification issue. Although we find the required information inherent in the program structure, utilizing this information requires complex code analysis. A programmer could clearly describe the idea of the update-mechanism, e.g., *"trigger the update after the execution of any method that may cause a change of a field read somewhere in my update processing."* Such a statement could be reused in many similar cases, e.g., for updating a textual data representation in a table view; no further change is needed except for selecting the update process. However, existing language features are not capable of expressing the quantification logic in a programmer's mind. Neither code analysis nor the use of analysis results are supported by the languages presented so far in this paper.

4 Generic Modules

Our approach aims to enhance abstraction and reuse capabilities in scenarios similar to the one described above, where we prefer to reference program elements like classes and methods in a generic way to promote reuse in multiple scenarios. Therefore, we introduce logic meta-variables, which will be bound and replaced by the use of meta-programming. Our goal is to build an extended version of the OT/J language, called *Generic Object Teams* (GOT), which during a precompilation step transforms its code into valid OT/J code. This way, we ensure full compatibility with OT/J and prevent interference with existing tools and the compiler. As the transformation process implies, we rely on static program information and do not intend to use dynamic execution properties. First, the generic parts of GOT represent regular static elements and do not have any other effect on program execution. Second, dynamic dependencies may still be addressed by guard predicates, although they will only affect program execution and not the transformation. To handle meta-variables in GOT , we introduce three language constructs: (1) queries to declaratively describe sets of program elements; (2) matching statements to evaluate

queries and bind free variables; and (3) *per*-blocks to build the unit for applying meta-variables and code transformation.

4.1 Querying Program Elements

To facilitate the binding of meta-variables, we need to offer the option of querying various program elements about their properties and relationships, thus making the abstract syntax tree (AST) information available to the programmer. Several similar approaches that use XQuery, SQL or OCL [EMO04, SS08] exist, featuring a direct and readable syntax for single elements. However, they do not support transitive closures. Queries involving multiple elements may make frequent use of quantifiers (e.g., *forall* and *exists*) and include relations across tree branches, resulting in recursion and/or nested tree walking. Instead, we have chosen a logic-based representation in Prolog, which offers more concise and elegant expressions in such scenarios because information is globally available in facts without the need to navigate a tree structure. For each AST node type, there is a fact representation. For example, a method call is represented in a fact, *callT(#id, #enclosingMethod, #receiver, #method, [#args, ...])*, holding a unique id for the fact, a reference (by id) to the calling method, a reference to the expression representing the receiver of the call, a reference to the invoked method, and a list of references to argument expressions. Thus, a Prolog query may include meta-variables that get unified to every possible match in the factbase; e.g., the query *callT(_, Caller, _, Callee, _)* would match pairs of the meta-variables *Caller* and *Callee* to the method ids, such that the Caller invokes the Callee at least once in the program (using wildcards for irrelevant information). Our query module attempts to encapsulate logical queries in a manner as similar as possible to that in Java. A query is a special kind of method, marked with the *otquery* keyword (see listing 2) for processing meta-variables. A query is a conditional, built from facts or other queries.

```
1  otquery qrySubjects(?Method ?update, ?Class ?baseclass,
               ?Method ?setter) {
2      ?Field -readField;
3      ?Method -calledMethod, -executedMethod;
4      ?Method -executedMethod;
5      call(_, ?update, _, -calledMethod, _) &&
6      overrides(-executedMethod, -calledMethod) &&
7      readsField(-executedMethod, -readField) &&
8      setsField(?setter, -readField) &&
9      isMemberOfClass(?setter, ?baseclass)
10 }
```

Listing 2: A query for the declarative description of the observer's subjects

Two points about meta-variables need to be mentioned: (1) they are typed with a special set of types and (2) they are declared using a prefix. These points are discussed in detail as follows:

Typing of variables. Meta-variables may store values of certain types. To be consistent with typed languages OT/J and Java, the newly introduced meta-variables shall follow the rules of declaration. Since the set of possible types is different, they will be indicated by using a *"?"* prefix. Our type set includes *?Class*, *?Method* and *?Field* and others related to the AST. For more general purposes, the types *?String*, *?int* and *?boolean* are also included. Typing allows the generic parts of the code to be statically checked before the evaluation; e.g., in a playedBy-statement, we clearly expect a meta-variable of type *?Class*. Any other type could not possibly lead to a valid result.

Prefixes. Meta-variables get bound to a set of matches. Although a match-statement looks like a method, its parameters are either *in* or *out* as in Prolog; in contrast to Java's strict call-by-value method parameters. The prefixes allow narrowing the binding options of a variable: The - (*"minus"*) declares an *out*-parameter, meaning the variable must not be bound at evaluation time. In contrast, the + (*"plus"*) declares an *in*-parameter, which has to be bound to evaluate other dependent properties. The *?* indicates an *in*- or *out*-parameter.

4.2 Integration of Meta-variables and Transformation

To offer the aforementioned genericity for OT/J, meta-variables should be accepted in the least by the playedBy and callin-bindings, though they would generally be accepted in several other locations as well. Listing 3 shows a sketch of a generic role for the observer example. The base class of the `Subject` role is defined by the free variable `?b`, whereas the triggering callin is defined by the free variable `?m`. Both variables have to be declared and bound. An obvious location to do this is on the level of the enclosing team, assigning the team as the main unit for collaborations. Therefore, a statement *match*, which looks similar to a method, is appended to the team declaration. The match statement declares the variables as its parameters, enabling them to be used within the team body. Their bindings are defined in the match body by a query expression.

```
1  public team class FigureDisplay
2    match(?Class ?b, ?Method ?m){qrySubjects(update,?b,?m)} {
3    ...
4    per (?b) {
5      protected class -Subject playedBy ?b
6            base when (figures.contains(base)) {
7        per (?m) {
8              update <- after ?m;
9        }
10     }
11   }
12 }
```

Listing 3: Version of FigureDisplay with Generic Object Teams

The query for the observer example in listing 2 realizes the selection of program elements as intended by the programmer. In the match statement, the first parameter is bound to the `FigureDisplay.update` method, leaving the other parameters open as outputs, expecting a set of tuples for base-classes and setter methods. The call fact will match meta-variable `-calledMethod` to `Figure.paint`. Considering dynamic dispatch, we need all methods overriding the `Figure.paint` method. A query `overrides` (not included in the example) then binds the meta-variable `-executedMethod` to match `Point.paint` and `Line.paint`. Then, we need to identify the fields that are (indirectly) read in these methods, and the methods (and the classes they belong to), which mutate that fields. This way, we finally identify the relevant setter methods that trigger an update. The evaluated result of the match for the observer example will be a set of class/method tuples: $\{(Point, setX); (Point, setY); (Line, setP1); (Line, setP2)\}$.

The goal of the transformation is to change the team class according to the matches so that every code fragment dependent on a meta-variable is generated once for every match instance. In the class body, a meta-variable may only appear within a *per-block*. A per-block defines a non-empty set of meta-variables to be in its scope. Per-blocks may be nested; the inner block extends its set of meta-variables by the set of the outer block. A meta-variable may be put in place of a direct reference. In the example in listing 3, the playedBy relationship holds `?b` in place for the base class, and the callin-binding holds `?m` in place for base methods. Per-blocks control the transformation; for each tuple of matching values of a per-block's set of meta-variables, the per-block's body gets generated once in the transformation output with every occurrence of a meta-variable replaced by its matched value.

Every generated program element on the per-block level must use an unbound meta-variable for its name because simple names would collide if there is more than one instance in the generation output. These elements may be classes, methods, or fields. Such meta-variables are defined in place and are not declared in a matching statement. Their types are inferred and bound by the transformation, e.g., class `-Subject` is stated as a meta-variable. To prevent ambiguous expressions, a non-declared meta-variable cannot be referenced from outside the per-block; inside the per-block, it is unique and safe to use.

Finally, the code of the example will transform to a team with two role classes replacing `-Subject`, one played by `Point`, the other played by `Line`, both having two callin statements linking the update method to setter methods of their base class. The `FigureDisplay` class remains decoupled of the base classes and is robust with respect to evolution. For instance, consider adding a new property in one of the figure classes, e.g., a color, or adding a new figure class; in all cases, the team and the query will adapt the new situation without change. In addition, the template class may be used in other observer scenarios, e.g. managing the textual representation of the figures and their properties in a table. All that is required is to give different arguments for the match-query. Different match-instantiations of GOT team classes may be realized by inheritance, though the details of this mechanism are beyond the scope of this paper.

5 Related Work

A role-based approach explicitly targeting cross cutting concerns is presented in [HMK05]. The authors demonstrate the value of abstracting cross cutting concerns with roles. However, this approach focuses on refactoring design patterns using AOP. Selection mechanisms are based on lexical information and require manual adjustments. The approach does not propose a general-purpose role-based programming language and follows the already discussed pointcut principles. The fragility of pointcuts is a well-documented problem; approaches in this field aim to support the programmer by providing either mechanical assistance in maintenance of pointcut expressions [KGRX11] or feedback about the implicit effects of base code changes on pointcut evaluation [KS04]. Another aspect-oriented approach addressing the genericity of program elements is *LogicAJ* [RK04]. The research shows the limitations of wildcard matching and demonstrates the benefits of genericity by enhancing pointcuts and advices with meta-variables. LogicAJ was a major source of inspiration for the development of our current approach. Unlike Object Teams, LogicAJ is purely aspect-oriented and does not feature roles or context. Application of genericity is based on general AspectJ constructs, whereas Object Teams focuses on the modularization of roles and its special needs and opportunities.

There are several logic-based approaches in Prolog or Datalog for querying program structure [KHR07, HVdMdV05, JDV03]. In refactoring approaches, as in the *REFACOLA* language [SKvP11], logic programming is also used for program transformation. The approaches share the need to represent imperative program elements in a declarative way to enable them to efficiently query the database. The translation is done by describing AST nodes and relationships in terms of facts and rules. In refactoring, the goal is to make slight changes in program structure while retaining the observable behavior of the program. In contrast to refactoring, in Object Teams the input language differs from the output language and the transformation focuses on replacing and generating new, adapted program elements like classes and methods. The AO enhancement *tracematches* also makes use of free variables, but limits them to utilizing execution history, for which it introduces new pointcut idioms [AAC+05]. The *Alpha* language [OMB05] goes even further by providing Prolog queries over an extensive representation of the program, including dynamic properties of the program execution (making significant concessions regarding efficiency). Both approaches target a more sophisticated aspect-oriented join point selection, rather than modularization and reuse.

Meta-AspectJ (MAJ) is a structured meta-programming tool for generating AspectJ programs using code templates [ZHS04]. Partial artifacts of the code are evaluated to generate either AspectJ or plain Java code. In contrast to our approach, MAJ does not provide expressions to declaratively select program elements; instead references are gained by traversing the Java AST.

An approach to handling context-related concerns is *context-oriented programming* (COP) [HCN08]. In COP, *layers* build first-class entities that may modularize and control context-dependent behavior. Although layers support dynamic dispatching, activation and scoping, there is no intention to support quantification beneath the activation of layers. Layers build an explicit enhancement for classes and are intentionally scattered across modules.

6 Conclusions and Further Work

In this paper, we have discussed the potential for modularization and reuse beyond standard object- and aspect-oriented approaches, and we presented Generic Object Teams, an enhanced version of Object Teams, which is able to support these opportunities. GOT uses generative programming to replace manual search and adaption of program elements with automatic generation. Initially, writing GOT queries and teams requires extra effort, but in return, they provide enhanced reuse capabilities and are more robust towards evolution.

The implementation of a proof-of-concept prototype for GOT is work in progress. So far, we have extended the OT/J language with the required additional language features and build a representation (based on the Eclipse modeling framework (EMF)) of the GOT abstract syntax tree in Prolog facts. We are able to analyze the program structure using Prolog queries. Further work is required to create a more detailed design and implement the automated code transformation mechanism. The combination of different dimensions of reuse, namely inheritance and generics, pose several interesting challenges that will be addressed next. We believe that this combination of features possesses the potential for even more reuse than what we have shown so far.

References

[AAC+05] Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Ondřej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding trace matching with free variables to AspectJ. *SIGPLAN Not.*, 40(10):345–364, 2005.

[EMO04] Michael Eichberg, Mira Mezini, and Klaus Ostermann. Pointcuts as Functional Queries. In *APLAS '04: Proceedings of the 2nd Asian Symposium on Programming Languages and Systems*, volume 3302 of *Lecture Notes in Computer Science*, pages 366–381. Springer, 2004.

[FF00] Robert E. Filman and Daniel P. Friedman. Aspect-Oriented Programming is Quantification and Obliviousness. In *Workshop on Advanced Separation of Concerns, OOPSLA*, 2000.

[GB03] Kris Gybels and Johan Brichau. Arranging language features for more robust pattern-based crosscuts. In *AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 60–69, New York, NY, USA, 2003. ACM.

[HCN08] Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. Context-oriented Programming. *Journal of Object Technology*, 7(3):125–151, 2008.

[Her07] Stephan Herrmann. A precise model for contextual roles: The programming language ObjectTeams/Java. *Appl. Ontol.*, 2(2):181–207, 2007.

[HK02] Jan Hannemann and Gregor Kiczales. Design pattern implementation in Java and aspectJ. *SIGPLAN Not.*, 37(11):161–173, 2002.

[HMK05] Jan Hannemann, Gail C. Murphy, and Gregor Kiczales. Role-based refactoring of crosscutting concerns. In *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*, pages 135–146, New York, NY, USA, 2005. ACM.

[HVdMdV05] Elnar Hajiyev, Mathieu Verbaere, Oege de Moor, and Kris de Volder. CodeQuest: querying source code with datalog. In *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 102–103, New York, NY, USA, 2005. ACM.

[JDV03] Doug Janzen and Kris De Volder. Navigating and querying code without getting lost. In *AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 178–187, New York, NY, USA, 2003. ACM.

[KGRX11] R. Khatchadourian, P. Greenwood, A. Rashid, and G. Xu. Pointcut Rejuvenation: Recovering Pointcut Expressions in Evolving Aspect-Oriented Software. *Software Engineering, IEEE Transactions on*, PP(99):1, 2011.

[KHH+01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An Overview of AspectJ. In *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming*, pages 327–353, London, UK, 2001. Springer-Verlag.

[KHR07] Günter Kniesel, Jan Hannemann, and Tobias Rho. A comparison of logic-based infrastructures for concern detection and extraction. In *LATE '07: Proceedings of the 3rd workshop on Linking aspect technology and evolution*, page 6, New York, NY, USA, 2007. ACM.

[KIL+97] Gregor Kiczales, John Irwin, John Lamping, Jean-Marc Loingtier, Cristina Videira Lopes, Chris Maeda, and Anurag Mendhekar. Aspect-Oriented Programming. In *ECOOP '97: Proceedings of the 11th European Conference on Object-Oriented Programming*, pages 220–242. Springer-Verlag, 1997.

[KS04] Christian Koppen and Maximilian Stoerzer. Pcdiff: Attacking the fragile pointcut problem. In *First European Interactive Workshop on Aspects in Software (EIWAS)*, 2004.

[OMB05] Klaus Ostermann, Mira Mezini, and Christoph Bockisch. Expressive Pointcuts for Increased Modularity. In Andrew Black, editor, *ECOOP 2005 - Object-Oriented Programming*, volume 3586 of *Lecture Notes in Computer Science*, pages 214–240. Springer Berlin / Heidelberg, 2005.

[RK04] Tobias Rho and Günter Kniesel. Uniform Genericity for Aspect Languages. In *Technical Report IAI-TR-2004-4, Computer Science Department III, University of Bonn*. Dec 2004.

[SKvP11] Friedrich Steimann, Christian Kollee, and Jens von Pilgrim. A Refactoring Constraint Language and Its Application to Eiffel. In Mira Mezini, editor, *ECOOP 2011 - Object-Oriented Programming*, volume 6813 of *Lecture Notes in Computer Science*, pages 255–280. Springer Berlin / Heidelberg, 2011.

[SS08] Mirko Seifert and Roland Samlaus. Static Source Code Analysis using OCL. In Jordi Cabot and Pieter Van Gorp, editors, *OCL'08*, 2008.

[ZHS04] David Zook, Shan Shan Huang, and Yannis Smaragdakis. Generating AspectJ programs with Meta-AspectJ. In *Generative Programming and Component Engineering: Third International Conference, GPCE 2004, volume 3286 of LNCS*, pages 1–19. Springer, 2004.

Reduktion von Testsuiten für Software-Produktlinien

Harald Cichos[1], Malte Lochau[2], Sebastian Oster[1], Andy Schürr[1]

[1]Technische Universität Darmstadt
Institut für Datentechnik
{cichos, oster, schuerr}@es.tu-darmstadt.de
[2]Technische Universität Braunschweig
Institut für Programmierung und Reaktive Systeme
lochau@ips.cs.tu-bs.de

Abstract: Eine Software-Produktlinie (SPL) bezeichnet eine Menge ähnlicher Produktvarianten, die bei entsprechend großer Anzahl einen erheblichen Testaufwand verursachen können. Viele modellbasierte SPL-Testansätze versuchen diesen Testaufwand zu verringern, indem Testfälle und Testmodelle aus vorangegangenen Testprozessen ähnlicher Produkte, wenn möglich, wiederverwendet werden. Eine weitere Möglichkeit den Testaufwand zu senken besteht darin, die Anzahl der auf den einzelnen Produkten auszuführenden Testfälle (Testsuite) mittels Testsuite-Reduktionstechniken zu reduzieren. Bisher existierende Verfahren wurden jedoch nicht für den Einsatz im SPL-Kontext entworfen und können daher nicht für jedes Produkt den Erhalt der erreichten Testabdeckung bzgl. eines Überdeckungskriteriums garantieren, wenn Testfälle produktübergreifend wiederverwendet werden. In dieser Arbeit wird diese aus der allgemeinen Testsuite-Reduktion bekannte Anforderung erstmals in erweiterter Form auf den SPL-Kontext übertragen. Darauf aufbauend werden zwei für SPLs ausgelegte Testsuite-Reduktionsansätze vorgestellt, die trotz Reduktion die erreichte Testabdeckung auf jedem Produkt beibehalten. Die Implementierung dieser Ansätze wird auf ein Anwendungsbeispiel angewendet und die Ergebnisse diskutiert.

1 Einleitung

Eine Software-Produktlinie (SPL) bezeichnet eine Menge sich in ihrer Funktionsweise ähnelnder Softwareprodukte [PBvdL05]. Die Ähnlichkeit der Softwareprodukte in dieser Menge resultiert aus der Verwendung einer gemeinsamen Programmplattform, welche mit zusätzlichen Programmteilen variabel erweitert wird. Diese variablen Programmteile realisieren funktionale Eigenschaften, sogenannte Features. Dabei führt eine hohe Anzahl an Features in der Regel zu einer ebenfalls hohen Anzahl an möglichen Produktvarianten. Wenn jede Produktvariante einer SPL ausführlich getestet werden soll, kann das bei großen SPLs, die viele Produkte umfassen, zu einem enormen Aufwand im Testprozess führen [McG01]. Diesen Aufwand versuchen viele modellbasierte SPL-Testansätze zu verringern, indem Testfälle und Testmodelle aus vorangegangenen Testprozessen ähnlicher Produktvarianten, wenn möglich, wiederverwendet werden. Eine weitere Möglichkeit den Aufwand zu verringern besteht darin, die auf den einzelnen Produkten auszuführenden

Testfälle (Testsuite) in ihrer Anzahl zu reduzieren. Existierende Testsuite-Reduktionstechniken sind jedoch nicht für den Einsatz im SPL-Kontext geeignet, da diese nur Testsuiten betrachten, deren Testfälle auf *einem* Produkt ausgeführt werden. In Testsuiten einer SPL können aber Testfälle existieren, die zur Ausführung auf *mehreren* Produkten vorgesehen sind [COLS11], um so die Kosten für Erstellung, Ausführung (z.B. Einsatz als Testorakel) oder Wartung (z.B. Testfallbeschreibung) zu senken. Aufgrund dieser produktübergreifenden Wiederverwendung von Testfällen können bisherige Reduktionstechniken den Erhalt der erreichten Testabdeckung bzgl. eines Überdeckungskriteriums nicht für jede Produktvariante garantieren [YH08].

In dieser Arbeit werden zwei für SPLs ausgelegte Testsuite-Reduktionsansätze vorgestellt, die trotz Reduktion die erreichte Testabdeckung auf jedem Produkt erhalten. Der erste Reduktionsansatz *entfernt* redundante Testfälle aus einer Testsuite für SPLs, wohingegen der zweite Ansatz eine Reduktion erreicht, indem dieser Paare von existierenden Testfällen durch jeweils einen einzelnen, neu erstellten Testfall *ersetzt*. Die Implementierung dieser Ansätze wird auf ein Anwendungsbeispiel angewendet und die Ergebnisse diskutiert. Beide Ansätze berücksichtigen die produktübergreifende Wiederverwendung von Testfällen und können daher selbst bei einer geringen Reduktion in der Testfallanzahl eine starke Reduktion in der Anzahl der Testfallausführungen erreichen. Beide Ansätze tragen dadurch zur Kostensenkung im Testprozess einer SPL bei.

2 Themenverwandte Arbeiten

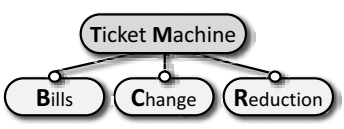
In der Regel nutzen SPL-Testansätze eine der folgenden zwei Strategien oder eine Kombination dieser [ER11]. Die erste Strategie verfolgt das Ziel die Anzahl der zu testenden Produkte zu reduzieren. Dafür wird aus den Produkten einer SPL eine Teilmenge identifiziert, die in Hinblick auf ein bestimmtes Kriterium repräsentativ für alle anderen Produkte ist. Anschließend wird die repräsentative Teilmenge stellvertretend für alle anderen Produkte getestet. Das von der repräsentativen Menge zu erfüllende Kriterium erfordert zumeist eine Überdeckung von Anforderungen [Sch07] oder basiert auf der Auswahl unterschiedlicher Kombinationen von Konfigurationsparametern (Features) der Produkte einer SPL, ähnlich wie beim kombinatorischen Test [OMR10]. Wie groß solch eine repräsentative Teilmenge ist, bzw. ob diese überhaupt in der SPL existiert, hängt maßgeblich von der Implementierung der zu testenden Produkte ab. Lässt sich keine repräsentative Teilmenge ausmachen, müssen im Rahmen der verfügbaren Ressourcen alle Produkte einer SPL einzeln getestet werden. Damit die zur Verfügung stehenden Ressourcen zum Testen einer großen Produktmenge ausreichen, verfolgt die zweite Strategie das Ziel, Testartefakte wie z.B. Testfälle oder Testmodelle für ähnliche Produkte wiederzuverwenden. Diese Ansätze nutzen teilweise Techniken des Regressionstests [ERS10], nur dass diese nicht auf die fortschreitenden Versionen eines einzelnen Produkts angewendet werden, sondern auf unterschiedliche Varianten eines Produkts. In [KBK11, COLS11] werden Ansätze vorgestellt, die für einen Testfall die Produkte ermitteln, auf denen dieser wiederverwendbar ist. In modellbasierten Testansätzen, wie z.B. SCenTED [RKPR05] und CADeT [Oli08] werden nicht die Testfälle, sondern die zur Herleitung von Testfällen genutzten Testmodel-

le wiederverwendet und für jedes Produkt angepasst. Eine ausführliche Zusammenfassung über modellbasierte Testansätze bietet [OWES11]. Von allen SPL-Testansätzen bisher völlig außer Acht gelassen wurde die Möglichkeit, die meist große Anzahl an auszuführenden Testfällen mittels Testsuite-Reduktionstechniken zu senken. Die vorliegende Arbeit nimmt sich als erste dieses Themas an und stellt zwei Ansätze vor, welche die in den Testfällen enthaltenen Informationen mit in den Reduktionsprozess einbeziehen, ähnlich wie in [CH11], um die erreicht Testabdeckung auf jedem Produkt zu erhalten. Eine ausführliche Übersicht über herkömmliche, nicht auf den SPL-Kontext zugeschnittene Testsuite-Reduktionstechniken bietet [YH08].

3 Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe aus den Bereichen „Modellbasiertes Testen“ und „Software-Produktlinien“ anhand eines Anwendungsbeispiels, einer Fahrkartenautomaten-SPL (FA-SPL), eingeführt. Eine Software-Produktlinie (SPL) wird durch eine Menge von Softwareprodukten $PC = \{pc_1, pc_2, \ldots, pc_k\}$ beschrieben, die sich in ihrer Funktionsweise ähnlich sind. Alle Produkte einer SPL besitzen dieselbe Basisimplementierung und damit auch dieselbe Kernfunktionalität. In jeder dieser Produktvarianten wird diese Kernfunktionalität durch *Features* $F = \{f_1, f_2, \ldots, f_n\}$ individuell erweitert bzw. angepasst. Dabei gibt die jeweilige Produktkonfiguration vor, welche Features in einer Produktvariante verwendet werden. Es gilt $PC \subseteq \mathcal{P}(F)$. Beispielsweise bietet die FA-SPL insgesamt 4 Features $F_{FA} = \{TM, B, C, R\}$. Die zulässigen Feature-Kombinationen sind im Feature-Modell nach FODA [KCH$^+$90] in Abbildung 1(a) dargestellt. In Abbildung 1(b) sind die 8 aus dem Feature-Modell ableitbaren, zulässigen Produktkonfigurationen mit den genutzten Features aufgelistet. Es lässt sich erkennen, dass alle Produktkonfigurationen das Feature TM enthalten. Das Feature TM ist für die gemeinsame Kernfunktionalität verantwortlich, die sich wie folgt beschreiben lässt:

Zuerst wird die benötigte Anzahl an Fahrkarten ausgewählt. Im Gegensatz zur Tageskarte und Kurzstrecke lässt sich eine ermäßigte Tageskarte nur auswählen, wenn Feature R in der Produktkonfiguration enthalten ist. Anschließend startet der Bezahlvorgang, bei dem Münzen, und wenn Feature B enthalten ist, auch Scheine als Zahlungsmittel akzeptiert werden. Daraufhin erfolgt die Ausgabe der Fahrkarten. Abhängig von der Anwesenheit des Features C wird im nachfolgenden Schritt das Wechselgeld ausgegeben oder nicht.



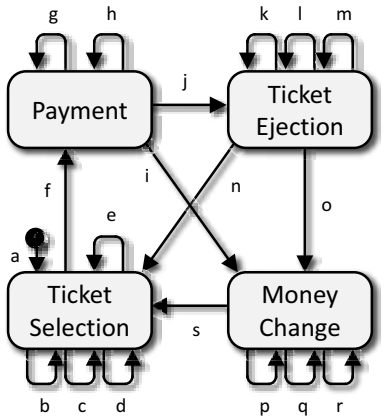
(a) Feature-Modell

	PC	1	2	3	4	5	6	7	8
Features*	TM	1	1	1	1	1	1	1	1
	B	0	0	0	0	1	1	1	1
	C	0	0	1	1	0	0	1	1
	R	0	1	0	1	0	1	0	1

* 1 = enthalten 0 = nicht enthalten

(b) Produktkonfigurationen

Abbildung 1: Zulässige Produktkonfigurationen der FA-SPL



(a) Zustandsautomat

#	F	Signal [Bedingung] / Aktion
a		/ costs=0;paid=0;tShort=0;tDay=0;tRed=0;
b		shortT / tShort++;costs+=2;
c		dayT / tDay++;costs+=4;
d	R	redT / tRed++;costs+=3;
e		cancel / tShort=0;tDay=0;tRed=0;
f		next [costs>0]
g		coin [paid<costs] / paid++;
h	B	bill [paid<costs] / paid+=5;
i		cancel [paid<costs] / tShort=0;tDay=0;tRed=0;
j		[paid>=costs] / paid-=costs;
k		[tShort>0 && tRed==0] / tShort--
l		[tDay>0 && tShort==0 && tRed==0] / tDay--
m	R	[tRed>0] / tRed--
n	¬C	[tDay==0 && tShort==0 && tRed==0] / paid=0;costs=0;
o	C	[tDay==0 && tShort==0 && tRed==0]
p	¬B	[paid>0] / paid--;
q	B	[paid>=5] / paid-=5;
r	B	[paid>0 && paid<5] / paid--;
s		[paid==0] / costs=0;

(b) Beschriftung der Transitionen mit benötigten Features

Abbildung 2: 150%-Testmodell der FA-SPL

Im modellbasierten Test werden die Testfälle, die auf das zu testende Softwareprodukt angewendet werden, aus einem *Testmodell* hergeleitet [UL06]. Das Testmodell beschreibt abstrakt das Verhalten des zu testenden Produkts und entspricht in dieser Arbeit einem Zustandsautomat, der aus Zuständen und Transitionen besteht. Im modellbasierten SPL-Test besitzt jedes zu testende Produkt pc ein zugehöriges Testmodell tm_{pc}. Alle produktspezifischen Testmodelle einer SPL lassen sich zu einer Menge von Testmodellen $TM = \{tm_1, \ldots, tm_k\}$ zusammenfassen. Zwecks einer kompakteren und übersichtlicheren Darstellung bietet es sich an, all diese produktspezifischen Testmodelle in einem sogenannten 150%-Testmodell [GKPR08] zusammenzufassen. Beispielsweise lassen sich alle 8 produktspezifischen Testmodelle der FA-SPL zu dem in Abbildung 2 dargestellten 150%-Testmodell vereinen. In Abbildung 2(a) ist die Struktur des 150%-Testmodells mit allen möglichen Zuständen und Transitionen dargestellt. Jede Transition ist mit einem Kleinbuchstaben gekennzeichnet, der auf die passende Transitionsbeschriftung in Abbildung 2(b) verweist. Um eines der 8 produktspezifischen Testmodelle tm_{pc} aus dem 150%-Testmodell herzuleiten, dürfen nur die Transitionen und Zustände aus dem 150%-Testmodell übernommen werden, die gemäß Produktkonfiguration pc zulässig sind. So darf eine Transition nur dann in ein Testmodell tm_{pc} übernommen werden, wenn die für die Transition benötigten Features (siehe Spalte F in Abbildung 2(b)) in dessen Produktkonfiguration pc enthalten sind. Beispielsweise werden für das Testmodell von Produkt 7 mit der Produktkonfiguration (TM,B,C) alle Transitionen bis auf d, m, n und p aus dem 150%-Testmodell übernommen (siehe Abbildung 3). Somit besteht jedes produktspezifische Testmodell aus einer Teilmenge von Zuständen und Transitionen des 150%-Testmodells.
Als Testendekriterium wird im modellbasierten Test oft ein Überdeckungskriterium (z.B. all-transitions oder all-states) auf das Testmodell angewendet. Dadurch werden Testziele $G = \{g_1, \ldots, g_l\}$ auf dem Testmodell definiert, die von den Testfällen erfüllt werden

PC	Exisitierende Testziele im jeweiligen produktspezifischen Testmodell																		
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
1	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓		✓		✓			✓
2	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓		✓			✓
3	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓			✓	✓			✓
4	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓			✓
5	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓			✓	✓	✓
6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓
7	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓			✓		✓	✓	✓
8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓

Abbildung 3: Alle Produktkonfigurationen der FA-SPL und deren enthaltene Testziele (Transitionen)

müssen. In dieser Arbeit ergibt sich die Menge aller Testziele G aus dem 150%-Testmodell und somit implizit aus allen produktspezifischen Testmodellen. Wird beispielsweise das Überdeckungskriterium *all-transitions* auf das 150%-Testmodell der FA-SPL angewendet, stellt jede Transition eines jeden Produkts ein Testziel dar, das durch mindestens einen Testfall abgedeckt werden muss. Welche Testziele (Transitionen) in welchem produktspezifischen Testmodell enthalten sind, ist in Abbildung 3 dargestellt.
Im modellbasierten Test wird eine Menge von Testfällen $T = \{t_1, t_2, \ldots, t_m\}$ (*Testsuite*) für ein Produkt pc aus dem zum Produkt gehörenden Testmodell tm_{pc} hergeleitet. Jeder Testfall besteht dabei aus einer Sequenz von Eingaben und erwarteten Ausgaben. Werden die Eingaben auf das Testmodell tm_{pc} angewendet, lässt sich ein Testfall in einen Transitionspfad überführen, der aufzeigt, welche strukturellen Testziele vom Testfall in tm_{pc} abgedeckt werden. Nun existieren im modellbasierten SPL-Test mehrere produktspezifische Testmodelle. Um nicht für jede Produktvariante nahezu dieselben Testfälle wie für die Produktvarianten zuvor zu erstellen, wird überprüft, welche der bereits existierenden Testfälle wiederverwendet werden können. Ein Testfall t, hergeleitet aus Testmodell tm_{pc_i} für Produkt pc_i, lässt sich auch für Produkt pc_j wiederverwenden, g.d.w. die Eingabesequenz von t auf Testmodell tm_{pc_j} einen Transitionspfad erzeugt der genau dieselben Testziele abdeckt wie in tm_{pc_i}. In diesem Fall ist t sowohl auf Produkt pc_i, als auch auf pc_j *valide ausführbar*. Eine entsprechende Relation *valid* $\subseteq T \times G \times PC$ ist wie folgt definiert:

valid$(t, g, pc) :\Leftrightarrow$ Testfall t ist *valide ausführbar* auf Produktkonfiguration pc, wenn der Transitionspfad von t auf tm_{pc} das Testziel g erreicht.

Auf welchen Produktkonfigurationen ein Testfall valide ausführbar ist, lässt sich durch den in [COLS11] vorgestellten Ansatz feststellen. Mit diesem Ansatz ist es auch möglich, eine *vollständige SPL-Testsuite* auf effiziente Weise herzuleiten. Eine vollständige SPL-Testsuite besteht aus Testfällen, die auf jedem produktspezifischen Testmodell einer SPL eine vollständige Überdeckung für ein gewähltes Überdeckungskriterium C erreichen. Eine vollständige SPL-Testsuite ist in [COLS11] wie folgt definiert:

Definition 1 *(Vollständige SPL-Testsuite)*
Eine SPL-Testsuite $TS_C = (T', PC')$ mit einer Menge von Testfällen $T' \subseteq T$ und zulässigen Produktkonfigurationen $PC' \subseteq PC$ ist vollständig *bzgl. des Überdeckungskriteriums C, das die Testziele G definiert, gdw.:*

$$\forall g \in G, pc \in PC: \quad (\exists t \in T : \textit{valid}(t, g, pc)) \Rightarrow (\exists t' \in T' : \textit{valid}(t', g, pc))$$

Testfall	Features*				Transitionspfad
	TM	B	C	R	
t1	1	-	-	-	a b f
t2	1	1	0	1	a e d f h j m n
t3	1	-	-	-	a e
t4	1	-	-	-	a c f g g g g j l
t5	1	-	-	1	a d f g g g j m
t6	1	-	0	-	a b f g g j k n
t7	1	-	1	-	a b f g g j k o
t8	1	0	-	-	a b f g i p
t9	1	1	-	-	a b c f h i q
t10	1	1	-	-	a b f g i r
t11	1	-	-	-	a b f i s

* 1 = enthalten 0 = nicht enthalten - = egal

(a) SPL-Testsuite

PC	Testsuite	Anzahl
1	t3, t4, t6, t8, t11	5
2	t3, t4, t5, t6, t8, t11	6
3	t3, t4, t7, t8, t11	5
4	t3, t4, t5, t7, t8, t11	6
5	t3, t4, t6, t9, t10, t11	6
6	t3, t4, t5, t6, t9, t10, t11	7
7	t3, t4, t7, t9, t10, t11	6
8	t3, t4, t5, t7, t9, t10, t11	7
		= 48

(b) Produktspez. Testsuites (minimal)

Abbildung 4: FA-SPL-Testsuite ohne redundante Testfälle

Dabei bezeichnet $PC' \subseteq PC$ die Teilmenge der Produktmenge der SPL, die zum Testen ausgewählt wurde. In dieser Arbeit wird von $PC' = PC$ ausgegangen. Die SPL-Testsuite der FA-SPL ist in Abbildung 4(a) dargestellt. Sie enthält 11 wiederverwendbare Testfälle. Für jeden Testfall sind die Features, die dieser zur validen Ausführung auf einem Produkt benötigt, angegeben. Die von einem Testfall abgedeckten Testziele (Transitionen) sind aus dessen Transitionspfad ermittelbar. Aus dieser SPL-Testsuite lässt sich für jedes Produkt der FA-SPL eine produktspezifische Testsuite bilden, die auf dem dazu passenden Testmodell eine vollständige Überdeckung erreicht (siehe Abbildung 4(b)). Es ist zu beachten, dass ein auf einem Produkt valide ausführbarer Testfall nicht zwangsläufig auch auf diesem ausgeführt werden muss. Beispielsweise ist Testfall $t1$, obwohl dieser auf allen 8 Produkten valide ausführbar ist, auf keinem Produkt zur Ausführung eingeplant.

4 Reduktion von Testsuiten für Software-Produktlinien

Die Testsuite-Reduktion bezeichnet im Allgemeinen einen Vorgang, bei dem die Anzahl der in einer Testsuite enthaltenen Testfälle reduziert wird, um die Kosten bei der Testfallausführung zu senken. In einer SPL-Testsuite kann, aufgrund der möglichen Wiederverwendung von Testfällen, die Anzahl der enthaltenen Testfälle geringer ausfallen als die Anzahl der benötigten Testfallausführungen. Beispielsweise enthält die SPL-Testsuite der FA-SPL in Abbildung 4(a) nur 11 Testfälle, obwohl 48 Testfallausführungen benötigt werden, um auf allen 8 Produkten eine all-transitions-Überdeckung zu erreichen (siehe Abbildung 4(b)). Diese Entkopplung hat auch Auswirkungen auf die Reduktion. So kann auch eine nur geringfügig reduzierte SPL-Testsuite eine große Reduktion in der Anzahl der Testfallausführungen nach sich ziehen. Aus diesem Grund sollte bei der Testsuite-Reduktion im SPL-Kontext nicht die Anzahl der reduzierten Testfälle im Vordergrund stehen, sondern die Anzahl der reduzierten Testfallausführungen.
Existierende Testsuite-Reduktionstechniken gehen von der Annahme aus, dass die Testfälle einer Testsuite nur auf einem *einzigen* Produkt ausgeführt werden. Aus diesem Grund erfüllen diese Ansätze lediglich folgende Forderung:

Bisherige Forderung: *Bei der Reduktion einer Testsuite darf der durch diese Testsuite erzielte Grad der Abdeckung bezüglich eines geforderten Abdeckungskriteriums nicht verringert werden.*

Da im SPL-Kontext die Testfälle nicht nur durch die Menge abgedeckter Testziele charakterisiert sind, sondern auch durch die Menge von Produktvarianten, auf denen diese valide ausführbar sind, muss diese Forderung für SPLs wie folgt erweitert werden:

Neue Forderung: *Bei der Reduktion einer SPL-Testsuite darf der durch diese Testsuite erzielte Grad der Abdeckung bezüglich eines geforderten Abdeckungskriteriums auf den zu testenden Produkten der SPL nicht verringert werden.*

Wird nur der bisherigen Forderung bei der Reduktion einer SPL-Testsuite nachgekommen, kann der Erhalt der erreichten Testabdeckung nicht für jedes Produkt garantiert werden. Beispielweise könnten bisherige Ansätze fälschlicherweise den Testfall $t3$ der FA-SPL entfernen (siehe Abbildung 4), in der Annahme, dass dessen abgedeckten Testziele auch noch von Testfall $t2$ abgedeckt werden. Dass Testfall $t2$ aber nur auf Produkt 6 und nicht wie $t3$ auf allen 8 Produkten ausführbar ist, würde nicht bemerkt werden. Somit wäre das Testziel e auf 7 von 8 Produkten durch keinen Testfall mehr abgedeckt.

4.1 Entfernen redundanter Testfälle

Eine Möglichkeit, die Anzahl der Testfälle zu reduzieren und zugleich der neuen Forderung nachzukommen, besteht im gezielten Entfernen redundanter Testfälle aus einer SPL-Testsuite T. Ein Testfall t ist *redundant*, wenn es einen Testfall $t' \in T$, $t \neq t'$ gibt, der t *subsumiert*. Das heißt, t' deckt alle Testziele von t auf den gleichen Produkten wie t ab. Beispielsweise wird in Abbildung 4(a) der Testfall $t1$ von Testfall $t11$ subsumiert. Wir definieren hierfür eine Subsumptionsrelation $t \preceq t'$ auf Testfallpaaren $t, t' \in T$ eines 150%-Testmodells, einer Menge von Testzielen G und einer Menge von zu testenden Produktkonfigurationen PC wie folgt:

$$t \preceq t' :\Leftrightarrow \forall g \in G, pc \in PC : (\mathit{valid}(t, g, pc) \Rightarrow \mathit{valid}(t', g, pc))$$

In der Regel sind redundante Testfälle in einer Testsuite nicht vollständig auf Subsumption zwischen Paaren von Testfällen zurückführbar. Vielmehr können Subsumptionsbeziehungen auch zwischen *Mengen* von Testfällen der Testsuite bestehen. Beispielsweise wird in Abbildung 4(a) der Testfall $t2$ nicht von einem einzelnen Testfall, sondern von einer Testfallmenge, bestehend aus $t3$, $t5$, $t6$ und $t9$, subsumiert. Daher wird der Subsumptionsbegriff durch die folgende Definition weiter verallgemeinert.

Definition 2 *(SPL-Testsuite Subsumption)*
Eine Testsuite-Subsumptionsrelation $\preceq \subseteq \mathcal{P}(T) \times \mathcal{P}(T)$ *ist eine Ordnung auf Teilmengen von Testfällen einer Testsuite T, sodass:*

$$T' \preceq T'' :\Leftrightarrow \forall g \in G, pc \in PC : (\exists t' \in T' : \mathit{valid}(t', g, pc)) \Rightarrow (\exists t'' \in T'' : \mathit{valid}(t'', g, pc))$$

Durch den nun definierten Subsumptionsbegriff ergibt sich folgender Satz:

Satz 1 *(SPL-Testsuite-Reduktion)*
Wenn $TS' = (T', PC')$ eine vollständige SPL-Testsuite ist, dann ist $TS'' = (T'', PC')$ auch eine vollständige SPL-Testsuite, falls $T' \preceq T''$ gilt.

Beweis: Folgt direkt aus Definition 2.

Folglich kann die Anzahl der Testfälle einer SPL-Testsuite mit Hilfe des eingeführten Subsumptionsbegriffes um redundante Testfälle reduziert werden, ohne die SPL-Abdeckung zu verringern.

4.1.1 Algorithmus zum Entfernen redundanter Testfälle

Im Folgenden wird der in Abbildung 5 dargestellte Algorithmus zum Entfernen aller redundanten Testfälle aus einer SPL-Testsuite näher erläutert. Dieser Algorithmus überprüft jeden Testfall t in einer SPL-Testsuite $splTS$ daraufhin (1.), ob t ein Testziel (3.) auf einem Produkt, auf dem t ausführbar ist (4.), als einziger Testfall abdeckt (5.). Um effizient zu ermitteln, ob noch ein weiterer Testfall existiert der Testziel g auf Produkt pc abdeckt, wird in $splCoverage$ für jedes Testziel auf jedem Produkt protokolliert von wievielen Testfällen dieses abgedeckt wird (max. Speicherverbrauch: $|G| \cdot |PC|$). Testfall t ist nicht redundant (6.), wenn ein Testziel g auf einem Produkt pc nur von Testfall t abgedeckt wird. Sollten alle Testziele, die t abdeckt, auf allen Produkten, auf den t ausführbar ist, auch noch von anderen Testfällen abgedeckt werden, dann ist t redundant und kann aus der Testsuite entfernt werden (8.). Nach dem Entfernen des Testfalls muss $splCoverage$ für die nächste Iteration aktualisiert werden (9.). Wird dieser Algorithmus auf die FA-SPL angewendet (siehe Abbildung 4(a)), werden die Testfälle $t1$ und $t2$ entfernt. Der Algorithmus garantiert aber *nicht* die Herleitung einer minimalen SPL-Testsuite. Dafür muss die größtmögliche Anzahl an redundanten Testfällen aus einer SPL-Testsuite entfernt werden, was NP-schwer ist [HCL+03].

```
1.  for all t in splTS
2.      isRedundant = true;
3.      for all g in t.getSatisfiedGoals()
4.          for all pc in t.getPCs_t_isValidFor()
5.              if splCoverage.isOnlyOneWhichCovers(t,g,pc) then
6.                  isRedundant = false
7.      if isRedundant then
8.          splTS.remove(t)
9.          splCoverage.update()
```

Abbildung 5: Pseudocode zum Entfernen von redundanten Testfällen in einer SPL-Testsuite

4.2 Ersetzen von Testfällen

Enthält eine SPL-Testsuite keine redundanten Testfälle, lässt sich deren Testfallmenge T reduzieren, indem eine Menge von Testfällen $T' \subseteq T$ durch eine Menge neu erstellter Testfälle T'' ($T'' \cap T = \emptyset$) ersetzt wird, sodass gilt:

$$T' \preceq (T \setminus T') \cup T'' \text{ und somit auch } T \preceq (T \setminus T') \cup T''$$

Das heißt, wenn ein Testziel g eines Produkts pc *nur* von Testfällen aus T' abgedeckt wird, dann muss mindestens ein Testfall in T'' existieren, der g auf pc abdeckt. Da häufig noch Testfälle in $T \setminus T'$ existieren, die g in pc ebenfalls abdecken, sinkt die Anzahl der Testziele, die von den neuen Testfällen in T'' zwingend abgedeckt werden müssen. Damit durch das Ersetzen von Testfällen eine Reduktion erreicht wird, muss gelten:

$$\mid T' \mid > \mid T'' \mid \text{ und somit auch } \mid T \mid > \mid (T \setminus T') \cup T'' \mid$$

4.2.1 Algorithmus zum Ersetzen von Testfallpaaren

Im Folgenden wird der in Abbildung 6 dargestellte Algorithmus erläutert, der Testfallpaare in einer SPL-Testsuite durch jeweils einen neu erstellten Testfall ersetzt. Als Ausgangssituation wird angenommen, dass die zu reduzierende SPL-Testsuite keine redundanten Testfälle mehr enthält, da das Entfernen von redundanten Testfällen kostengünstiger ist als das Ersetzen, bei dem immer ein neuer Testfall erstellt wird.
Der Algorithmus versucht jeweils zwei Testfälle $t1$ und $t2$ (Testfallpaar) durch einen neu erstellten Testfall $tnew$ zu ersetzen. Dabei werden nur solche Paare betrachtet (1.), deren zwei Testfälle auf genau derselben Menge an Produkten valide ausführbar sind. Diese Forderung basiert auf der Erfahrung, dass ein neu erstellter Testfall häufig nur dann auf den Produkten des Testfallpaares valide ausführbar ist, wenn die beiden Testfälle des Testfallpaares bereits auf derselben Menge an Produkten valide ausführbar sind. Damit nach dem Ersetzungsschritt weiterhin jedes Testziel auf jedem Produkt abgedeckt wird, werden die Testziele bestimmt (2.), die nicht mehr abgedeckt werden würden, wenn das Testfallpaar ersatzlos aus der Testsuite entfernt wird. Für diese ausgewählten Testziele wird anschließend ein neuer Testfall $tnew$ für die Menge an Produkten gesucht, auf denen das

```
1.  for all pair(t1,t2) in splTS with pair.hasSameValidPCs(t1,t2)
2.      necessaryGoals = splCoverage.findUncoveredGoalsIfRemoved(t1,t2)
3.      tnew = createTestcase(necessaryGoals, getPCsPairIsValidFor(t1,t2))
4.      if tnew exists then
5.          pcs = analyzer.findPCsTestcaseIsValidFor(tnew)
6.          if getPCsPairIsValidFor(t1,t2) is subset of pcs then
7.              splTS.remove(t1)
8.              splTS.remove(t2)
9.              splTS.add(tnew)
10.             splCoverage.update()
```

Abbildung 6: Pseudocode zum Ersetzen von Testfallpaaren in einer SPL-Testsuite

	Testfall	Features*				Transitionspfad
		TM	B	C	R	
ersetzt durch t12	~~t3~~	1	-	-	-	a e
	~~t4~~	1	-	-	-	a c f g g g g j l
	t5	1	-	-	1	a d f g g g j m
	t6	1	-	0	-	a b f g g j k n
	t7	1	-	1	-	a b f g g j k o
ersetzt durch t13	t8	1	0	-	-	a b f g i p
	~~t9~~	1	1	-	-	a b c f h i q
	~~t10~~	1	1	-	-	a b f g i r
	t11	1	-	-	-	a b f i s
	t12	1	-	-	-	a e c f g g g g j l
	t13	1	1	-	-	a c c f g h i q r

* 1 = enthalten 0 = nicht enthalten - = egal

(a) SPL-Testsuite

PC	Testsuite	Anzahl
1	t6, t8, t11, t12	4
2	t5, t6, t8, t11, t12	5
3	t7, t8, t11, t12	4
4	t5, t7, t8, t11, t12	5
5	t6, t11, t12, t13	4
6	t5, t6, t11, t12, t13	5
7	t7, t11, t12, t13	4
8	t5, t7, t11, t12, t13	5
		= 36

(b) Produktspez. Testsuites (minimal)

Abbildung 7: Reduzierte FA-SPL-Testsuite aufgrund ersetzter Testfallpaare

Testfallpaar valide ausführbar ist (3.). Lässt sich ein solcher Testfall $tnew$ erstellen (4.), muss anschließend die Menge an Produkten ermittelt werden (5.), auf denen $tnew$ valide ausführbar ist. Für die Herleitung (3.) und Auswertung (5.) des Testfalls $tnew$ kann beispielsweise der Ansatz aus [COLS11] benutzt werden. Wenn der neue Testfall $tnew$ mindestens auf denselben Produkten valide ausführbar ist wie das Testfallpaar (6.), kann Testfall $tnew$ das Testfallpaar ersetzen ohne die SPL-Abdeckung zu senken. Anschließend kann das Testfallpaar entfernt (7.-8.) und der Testfall $tnew$ in die SPL-Testsuite aufgenommen werden (9.). Nachdem die SPL-Testsuite verändert wurde, muss die SPL-Abdeckung $splCoverage$ für die nächste Iteration aktualisiert werden. Durch jeden Ersetzungsschritt reduziert sich die Anzahl der in der SPL-Testsuite $splTS$ enthaltenen Testfälle um 1. Wird dieser Algorithmus auf die FA-SPL angewendet (siehe Abbildung 7), wird das Testfallpaar $t3$ und $t4$ durch $t12$ und das Testfallpaar $t9$ und $t10$ durch $t13$ ersetzt. Dadurch ergibt sich, ausgehend von einer redundanzfreien FA-SPL-Testuite, eine Reduktion in der Anzahl der Testfälle von 9 auf 7 und eine Reduktion in der Anzahl der Testfallausführungen von 48 auf 36 (vergleiche Abbildung 4(b) mit 7(b)). Die stark reduzierte Anzahl der Testfallausführungen um 25% lässt sich darauf zurückführen, dass beide Testfallpaare auf vielen Produkten der FA-SPL zur Ausführung vorgesehen waren.

4.3 Diskussion

Im Rahmen dieser Arbeit wurden die beiden vorgestellten Ansätze in Azmun [Has], ein Framework für den modellbasierten Test, realisiert und auf weitere, aus 150%-Testmodellen hergeleitete Testsuiten angewendet. Die Auswertung der untersuchten SPL-Testsuiten ergab, dass die Anzahl an Testfallausführungen durch Ersetzen von Testfällen umso stärker reduziert werden kann, je *größer* die gemeinsame Menge von Produkten ist, auf der die zu ersetzenden Testfälle zur Ausführung eingeplant sind. Folglich lässt sich der in Abschnitt 4.2.1 vorgestellte Algorithmus in seiner Effizienz steigern, wenn die Paare zuvor noch nach der Anzahl an Produkten, auf denen diese beide zur Ausführung vorgesehen sind, sortiert werden.

Bei der Reduktion durch Ersetzen ist zu beachten, dass die Produktmenge, auf der sich ein neu erstellter Testfall ausführen lässt, häufig (abhängig vom 150%-Testmodell) umso kleiner ausfällt, je mehr Testziele dieser zwingend abdecken muss. Sollte aber die Produktmenge eines solchen Testfalls nicht eine Obermenge der Produktmenge sein, auf welcher die zu ersetzenden Testfälle valide ausführbar sind, ist eine Subsumption nicht mehr möglich. Aus diesem Grund sollte bei der Erstellung eines Testfalls darauf geachtet werden, dass von diesem nur die Abdeckung der zwingend notwendigen Testziele gefordert wird. Eine Methode zur Bestimmung dieser Testziele wurde in Abschnitt 4.2 vorgestellt.

Beide vorgestellten Ansätze reduzieren eine SPL-Testsuite produktübergreifend. Jedoch ist es auch möglich, jede produktspezifische Testsuite einer SPL einzeln zu reduzieren, solange die in Abschnitt 4.1 neu eingeführte Forderung bzgl. der SPL-Abdeckung erfüllt wird. Dieses Vorgehen wirkt sich allerdings negativ auf die Wiederverwendbarkeit der Testfälle aus, sowohl beim Entfernen als auch beim Ersetzen. So sinkt beim Entfernen von Testfällen zwar die Anzahl der Ausführungen, aber nicht zwangsweise die Anzahl der in einer SPL-Testsuite enthaltenen Testfälle. Beispielsweise könnte in jeder produktspezifischen Testsuite ein anderer redundanter Testfall entfernt werden. Beim Ersetzen kommt hinzu, dass die neuen Testfälle für jede produktspezifische Testsuite erneut erstellt werden, was einerseits zu hohen Kosten in der Testfallgenerierung führt und andererseits zu evtl. nicht wiederverwendbaren Testfällen. Durch Letzteres steigt die Anzahl der in der SPL-Testsuite enthaltenen Testfälle, was wiederum zu höheren Wartungskosten bei den Testfällen führt (z.B. Aktualisierung der Testfallbeschreibung). Testfälle sollten auch nur dann zwecks Kosteneinsparung ersetzt werden, wenn Grund zur Annahme besteht, dass durch das Erstellen *und* die Ausführung der neu erstellten Testfälle weniger Kosten als durch die Ausführung der zu ersetzenden Testfälle entstehen.

Beide vorgestellten Reduktionsansätze garantieren nur den Erhalt der erreichten Überdeckung einer SPL-Testsuite, aber nicht den Erhalt der erreichten Gesamtfehlersensitivität einer Testsuite. Aus diesem Grund sollten beide Ansätze nur dann angewendet werden, wenn eine SPL-Testsuite aufgrund von Kostengründen reduziert werden *muss* und dabei keine Rücksicht auf den Erhalt der Gesamtfehlersensitivität genommen werden kann.

5 Zusammenfassung

In dieser Arbeit wurden zwei neue Testsuite-Reduktionsansätze für SPLs vorgestellt. Im Gegensatz zu bisher existierenden Testsuite-Reduktionstechniken berücksichtigen beide Ansätze, dass im SPL-Kontext Testfälle existieren können, die zur Ausführung auf mehreren Produkten vorgesehen sind. Dadurch ist es mit beiden Ansätzen möglich, die Anzahl der in einer SPL-Testsuite enthaltenen Testfälle produktübergreifend zu reduzieren ohne dabei die zuvor erreichte Testabdeckung auf einem einzigen Produkt zu verringern. Aufgrund dieser produktübergreifenden Testsuite-Reduktion bleiben die verbleibenden Testfälle in einer SPL-Testsuite wiederverwendbar. Für zukünftige Arbeiten ist eine Evaluation beider Ansätze an industriellen Beispielen geplant, die zeigen soll, ob diese skalieren und in der Praxis anwendbar sind.

Literatur

[CH11] H. Cichos und T. Heinze. Efficient Test Suite Reduction by Merging Pairs of Suitable Test Cases. In J. Dingel und A. Solberg, Hrsg., *Models in Software Engineering - Workshops and Symposia at MODELS 2010*, Jgg. 6627 of *LNCS*, Seiten 244–258, 2011.

[COLS11] H. Cichos, S. Oster, M. Lochau und A. Schürr. Model-based Coverage-Driven Test Suite Generation for Software Product Lines. In *Proc. of MoDELS'2011*, Jgg. 6981 of *LNCS*, Seiten 425–439, 2011.

[ER11] E. Engström und P. Runeson. Software Product Line Testing - A Systematic Mapping Study. *Information and Software Technology*, 50:1098–1113, January 2011.

[ERS10] E. Engström, P. Runeson und M. Skoglund. A Systematic Review on Regression Test Selection Techniques. *Information & Software Technology*, 52(1):14–30, 2010.

[GKPR08] H. Grönniger, H. Krahn, C. Pinkernell und B. Rumpe. Modeling Variants of Automotive Systems using Views. In *Modellierung*, 2008.

[Has] S. Haschemi. Azmun - The Model-Based Testing Framework. Online: 2011-10-05.

[HCL+03] H. S. Hong, S. D. Cha, I. Lee, O. Sokolsky und H. Ural. Data Flow Testing as Model Checking. In *Proc. of ICSE'03*, Seiten 232–242. IEEE, 2003.

[KBK11] C.H.P. Kim, D.S. Batory und S. Khurshid. Reducing Combinatorics in Testing Product Lines. In *Proc. of AOSD'2011*, Seiten 57–68. ACM, 2011.

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak und A. S. Peterson. Feature-Oriented Domain Analysis (FODA). Bericht, Carnegie-Mellon University, 1990.

[McG01] J. D. McGregor. Testing a Software Product Line. Bericht CMU/SEI-2001-TR-022, Carnegie Mellon, Software Engineering Institute, 2001.

[Oli08] E. M. Olimpiew. *Model-Based Testing for Software Product Lines*. Dissertation, George Mason University, 2008.

[OMR10] S. Oster, F. Markert und P. Ritter. Automated Incremental Pairwise Testing of Software Product Lines. In *Proc. of SPLC'2010*, Seiten 196–210, 2010.

[OWES11] S. Oster, A. Wübbeke, G. Engels und A. Schürr. Model-Based Software Product Lines Testing Survey. In J. Zander, I. Schieferdecker und P. Mosterman, Hrsg., *Model-based Testing for Embedded Systems*. CRC Press/Taylor&Francis, 2011.

[PBvdL05] K. Pohl, G. Böckle und F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[RKPR05] A. Reuys, E. Kamsties, K. Pohl und S. Reis. Model-based System Testing of Software Product Families. In *Proc. of CAiSE'2005*, Seiten 519–534, 2005.

[Sch07] K. Scheidemann. *Verifying Families of System Configurations*. Dissertation, TU Munich, 2007.

[UL06] M. Utting und B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., 2006.

[YH08] S. Yoo und M. Harman. Regression Testing Minimization, Selection and Prioritization: A Survey. *Software Testing, Verification and Reliability*, 2008.

Erfahrungsberichte

Softwaretest-Umfrage 2011

Erkenntnisziele, Durchführung und Ergebnisse

Mario Winter
Fachhochschule Köln
mario.winter@fh-koeln.de

Karin Vosseberg
Hochschule Bremerhaven
karin.vosseberg@hs-bremerhaven.de

Andreas Spillner
Hochschule Bremen
andreas.spillner@hs-bremen.de

Peter Haberl
ANECON Software Design&Beratung GmbH
peter.haberl@anecon.com

Abstrakt: Um nachhaltig die "New Driving Force" zu sein, muss Software auch höchsten professionellen Qualitätsansprüchen genügen können. Hier spielen Qualitätssicherung und Test eine herausragende Rolle, was in der Forschung mittlerweile in erfreulicher Weise gewürdigt wird. Aber wie sieht der Alltag der Softwareentwicklung aus? Wie versuchen heutige software-entwickelnde Firmen die Qualitätsansprüche zu erfüllen? Im Mai 2011 wurde im deutschsprachigen Raum eine anonyme Online-Umfrage zum Thema „Softwaretest in der Praxis" durchgeführt. Dieser Beitrag beschreibt die Erkenntnisziele, die zugrundeliegenden Annahmen, die Durchführung sowie ausgewählte Ergebnisse der Umfrage. Ein Teil der Fragen wurde aus einer Umfrage zu „Prüf- und Testprozesse in der Softwareentwicklung" aus dem Jahr 1997 übernommen. Mit den Ergebnissen beider Umfragen lässt sich zeigen, was sich im Bereich der Qualitätssicherung bei der Softwareentwicklung verändert hat. Zusätzlich beantworten wir, ob „Outsourcing“ und „MBT“ sich bereits einen Platz in der Test-Praxis erobert haben.

1 Einführung

Die IKT-Branche erlebte in den letzten 10 Jahren wie kaum eine andere Branche einschneidende Veränderungen und Umbrüche. Seien es die Folgen der geplatzten Dot-Com-Blase auf die Unternehmen, die Forderung nach Agilität an die Prozesse oder die Heterogenität und Komplexität der Produkte und Plattformen – all diese Randbedingungen führten zu immer mehr „Notfall-Operationen am offenen Herzen“. Hierzu gehören plötzliche Unternehmenszusammenschlüsse, Outsourcing-Aktivitäten, kurzfristige Prozess- oder Produkt-Änderungen sowie die in der Regel damit einhergehenden einschneidenden Veränderungen im Personalbereich.

Vor diesem Hintergrund interessieren uns die Auswirkungen auf Organisation, Vorgehensweisen, Methoden und Techniken im Bereich Software-Qualitätssicherung und -Test, gerade weil Testen eine der wichtigsten analytischen Maßnahmen der Praxis zur Sicherung der Qualität von Software ist. Dies wird in der Forschung mittlerweile in erfreulicher Weise gewürdigt und schlägt sich in zahlreichen Fachbeiträgen nieder. Neue

Trends und Überlegungen, wie z.B. Test Driven Development, agiles Testen, exploratives Testen und modellbasiertes Testen, sind zu erkennen.

Näher betrachtet handelt es sich bei nicht wenigen solcher Vorschläge zwar eher um „Alten Wein in neuen Schläuchen“ ([Wi09]), nichts desto trotz bieten solche Trends und Überlegungen sowohl Chancen als auch Herausforderungen für die Qualitätssicherung. Außerdem ist ja alter Wein nicht unbedingt schlechter Wein, und so gehören viele der lange bekannten grundlegenden Vorgehensweisen der Software-Qualitätssicherung wie z.B. Reviews, statische Analysen oder dynamisches Testen zu den effektivsten Mitteln zur Sicherstellung und Überprüfung der Software-Qualität. Aber was davon hat wirklich Einzug in die Praxis gehalten? Wie sieht der Test-Alltag heute in den Unternehmen im deutschsprachigen Raum aus? Hat sich dort in den letzten Jahren etwas verändert? Wie versuchen software-entwickelnde Firmen im Spannungsfeld aus Kostendruck, time-to-market und Technologie-Drift die Qualitätsansprüche zu erfüllen?

Diese und ähnliche Fragen beschäftigen die „Test-Community“ seit langem. Zwar werden auf Fachkonferenzen und Workshops zum Thema Qualitätssicherung sowie in Fachzeitschriften zunehmend Berichte aus der Praxis einzelner Unternehmen vorgestellt. Diese sind aber in der Regel auf bestimmte Aspekte fokussiert und spiegeln oft lediglich erste Erfahrungen bei der Einführung neuer Verfahrensweisen wider. Anfang 2011 haben wir daher beschlossen, diesen Fragen in der Breite und Tiefe nachzugehen. Es ergab sich eine Kooperation der Hochschulen Bremen und Bremerhaven, der Fachhochschule Köln, der ANECON Software Design und Beratung G.m.b.H., dem German Testing Board e.V. (GTB) und dem Swiss Testing Board (STB). Im Mai 2011 wurde dann im deutschsprachigen Raum die anonyme Online-Umfrage zum Thema „Softwaretest in der Praxis" durchgeführt. Ein Teil der Fragen wurde aus einer 1997 durchgeführten Umfrage zu „Prüf- und Testprozesse in der Softwareentwicklung" übernommen [MWA98]. Mit den Ergebnissen beider Umfragen lässt sich zeigen, was sich im Bereich der Qualitätssicherung bei der Softwareentwicklung verändert hat, und ob aktuelle Trends sich bereits einen Platz in der Praxis erobert haben.

2 Ausgangslage, Erkenntnisziele und Aufbau des Fragebogens

Zur Beleuchtung der Ausgangslage, aus der heraus die Umfrage initiiert und durchgeführt wurde, formulieren wir als erstes die Erkenntnisziele unserer Umfrage. Danach skizzieren wir andere Studien bzw. Erhebungen mit ähnlicher Fragestellung und Zielgruppe. Den Schluss dieses Kapitels bilden Hinweise zu Aufbau und Inhaltsbereichen unseres Fragebogens.

2.1 Erkenntnisziele

In einer ersten Diskussionsrunde wurden zunächst die Interessen der Kooperationspartner erhoben. Daraus wurden mögliche Erkenntnisziele formuliert und abgeglichen. Im Endergebnis sollte die Umfrage folgende Thesen untermauern oder abweisen:

1. Das Qualitätsbewusstsein hat zugenommen. Maßnahmen der Qualitätssicherung werden frühzeitig im Rahmen der Softwareentwicklung integriert.
2. Durch agile Vorgehensmodelle wird Testen in den Entwicklungsprozess integriert, aber explizite Aufgabenbereiche wie Integrations- und Systemtest sowie eine methodische Testfallentwicklung verschwinden.
3. Tester ist als eigenständiges Berufsbild akzeptiert und genießt ein gewisses Ansehen.
4. Testautomatisierung fokussiert auf den Unit-Test und wird wenig systematisch durch die Entwickler durchgeführt.
5. Einfache, wiederholt durchzuführende Testtätigkeiten werden zunehmend an externe Dienstleister ausgelagert.
6. Modellbasiertes Testen (MBT) ist in speziellen Domänen (z.B. Automotive, eingebettete Systeme) der Standard und auf der Schwelle zur Marktreife, wird aber „in der Breite“ noch wenig eingesetzt.

2.2 Andere Studien und Erhebungen

In den zurückliegenden Jahren wurden mehrere Studien und Erhebungen in den Themenfeldern Software-Qualität, Qualitätssicherung, Test etc. durchgeführt und veröffentlicht. Einige davon sind auf Einzelaspekte wie z.B. die Testautomatisierung oder das Testmanagement beschränkt, einige gehen mehr in die Breite. Manche wurden einmalig z.B. zur Initiierung von Forschungs- oder Entwicklungsprojekten durchgeführt, andere werden periodisch wiederholt, um Trends im Zeitverlauf widerspiegeln zu können. Durchführungstechnisch sind einige Studien als offene oder geschlossene Umfragen konzipiert, andere basieren auf Interviews, wieder andere auf Feldbeobachtungen in Unternehmen bzw. Organisationen. Gemäß unserer Ausgangsfragestellung haben wir zunächst Studien mit ähnlicher Fragestellung und Zielgruppe untersucht.

Bereits aus dem Jahr 1997 stammt die in Deutschland durchgeführte Umfrage „Prüf- und Testprozesse in der Softwareentwicklung“ [MWA98][Mü99]. Ziel der Arbeit war, die tatsächliche Ausgestaltung der Prüf- und Testprozesse der Unternehmen zu erheben und zu eruieren, wie die Unternehmen diese hinsichtlich ihrer Leistungsfähigkeit und der entstehenden Kosten einschätzen. Der papierbasierte Fragebogen wurde an 504 Unternehmen versendet, wovon 73 in auswertbarer Form geantwortet haben. Die Ergebnisse zeigten eine teilweise große Lücke zwischen dem damaligen „State of the Art“ und den in der Praxis vorzufindenden Prüf- und Testprozessen auf. Diese wurden in nahezu allen betrachteten Unternehmen als „verbesserungswürdig“ eingestuft. Auch war in vielen Unternehmen keine selbstständige Qualitätssicherungsabteilung realisiert und der Einsatz von Methoden, Techniken sowie Werkzeugen ungenügend ausgeprägt und zu wenig an Unternehmens- und Produkt-Spezifika adaptiert.

Aus dem Jahr 2004 stammt eine in Form von Interviews und einer Online-Umfrage einmalig durchgeführte Studie zum Stand der Praxis von Software-Tests und deren Automatisierung in Deutschland [AOS04]. Befragt und zum (kleinen) Teil interviewt wurden

123 Unternehmen als „typische Vertreter kleiner und mittelständischer Unternehmen (KMU)“. Die Autoren ziehen als Fazit aus der Studie „dass im Bereich von Testen und Testautomatisierung ein großes Potential zur Verbesserung der Software, der Methoden und zur Kosteneinsparung für die Unternehmen sichtbar ist und dass dieses erst bei wenigen Unternehmen durchgängig etabliert ist. Ein Großteil der Möglichkeiten, welche derzeit existieren, wird noch nicht ausgeschöpft und bietet daher große Chancen, ökonomischere und stabilere Softwareprodukte herzustellen.“ [AOS04].

Auch im Jahr 2004 wurde an der Universität Stuttgart eine Umfrage zur analytischen Qualitätssicherung in der Industrie durchgeführt [WS05]. Die teilweise in Form strukturierter Interviews, teilweise mit Fragebogen erhobenen Daten ließen zwar im Vergleich zu [Mü99] einen Trend zur Steigerung der QS-Aktivitäten erkennen, können jedoch aufgrund der geringen Anzahl von 8 Teilnehmern nicht als repräsentativ gelten. Gleichwohl sind die Ergebnisse insbesondere wegen des expliziten Bezuges zu [Mü99] für die vorliegende Umfrage interessant.

Im Jahr 2006 führte die Firma Software Quality Lab in Österreich erstmalig eine in die Breite zielende Umfrage zum Qualitätsmanagement in IT-Unternehmen durch. Aktuell verfügbar sind die Ergebnisse der Studie des Jahres 2008 [Sq08], wobei weder die Größe der Stichprobe noch deren Verteilung über Branchen, Firmengrößen etc. angegeben sind. Auch wurden die Antworten lediglich rein deskriptiv ausgewertet. Der Großteil der Fragen dieser Studie bezieht sich auf Kenngrößen zum Testprozess, nur einige wenige Fragen erheben Informationen zu verwendeten Methoden, Techniken und Werkzeugen.

Die Zielsetzung des seit 2009 jährlich durchgeführten „World Quality Reports“ der Firmen Capgemini, HP und Sogeti [CSH11] ist, „*über Branchen- und Ländergrenzen hinweg den aktuellen Zustand von Anwendungsqualität und -test zu überprüfen. Jedes Jahr betrachtet dieser Report gegenwärtige Trends im Bereich Software-Qualität und analysiert aufkommende Tendenzen, welche die Zukunft des Software-Tests beeinflussen können.*“ (übersetzt aus [CSH11]). Die Studie von 2011 bedient das Management und geht nicht auf konkrete Verfahren und Techniken ein. Befragt wurden ca. 1.200 CEOs, CIOs, CFOs, IT-Direktoren und -Manager sowie QA-Manager von Unternehmen aus unterschiedlichen Branchen weltweit. Dazu gehörten einige aus dem deutschsprachigen Raum, ohne deren Antworten gesondert auszuwerten.

Auch seit 2009 wird jährlich die Breitenstudie „Testing Trends & Benchmarking Schweiz“ der Firma SwissQ aufgelegt [Sw11]. Neben ca. 200 ausgefüllten Fragebogen wurden persönliche Interviews von über 20 Entscheidungsträgern im Bereich Testing ausgewertet. Neben neuen Trends wurden auch altbekannte Themen untersucht. SwissQ resümiert: „*Leider haben wir als Test Community immer noch nicht geschafft, diese befriedigend umzusetzen. Neue Ansätze in Test Automatisierung und neue Technologien stellen daher klare Anforderungen ans Testen: Schneller, flexibler, agiler!*“ [Sw11].

Einmalig und im Rahmen der Forschungsverbünde FLEXI (www.flexi-itea2.org) und NESSI (www.nessi-europe.com) wurde im Jahr 2009 projektbezogen eine Tiefenumfrage zu Aspekten des Tests in agilen Projekten durchgeführt [CSP10]. Als Forschungsfrage formulieren die Autoren „*Ist es möglich, die hauptsächlichen*

Diskrepanzen zwischen den bekannten und den in der Praxis bevorzugten Test-Praktiken zu identifizierenen und aufzulisten, die als Hindernisse für praktizierende Software-Tester angesehen werden könnten?“ [CSP10]. Befragt wurden ca. 80 Mitarbeiter der an den beiden Forschungsverbünden beteiligten Unternehmen bzw. Forschungseinrichtungen. Die Ergebnisse aus dem deutschsprachigen Raum werden nicht gesondert dargestellt; darüber hinaus ist die Stichprobe sowohl seitens der Branchen als auch der geringen Größe nicht repräsentativ.

Vor diesem Hintergrund konzentrieren wir uns auf den deutschsprachigen Raum (D A CH). Wir haben den Anspruch, eine sowohl in die Breite als auch in die Tiefe gehende repräsentative Erhebung durchzuführen. Der Umfrage sollte insbesondere eine Stichprobe angemessenen Umfang zugrunde liegen, deren Verteilung die Größe und Branchenstruktur der hier agierenden Unternehmen annähert. Darüber hinaus suchen wir den direkten Vergleich zur Studie von 1997 [MWA98][Mü99] – immerhin sind seither fast 15 Jahre ins Land gegangen. Ein Teil der von uns eingesetzten Fragen wurde daher direkt aus dieser Umfrage übernommen, um Veränderungen und neue Trends im Bereich der Qualitätssicherung bei der Softwareentwicklung aufzeigen zu können. Was hat sich geändert? Was nicht? Was sollte also noch in der Praxis passieren? Was braucht die Praxis heute?

2.3 Aufbau des Fragebogens

Der Fragebogen gliedert sich in drei Bereiche, wobei einzelne Fragen bzw. Fragegruppen teilweise aufeinander aufbauen bzw. einander ausschließen. Den Anfang des ersten Bereichs mit unternehmens- und prozessorientierten Fragen bilden allgemeine Fragen zum Unternehmen (Größe, Branche, Anteil der Beschäftigten in der Softwareentwicklung und –test, Projekt-Dauer und Volumen, …). Es folgen Fragen zur methodischen Vorgehensweise bei der Softwareentwicklung, unterteilt nach phasenorientierten oder agilen Vorgehensmodellen. Fragen zu Risiko- und Qualitätsmanagement sowie zum Testprozess schließen diesen Bereich ab.

Im zweiten Bereich werden Fragen zur Durchführung der Software-Qualitätssicherung gestellt. Hierunter fallen solche zu den verwendeten statischen und dynamischen Testverfahren ebenso wie Fragen zu den in den einzelnen Teststufen sowie zum Testmanagement eingesetzten Werkzeugen.

Der dritte Bereich erhebt zunächst Informationen zu Qualifizierung und Weiterbildung im Softwaretest, wobei ein Schwerpunkt auf das international eingesetzte Ausbildungsschema ISTQB® Certified Tester [Is11] gelegt wird. Angaben zum Probanden selbst (Berufserfahrung, Ausbildung, Geschlecht, …) beenden den Fragebogen.

3 Annahmen und Durchführung

Gemäß der oben beschriebenen Ausgangslage und Zielsetzung trafen wir einige Annahmen, welche die Basis zur Konzeption der Umfrage darstellten. Die Durchführung orientierte sich an den Empfehlungen aus [Ka05].

3.1 Annahmen

Zunächst einmal gehen wir davon aus, dass die in der Praxis gelebten Praktiken der Qualitätssicherung bzw. des Softwaretests sich in den vergangenen zwei Dekaden nicht wesentlich verändert haben, so dass überhaupt Querbezüge zu früheren Studien sinnvoll sind. Hier gaben die oben referenzierten vorhandenen Studien sowie ein Blick in entsprechende Konferenz- und Zeitschriftenbeiträge genügend Evidenz.

Weiterhin setzen wir voraus, dass sich auch die Verteilung der Software entwickelnden (und testenden) Firmen nach Branche, Unternehmensgröße etc. nicht wesentlich von jenen seit dem Jahr 1997 unterscheidet. Auch für diese Annahme gaben die oben referenzierten Studien sowie die vom statistischen Bundesamt 2008 überarbeitete Klassifikation der Wirtschaftszweige (WZ 2008) eine gewisse Evidenz.

Auch wenn der Stichprobenumfang der Umfrage von 1997 relativ gering war, glauben wir, dass die dortigen Ergebnisse als repräsentativ gelten können, da sie auf den Tagungen der GI-FG „Test, Analyse und Verifikation von Software“ und anderen Konferenzen vorgestellt und vom Fachpublikum zustimmend aufgenommen wurden. Evidenz dafür ist darüber hinaus gegeben, wenn die „Basisangaben“ bei der aktuellen Umfrage bei größerem Stichprobenumfang vergleichbar ausfallen.

3.2 Durchführung

Der Stichprobenumfang sollte im Vergleich zu 1997 (74 papierbasierte Antworten) viel höher ausfallen, da wir eine hohe Zuverlässigkeit und Präzision anstreben, also einen kleinen Stichprobenfehler und eine hohe Vertrauensgrenze. Zur präziseren Festlegung ist zunächst einmal die Grundgesamtheit zu identifizieren. Im Mai 2011 gab es allein in Deutschland 846.500 Beschäftigte im Wirtschaftszweig „Information und Kommunikation“ [Bu11], im Jahr 2009 gab es dort 124.275 umsatzsteuerpflichtige Betriebe. Annäherungsweise wird im Weiteren daher von einer unbegrenzten Grundgesamtheit ausgegangen.

Bei Annahme einer Normalverteilung ergibt sich gemäß des zentralen Grenzwert-Theorems die Mindest-Stichprobengröße n_{min} bei sigma=1,96 (95% fallen unter die beobachtete Verteilung, Stichprobenfehler e=0,05), einer erwarteten relativ hohen Varianz der Population von sig_p=0,7 und dem überwiegenden Einsatz von Fragen mit Antworten auf Ordinal- oder höherem Skalenniveau nach [Ka05] zu

$$n_{min} = sigma^2 \cdot sig_p{}^2 / e^2, \text{ i.e. } 1{,}96^2 \cdot 0{,}7^2 / 0{,}05^2 = 752{,}9$$

Es mussten also mehr als 750 Antworten erreicht werden. Mit dem Ziel, möglichst eine Zufallsstichprobe zu erheben, wurde auf eine breite Streuung der Information geachtet: Neben der Auslage von Werbeprospekten auf einschlägigen, sowohl entwicklungs- als auch testbezogenen Fachkonferenzen, Annoncen in Zeitschriften und dem Web, wurden für eine bilaterale direkte Ansprache die eMail-Verteiler der Kooperationspartner und Unterstützer verwendet. Insgesamt wurden 5000 Werbeprospekte ausgelegt bzw. per Post versendet und über 3000 eMail-Adressaten angeschrieben.

Die Dauer der Umfrage wurde auf einen Monat begrenzt (Mai 2011). Sehr schnell wurde das von den Partnern aus der Industrie geäußerte Interesse nach personalisierten Antworten zugunsten einer anonymen Umfrage zurückgenommen. Einerseits aus Datenschutzgründen, andererseits erschien das Erreichen der erforderlichen Mindest-Stichprobengröße gefährdet, da die Rücklaufquote bei personalisierten Umfragen erfahrungsgemäß nur ca. 10% beträgt.

Die Umfrage sollte Informationen auf operativer, projektbezogener und Unternehmensebene erheben. Um den Umfang des Fragebogens insbesondere für Manager erträglich zu halten, wurde ein rollenspezifischer Aufbau gewählt. Hierbei wurden die drei Rollen „Tester“ (Projektleiter, QS-Beauftragte, Testmanager und Tester), „Entwickler“ (Business Analyst, Entwickler, Mitarbeiter aus Betrieb&Support und andere Mitarbeiter) sowie „Management“ (Executive und mittleres Management) definiert.

4 Ergebnisse

Insgesamt haben 1623 Personen an der Umfrage teilgenommen. Die Rolle „Tester“ ist mit 1008 Teilnehmenden vertreten, aber auch bei Vertretern der Rolle „Entwickler“ mit 394 und der Rolle „Management“ mit 221 Fragebogen gab es eine gute Resonanz. 50% der Teilnehmenden haben den umfangreichen Fragebogen bis zum Ende durchgearbeitet. Die Mehrzahl kam aus Deutschland (77%) gefolgt von der Schweiz (13%) und Österreich (10%). Fast ein Drittel der Teilnehmer arbeitet in mittelgroßen Firmen mit 101-1.000 Mitarbeitern, aber auch kleine Firmen (11-100 Mitarbeiter) sind mit einem Viertel der Teilnehmer sehr gut vertreten. Große und sehr große Unternehmen nahmen mit ca. 8% bzw. 13% an der Umfrage teil. Insgesamt lässt sich im Vergleich zur Studie von 1997 [MWA98] ein leichter Trend hin zu kleineren Firmengrößen ausmachen. Weitere Details hierzu sind auf der Internetseite der Umfrage zu finden [Um11].

Erste quantitative Ergebnisse der Umfrage wurden bereits veröffentlicht [Sp11], eine etwas umfangreichere Darstellung dieser Ergebnisse bietet [Ha11]. Die folgenden Abschnitte fassen nun unsere Ergebnisse bezüglich der beiden Fragen „Was hat sich im Bereich der Qualitätssicherung bei der Softwareentwicklung seit 1997 verändert?“ und „Konnten sich aktuelle Trends einen Platz in der Praxis erobern?“ zusammen, gegliedert nach den sechs in Abschnitt 2.1 formulierten Thesen.

4.1 Qualitätsbewusstsein und frühe Qualitätssicherung

In der Umfrage von 1997 wurde ein phasenorientiertes Vorgehen vorausgesetzt. Um einen Vergleich der Ergebnisse zu ermöglichen, fragten wir nach dem Vorgehensmodell. Gut ein Viertel aller Befragten ist agil unterwegs, über die Hälfte gibt an, dass ihre Software eher nach einem phasenorientierten Modell entwickelt wird. Knapp 20% der Antwortenden haben kein explizites Modell zur Entwicklung ihrer Software.

Bei den agilen Vorgehensweisen ist Scrum der eindeutige Favorit. Allerdings wiesen viele Teilnehmer darauf hin, dass in der Praxis eher eine Mischung unterschiedlicher agiler Vorgehensweisen verwendet wird, als ein einziges Verfahren in Reinkultur.

Unter den phasenorientierten Modellen wird das allgemeine Phasenmodell nach Barry Boehm von gut einem Drittel der Projekte verwendet. Über 20% verwenden ein eigenes Modell. Über 10% entwickeln nach dem W-Modell, bei dem der Entwicklungs- und Testprozess parallel durchgeführt werden. Das Wasserfallmodell mit der sequenziellen Abfolge der Entwicklungsphasen wird noch von 10% der Befragten eingesetzt, knapp gefolgt vom V-Modell XT. Interessant ist die Aufteilung der Vorgehensmodelle innerhalb der einzelnen Branchen (Abb. 1), welche unsere „Erwartungen“ an den Einsatz der Modelle bis auf eine Ausnahme (Medizintechnik) bestätigt hat.

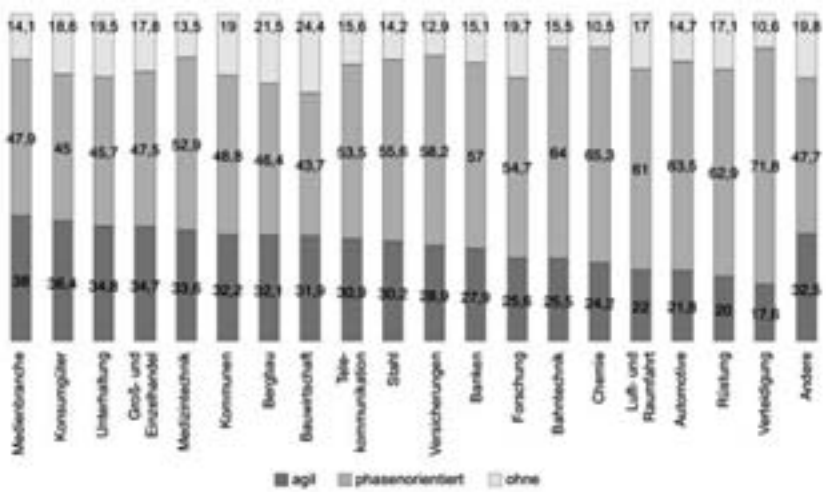


Abb. 1. Verteilung der Vorgehensmodelle über die Branchen

Diejenigen, die ein phasenorientiertes Modell befolgen, wurden gefragt, in welchen Phasen sie Maßnahmen zur Qualitätssicherung durchführen. Es ergibt sich der in Abb. 2 gezeigte Vergleich der 1997 und 2011 zustimmenden und ablehnenden Antworten.

In Projekten mit phasenorientierten Vorgehensmodellen hat der geplante Anteil der Qualitätssicherungsmaßnahmen im Vergleich zur Umfrage 1997 zwar leicht zugenommen (1997; 21,6%, 2011: 22,9%), ist aber nach wie vor auf die späten Phasen der Softwareentwicklung konzentriert. So setzen immer noch fast 40% der Befragten in der Vorstudie überhaupt keine Maßnahmen zur Qualitätssicherung ein.

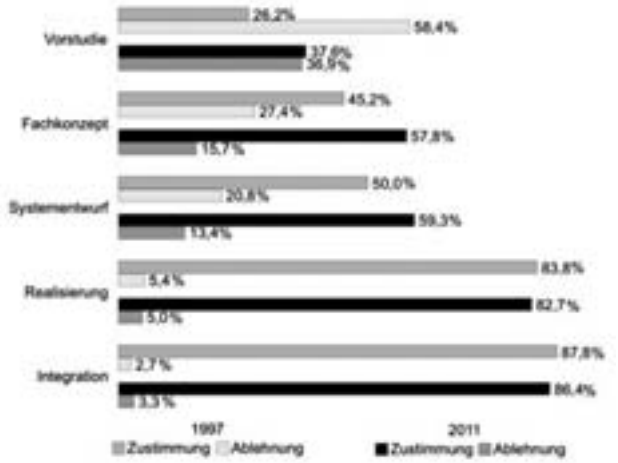


Abb. 2. Verteilung der QS-Maßnahmen über die Entwicklungsphasen

1997 wurde auf einer Likert-Skala die Zustimmung oder Ablehnung bzgl. des Einsatzes einzelner Testverfahren erhoben. 2011 wurde „binär" erhoben, ob die Verfahren eingesetzt werden, wobei Mehrfachnennungen möglich waren. Zum Vergleich wurden in den Daten von 1997 alle Angaben bis einschließlich „teils/teils" als Einsatz der Verfahren interpretiert. Sowohl 1997 (93,4%) als auch 2011 (82,1%) steht bei den Black-Box-Testentwurfsverfahren der Funktionale Test im Mittelpunkt. In 2011 werden aber zunehmend Anwendungsfälle (Use-Cases) zur Spezifikation der Testfälle herangezogen (77,1%). Dies verwundert natürlich nicht, denn 1997 war die Modellierung mit UML noch nicht sehr verbreitet. Die klassischen Verfahren wie Äquivalenzklassenbildung (1997: 47,8%, 2011: 60,9%) und Grenzwertanalyse (1997:72,1%, 2011: 67,6%) sind in ihrer Bedeutung eher gleich geblieben. Der Zufallstest (1997: 63,2%, 2011: 38,9%) hat abgenommen. Die prozentuale Verteilung weiterer Testverfahren in 2011 sind unter [Um11] im Untermenue Testverfahren aufgeführt.

These 1 kann nach diesen Ergebnissen nur schwach bestätigt werden.

4.2 Agile Vorgehensmodelle und methodisches Testen

Um die Auswirkung agiler Vorgehensweisen auf die Qualitätssicherung zu untersuchen, fragten wir zunächst nach Teststufen wie Komponententest, Integrationstest und Systemtest. Während bei phasenorientierten Projekten über 90% Teststufen unterscheiden, sind dies bei agilen Projekten nur 78,5% und bei solchen ohne explizites Vorgehensmodell nur 51%. Bezüglich der eingesetzten Testverfahren inkl. der Durchführung von Reviews fallen nur die Projekte ohne explizites Vorgehensmodell deutlich zurück.

Als weiteren Indikator für These 2 untersuchten wir auch den Bereich Risikomanagement. Fast 70% der Teilnehmer gaben an, in den Projekten Risikomanagement durchzuführen. Am häufigsten wurde hierbei genannt, Risiken mehrmals bei Bedarf (38%) oder vorab geplant bei wichtigen Projektereignissen (28%) zu überprüfen. Regelmäßig wö-

chentlich oder monatlich wird nur bei 7% bzw. 16% der Teilnehmer eine Risikoüberprüfung vorgenommen. Dies lässt den Schluss zu, dass die Risiken durch Projektereignisse (wie Treffen des Leitungsgremiums, Phasenwechsel etc.) oder auf Nachfrage durch den Projektmanager bzw. Linienverantwortlichen zu einer Neubewertung gebracht werden. Tabelliert man die Angaben zum Risikomanagement über denen zum Vorgehensmodell zeigt sich, dass nur unwesentlich mehr Anwender phasenorientierter Modelle Risikomanagement betreiben als solche agiler Vorgehensmodelle (78% zu 63%, Tab. 1). Im Gegensatz dazu setzen lediglich 34% derjenigen, die „Kein explizites Vorgehensmodell" angaben, Risikomanagement ein.

	Mit Risikomanagement	Ohne Risikomanagement	„weiß nicht"
Phasenorientiertes Vorgehensmodell	398 (78,5%)	65 (12,8%)	44 (8,7%)
Agiles Vorgehensmodell	129 (63,2%)	55 (27,0%)	20 (9,8%)
Kein explizites Vorgehensmodell	42 (33,9%)	50 (40,3%)	32 (25,8%)

Tab. 1 Vorgehensmodell vs. Risikomanagement

Insgesamt kann auch These 2 nicht erhärtet werden – der Einsatz agiler Vorgehensmodelle geht nicht zu Lasten des methodischen Testens. Wichtig ist, dass überhaupt ein Vorgehensmodell befolgt wird.

4.3 Tester als eigenständiges Berufsbild

Bezüglich These 3 fragten wir zunächst, ob eine selbstständige Qualitätssicherungsgruppe im Unternehmen existiert. Während dies 1997 nur von 32% bejaht wurde, liegt dieser Anteil 2011 schon bei über 66%. Allerdings existiert ein Unterschied zwischen phasenorientiert und agil arbeitenden Unternehmen: Erstere bejahen zu 72%, letztere nur zu 63%, während Antwortende aus Unternehmen, die kein explizites Vorgehensmodell angeben, nur zu 51% zustimmten.

1997 haben hauptsächlich Entwickler die Tests durchgeführt während ausgebildete Tester eher eine untergeordnete Rolle spielten. Wie sieht dies nun heute aus? Immer noch sehen 86% der Teilnehmer die Entwickler in der Testdurchführung, aber mit einem großen Sprung von 26% auf 77% haben die Tester deutlich aufgeholt. Die Testfälle werden bei 78% (1997: 35%) der Befragten von Testern, zu 56% (1997: 60%) von Entwicklern und zu 40% (1997: 54%) von Mitarbeitern der Fachabteilungen erstellt. Externe Berater werden hierzu nur von 7% (1997: 10%) der Unternehmen eingesetzt. Dies zeigt deutlich, dass die methodische Testfallerstellung durch hierfür qualifizierte Tester zugenommen hat.

In über 43% der Unternehmen gibt es bereits individuelle Qualifizierungsprogramme im Bereich Qualitätssicherung, fast 90% der Befragten kennen in diesem Zusammenhang das Ausbildungsschema zum ISTQB® Certified Tester [Is11]. These 3 konnte somit erhärtet werden.

4.4 Testautomatisierung

Zur Beleuchtung von These 4 fragten wir nach dem Grad der Testautomatisierung in den Teststufen. Abb.3 zeigt, dass dieser deutlich auf den Unit-Test fokussiert ist.

Außerdem interessierte die Automatisierung bei der Testfallerstellung. Testfälle werden überwiegend frei verbal/textuell beschrieben, insbesondere in der Gruppe Entwickler. In der Gruppe Tester ist die Dokumentation der Testfälle durch standardisierte Formulare gleich auf. Ein Viertel der Entwickler erstellt die Testfälle während der Durchführung, von den Testern sind es knapp über 10%. Die werkzeuggestützte Erfassung von Testfällen hat sich noch nicht umfassend etabliert. 50% der Tester und 30% der Entwickler verwenden eine Werkzeugunterstützung zur Erfassung der Testfälle. Entwickler haben erhebliche Ressentiments (45%) gegenüber den Werkzeugen zur Erfassung und Verwaltung von Testfällen. These 4 wurde also (leider) erhärtet.

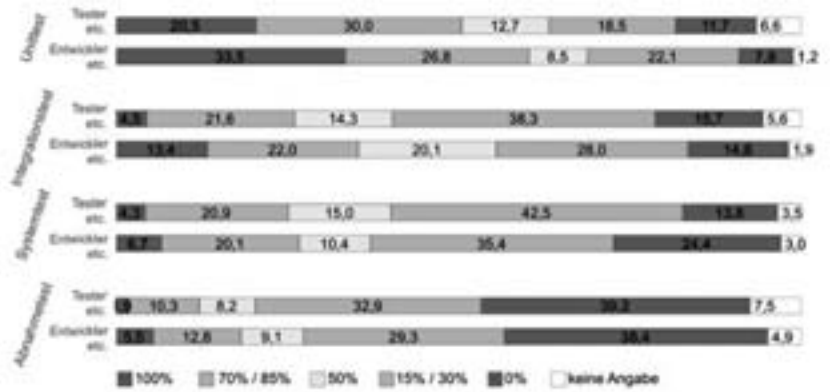


Abb. 3. Grad der Testautomatisierung in den Teststufen

4.5 Auslagerung von Testtätigkeiten

Ein Trend zur Auslagerung gemäß These 5 hat sich nicht bestätigt. So setzen nur 15% der Befragten externe Dienstleister für die Testdurchführung ein. Nur bei 6% der Befragten sind externe Dienstleister verantwortlich für die Qualitätssicherung in Projekten.

4.6 Modellbasiertes Testen (MBT)

Formale Sprachen oder modellbasierte Erstellung oder Dokumentation von Testfällen haben in der Praxis noch keinen hohen Stellenwert und werden eher wenig eingesetzt. Der Einsatz ersterer wurde von über 60%, derjenige letzterer von über 70% der befragten Tester und Entwickler verneint. Immerhin gaben 46% derjenigen, die hinsichtlich möglicher Themen einer Expertenausbildung zum Test antworteten, das Modellbasierte Testen als interessanten Themenbereich an.

5 Ausblick und Dank

Der Beitrag beleuchtet die Ausgangslage und Zielsetzung sowie die Durchführung und ausgewählte Ergebnisse unserer anonymen Online-Umfrage. Die große Resonanz während der Umfrage selbst zeigte einmal mehr die hohe praktische Relevanz der Fragestellung. Das umfangreiche Rohdatenmaterial wird uns noch geraume Zeit beschäftigen. So planen wir tiefer gehende multi-variate Auswertungen bezüglich unserer Hypothesen. Auch die Einbeziehung von Ergebnissen der oben skizzierten früheren Studien aus dem Themenfeld in die Trend-Analysen ist angedacht.

Wir danken den Förderern und Unterstützern: ANECON Software Design und Beratung G.m.b.H., German Testing Board e.V., Swiss Testing Board, Arbeitskreis Software-Qualität und Fortbildung e.V., Austrian Testing Board, dpunkt Verlag GmbH, Softwareforen Leipzig GmbH, GI-FG Test, Analyse und Verifikation von Software.

6 Literatur

[AOS04] Armbrust, O.; Ochs, M.; Snoek, B.: Stand der Praxis von Software-Tests und deren Automatisierung. Fraunhofer IESE-REPORT NR. 093.04/D, 2004.

[Bu11] Statistik der Bundesagentur für Arbeit, Datenstand: September 2011. Bundesagentur für Arbeit, Nürnberg, September 2011.

[CSH11] Capgemini, sogeti, HP: World Quality Report. Capgemini, 2011.

[CSP10] Causevic, A.; Sundmark, D.; Punnekkat, S.: An Industrial Survey on Contemporary Aspects of Software Testing. Proc. Third International Conference on Software Testing, Verification and Validation, Paris, 2010.

[Ha11] Haberl, P.; Spillner, A.; Vosseberg, K.; Winter, M.: Umfrage 2011: „Softwaretest in der Praxis“. dpunkt.verlag, Heidelberg, November 2011.

[Is11] ISTQB® Certified Tester Ausbildungsschema. International Software Testing Qualifications Board, 2011. http://www.istqb.org, in Deutschland: http://www.german-testing-board.info

[Ka05] Kasunic, M.: Designing an Effective Survey. Software Engineering Institute Handbook CMU/SEI-2005-HB-004, Pittsburgh, 2005.

[Mü99] Müller, U.: Prüf- und Testprozesse in der Softwareentwicklung, Dissertation, Universität zu Köln, Shaker Verlag, 1999.

[MWA98] Müller, U.; Wiegmann, T.; Avci, O.: »State of the Practice« der Prüf- und Testprozesse in der Softwareentwicklung, Ergebnisse einer empirischen Untersuchung bei deutschen Softwareunternehmen, 1998. (http://systementwicklung-archiv.bibliothek.informatik.uni-koeln.de/28//)

[Sp11] Spillner, A.; Vosseberg, K.; Winter, M.; Haberl, P.: Qualitätssicherung im Wandel: Was sich in den letzten 15 Jahren getan hat. OBJEKTspektrum, 6/2011, S. 46-51.

[Sq08] Software Quality Lab: QM Studie 2008. Software Quality Lab, Linz, 2008.

[Sw11] SwissQ: Testing Trends & Benchmarking Schweiz. SwissQ, Zürich, 2011.

[Um11] Internet-Seite der Umfrage http://www.softwaretest-umfrage.de/

[Wi09] Winter, M.: Modellbasierter Test – Alter Wein in neuen Schläuchen? In: Proc. of 10. SQC-Kongress, SQS, Düsseldorf, 2009.

[WS05] Wetzel, M.; Siegwart, K.: “Empirische Untersuchung der analytischen Qualitätssicherung in der Industrie“. Softwaretechnik-Trends, Bd. 25 Heft 3, GI, August 2005.

Model-Based Analysis of Design Artefacts Applying an Annotation Concept

Daniel Merschen*, Yves Duhr**, Thomas Ringler**, Bernd Hedenetz**, Stefan Kowalewski*
*Embedded Software Laboratory, RWTH Aachen University
Aachen, Germany
{merschen | kowalewski}@embedded.rwth-aachen.de

Group Research & Advanced Engineering, Daimler AG**
Boeblingen, Germany
{yves.duhr | thomas.ringler | bernd.hedenetz}@daimler.com

Abstract: In automotive software development, dependencies among process artefacts, i. e. requirements, implementation and test cases, are often not obvious. This causes time-intensive manual analysis efforts to incorporate changes during software evolution. Therefore, automated tool support is essential to establish an efficient change management during the software life cycle.

This paper presents a model-based concept which integrates the artefacts themselves as well as development-related meta information about them to establish both functional and process-related artefact analyses. To this end, we represent them as models in the Eclipse Modeling Framework and apply model transformations to support different kinds of automated analyses.

1 Introduction

Within the automotive industry market trends like functional innovations and an increasing number of car lines lead to a dynamic life cycle of software applications, e. g. the need to incorporate new features or changes of requirements late in the development process. Hence, a special software development approach is necessary which is better suitable to handle such dynamics exploiting systematic reuse and variability concepts.

To tackle these challenges, Daimler's Electrical and Electronics division for the body and comfort domain has been following the model-based approach with Matlab/Simulink [Matb] for several years [WDR08]. As Figure 1 visualises the product line is first modelled as a feature tree following the Feature-Oriented Design Analysis (FODA) [KCH+90]. This feature tree supports the engineer in an early stage by documenting the different dependencies. Second, the artefacts, i. e. a system specification (requirements), a Simulink model (implementation model) and a test specification, are built accordingly. However, as described by [TDH11] the current approach has to be extended in order to efficiently manage the increasing complexity of future functions, especially to support software *evolution*. One intuitive step towards this evolution support is already realised by hand-written documentation which is added to the subsystems which implement special features

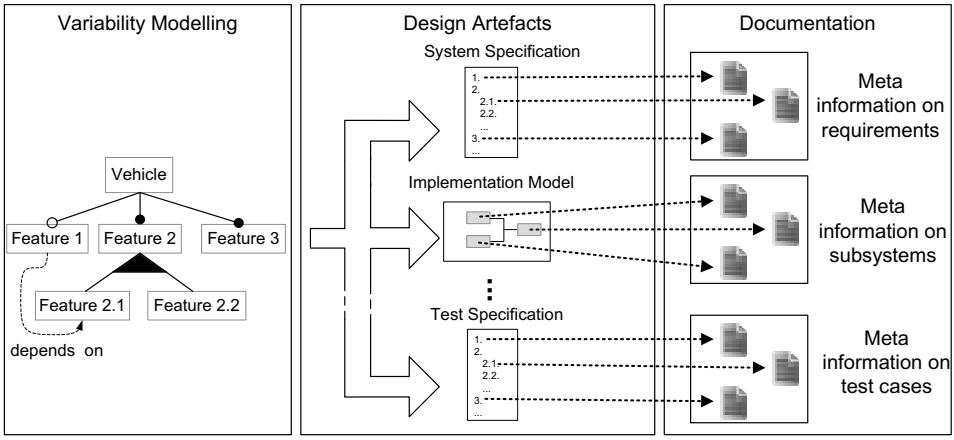


Figure 1: Workflow of the process described by Thomas et al. [TDH11]

in the Simulink model respectively the requirements or test cases (cf. right part of Figure 1). Nevertheless, due to the complexity of the product line artefacts and the documentation style (free text) maintenance and evolution are still time-consuming. This phenomenon is well-known in literature. As [Som07] points out the costs of maintenance and evolution of long-lifetime software systems exceed the development costs by factor 3-4. According to [MD08] 50% of development costs are needed just to understand legacy code.

To our mind, it is therefore important to provide engineers with automated analyses to manage software evolution. These analyses should first consider functional and structural aspects of artefacts to handle the artefact complexity, e. g. to create views on a Simulink model which visualise model elements depending on a given signal. Second, process-related and variability-related issues have to be taken into account, e. g. to identify reasons for changes of artefacts during life cycle. In [PMT+10] and [MPBK11] we focus on structural and functional analyses of single and multiple artefacts, while this paper will describe a concept to integrate process-related information into the different artefacts to widen the set of possible analyses by process-related ones. To this end, we have elaborated a general artefact integration concept as well as an annotation concept to capture meta information. This concept was inspired by the model-based development of embedded on-board software at Daimler AG and is explained in Section 2. That is why we mainly focus on the applied tools (Matlab/Simulink [Matb] and IBM Rational DOORS [IC]) here. However, the concept is held abstract enough to be generalisable for the use in other tools and application areas as well which will be explained in Section 2.4. In Section 3 we discuss the benefits based on a case study of experienced Daimler engineers. We describe related work in Section 4 before summarising the paper and giving an outlook on the next steps in Section 5.

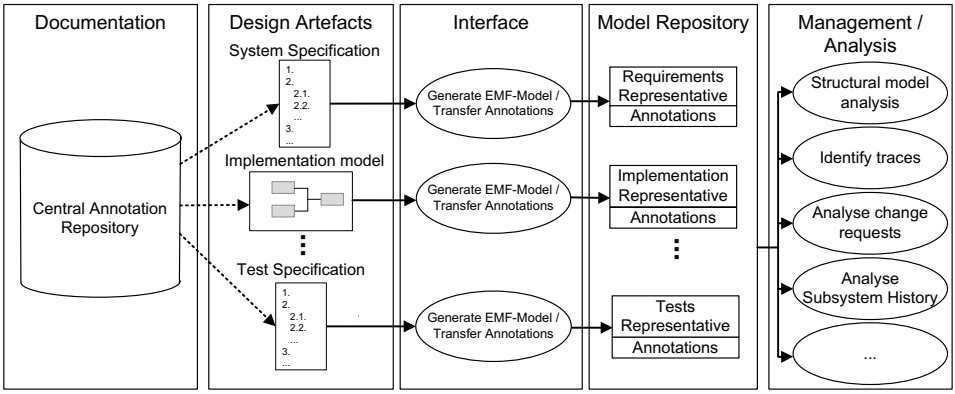


Figure 2: Conceptual overview to establish multi-artefact analyses consider also meta information

2 Approach

To tackle the challenges mentioned in Section 1 it is necessary to add automatically analysable process-related meta information to the artefacts and to connect the different tools (cf. [BFH+10]). We do this by following the concept description in Figure 2. To integrate meta information we elaborated a concept based on a central annotation repository. Section 2.1 will describe how this concept works. Subsequently, we integrate the different artefacts into one model repository. This step is described by the middle three parts of Figure 2 and will be explained in detail in Section 2.2. Based on integrated artefacts including meta information different kinds of analyses are planned to be realised (right part of Figure 2) one of which is outlined in Section 2.3. Finally, we will describe how the presented concept can be generalised in Section 2.4

2.1 Documentation of Design Artefacts

In order to efficiently incorporate late changes and to support evolution we would like to be able to analyse artefacts not only with respect to their structural and functional construction but also consider meta information about the development process like changes of subsystems due to change requests, bug reports or combinations of them. In model-based development with Matlab/Simulink such information is often specified as free text within special version information blocks from TargetLink [dSp] libraries. Hence, these blocks do not implement functionality but just answer the purpose of documentation. Figure 3 exemplifies the content of a version information block. Currently, diverse meta information is specified that way within these blocks, e. g. time stamp and author of the meta information itself, the version, the information is relevant for, the ID of a change request that initiated the change and the production line(s) the subsystem is relevant for.

Figure 3: A screenshot of a version information block for the documentation of subsystems

One approach to integrate this free text information into analyses would be to define a unique syntax for the free text so that a parser could collect the relevant information. In earlier work [MPBK11] we present an analysis which follows this approach. However, as, in practice, every user uses his own syntax parsing this text is error-prone. A further problem of using version information blocks is that the information can only be retrieved manually by stepping into the subsystems recursively. There is currently no way to receive an overview over subsystems and their documentation or realisation status, which complicates, for example, an estimation of the (remaining) effort to incorporate a change request. Furthermore, artefact elements that are related to the same functionality are likely to be annotated with similar text. The different syntax will complicate the identification of these relationships. Uniquely defined annotations, however, could facilitate it and hence, also support the establishment of traceability. In the following section we will define what is meant by an annotation in our context and explain which requirements an annotation concept has to meet to be useful and applicable in industrial practice. In Section 2.1.2 we will outline how we realised these requirements.

2.1.1 Requirements on the Annotation Concept

To address these challenges we suggest annotations on artefacts. To our mind, an annotation concept should (1) facilitate automated analyses, (2) ensure uniqueness of annotations, (3) allow artefact-comprehensive annotating and (4) allow for an overview over all annotations.

To facilitate automated analyses it is important to ensure that there is a unique syntax for each semantics and, hence, to limit free text specification as much as possible. This requirement cannot be covered by the current free text annotation. Hence, we provide a type for each annotation such that an annotation can be defined as a tuple <TYPE, VALUE>.

In order to ensure that annotation types are unique and artefact-comprehensive our concept provides a *central* management, i. e. there is a central repository of all available types together with a description about their meaning (cf. left part of Figure 2). Each tool should then connect to this central repository to retrieve these types such that the user can only select the desired type and then specify the desired annotation value (cf. second part of Figure 2).

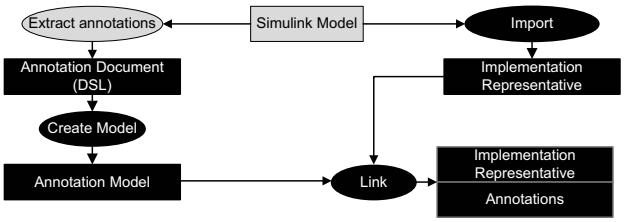


Figure 4: Import of an annotated Simulink model into our framework

2.1.2 Realisation

Up to now, we have focussed on the annotation of Simulink models. Each block in a Simulink model contains a parameter called *UserData*, which we use to annotate it with the help of the *set_param* command of the Matlab API [Mata]. To prevent the user from arbitrarily defining annotation types himself which would limit automatising analyses the annotation is performed via a Matlab script which reads out the available types from a database (cf. left part of Figure 2) such that the user can select one. Subsequently, the user is prompted to specify the annotation value. This workflow is repeated until the user finishes annotating. Finally, the script writes the annotation to the Simulink model with the above mentioned *set_param* method.

2.2 Artefact Integration

In order to establish analyses including multiple artefacts it is necessary to connect the different tools, i. e. to join the artefacts in one common repository. With this approach we can avoid the problem of tool-dependent model analyses. The corresponding process is visualised by the middle three parts of Figure 2. We currently import Simulink models and a csv-export of the requirements in IBM Rational DOORS with Xtext [EFd], a framework to create domain-specific languages. The resulting models are models of the Eclipse Modeling Framework (EMF) [EFa], which we call *representatives* of the original artefacts and which are stored in the model repository.

Besides the artefacts themselves, the annotations, which were added to them as described in Section 2.1, have to be transferred to the corresponding representative in the model repository. Otherwise the management component described later on could not consider them for analyses. Up to now, we realised this step for Simulink models. As Matlab/Simulink does not store them in clear text within the file we follow the process in Figure 4 to transfer the annotations from Matlab/Simulink to our model repository. The grey parts describe activities which take place within Matlab/Simulink while the black ones are performed within our framework. First of all, the annotations are extracted out of the Simulink model with Matlab methods, i. e. with a Matlab script which reads all available annotations and writes them into a text file following a domain-specific language (DSL) (cf. Listing 1).

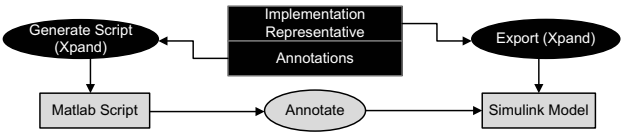


Figure 5: Export of an annotated representative implementation model of our framework to Matlab/Simulink

```
1  AnnotationModel AnnotModel {
2    annotations {
3      Annotation 0 {
4        path     "ADJ_tl06/ADJ/Subsystem/Subsystem/Subsystem/ADJ_Function/SoCProcessing"
5        type     "VERSION"
6        value    "1.0"
7        author   "JT"
8        timestamp     "2010/04/15 08:09:08"
9      },
10     [...]
11 }
```

Listing 1: An excerpt of an annotation document exported via a Matlab script

For each annotation the script captures the author, the time stamp, type and value as well as the path to the annotated block. The path describes the navigation through the Simulink model via the subsystems to the block starting at the root system (analogous to a path to a file in a file system). It is necessary for the further process to find the corresponding block in the representative implementation model within the model repository of the framework.

Next, the file is parsed by an Xtext parser leading to an EMF-based annotation model, which is stored within the model repository and directly linked to the implementation representative by a model transformation (cf. Figure 2).

Results of an artefact analysis might also be stored as annotations within the representatives. For instance, one could be interested in all subsystems of a Simulink model that were affected by a change request. In this case one could append a suitable annotation to the respective subsystems of the representative. In such cases it also has to be possible to transfer such annotations from the model repository to the Simulink model. Figure 5 describes the steps to be followed for that purpose.

To this end, we apply XPand [EFc] to (1) export the implementation representative to Matlab/Simulink and (2) to generate an annotation script for it from the annotation model. Subsequently, we run the generated script to transfer the annotations to the exported Simulink model.

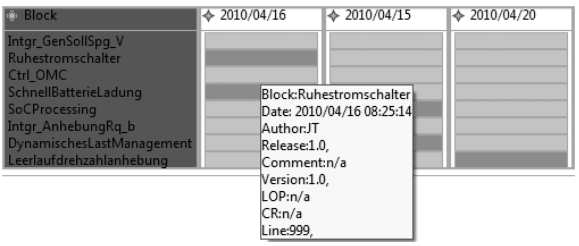


Figure 6: Resulting table of a history analysis visualised in a graphical editor of the GMP

2.3 Artefact Management and Analysis

In the right part of Figure 2 we mention some examples for analyses which base on the model repository. For details and examples about structural model analyses we refer the reader to previous work [MPBK11, PMT+10]. To estimate the potential of the presented annotation concept for automated, process-related model analysis we focus here on the analysis of the subsystem history which generates a tabular overview over subsystems of a Simulink model and the dates of their changes in a graphical editor of the Graphical Modelling Project (GMP) [EFb]. Furthermore, for each date of change detailed information about the change is collected and displayed to the user (cf. Figure 6). Previously, we retrieved this information from the version information blocks mentioned in Section 2.1 by parsing the free text with an Xtext parser. However, when we applied the concept on real productive models this approach turned out not to be scalable enough for industrial practice due to too different kinds of syntax used for the free text. Now we have applied the described annotation concept, i. e. we added the information of the version information blocks with annotations on the affected subsystems. As expected, the model transformation now works well such that we can assume the concept to be a benefit for automated analysis and even for linking artefacts based on annotations.

2.4 Generalisation

Up to now, we focussed primarily on the integration of annotations into the implementation model exemplified by Matlab/Simulink. However, this approach is generally not restricted to Matlab/Simulink although some modification will be necessary to transfer the concept on other tools as well.

With respect to the implementation model we assume that the basic functionality of other tools is comparable to Matlab/Simulink and that models can be transformed into Simulink models with equivalent semantics. This assumption, of course, would have to be evaluated thoroughly before the approach is transformed to a different tool platform.

Concerning requirements and test cases the situation is different. As we can only import csv files into the model repository the syntax is fixed while the semantics is not. Hence,

the import will not have to be adapted if the applied tool can export the requirements as a csv file. However, as the semantics of the resulting requirements respectively test model is unknown, the model transformations have to be adapted. This step is supposed to be time-intensive.

Moreover, we have to consider how the central repository can be accessed by the applied tools. Up to now, we have established the repository access in Matlab/Simulink. However, as the repository is realised by a relational database the available annotation types can be retrieved via usual SQL queries. Hence, the concept can be integrated into each tool which is able to communicate with a relational database.

Due to the abstractness of the concept it also becomes applicable to other application areas, e. g. to support software project planning. To this end, it has to be further extended to capture more artefacts in the model repository, e. g. whole descriptions of change request, bug reports and feature models which have to be managed by project leaders.

3 Practical Experience

In order to estimate the benefits of the presented concept for the model-based development at Daimler the concept was evaluated by three Daimler engineers with 6, 10 and 15 years of experience in software development of embedded systems (max. 10 years experience in *model-based* development). To do so, they focussed on the implementation model in Matlab/Simulink of a real-world embedded software product line. The application of the concept on other artefacts like requirements and test cases has not been evaluated yet. The evaluation criteria were (1) error-proneness , (2) benefits for automatising single artefact as well as multi-artefact analyses including meta information, (3) generalisability, (4) barriers to overcome before going active.

Error-proneness One of the targets of the presented annotation concept is to reduce the use of free text in order to prevent errors during automated model analysis. The concept can be expected to meet this requirement. The assumption is based on the analysis presented in Section 2.3 which was performed on a Simulink model with 1,000 blocks and a subsystem hierarchy depth of 8. Applying the former concept which parsed the free text of the version information blocks lead to frequent parsing errors. These are caused by the fact that many engineers work at the same Simulink model and everyone uses his own syntax to specify meta information. The same model was now successfully annotated using the presented annotation concept and analysed with an adapted model transformation which should be applicable to every implementation model representative.

Expected benefits for model analyses The concept's potential to support automated model analyses with respect to meta information was estimated as high. As the annotation syntax is unique, annotated meta information becomes automatically comparable which is regarded as an essential break-through for automatising model analyses. In order to establish multi-artefact analyses the central management of annotations is important as

different tools can access the same database leading to the same annotation types in different artefacts. That way artefacts can be linked based on annotated meta information. However, up to now just annotation *types* are unique as they are stored centrally and offered to the user for selection. Annotation *values* are not yet kept centrally. This limits comparisons among annotations again and should be addressed in the next steps.

Generalisability The concept is held abstract enough to be applicable to other tool chains as well. Model analyses do not operate on the original artefacts but on their representatives in the model repository. Hence, the "only" tool-specific part of the concept is the interface which will have to be adapted to integrate other tools. However, changing the interface can be expected to be more reasonable than changing a tool as the latter would mean lots of costs for migration and teaching personal. That is why an interface change can supposed to pay off.

The extensibility for further artefacts was regarded as a further advantage of the presented concept. Integrating change requests, for instance, would also make the concept applicable to whole process management tools supporting estimating efforts.

Moreover, new annotation types can be added centrally and need not be made available to the different tools individually.

Barriers for concept introduction As almost every change of process the introduction of the presented concept cannot be established without preparatory work. All currently available free text information has to be added to the artefacts first meaning lots of hours of work. However, to the engineers' mind, the concept will pay off due to decreasing efforts in model analysis during evolution.

For real-world scenarios annotations should be made more expressive. For instance, annotations might be related to each other or even annotate other annotations. To this end, the data modelling and model transformations should be adapted accordingly. That way, the scope of model analyses could be widened even more.

4 Related Work

In [PMT+10] we elaborated structural and functional analyses to generate views on Simulink models to support the engineer with modelling tasks, e. g. to analyse parts of a Simulink model which depend on a given signal. We extended this work in [MPBK11] where we also considered analyses concerning inter-artefact relationships and made a first attempt to analyse meta information. However, as the meta information was specified using free text the concept was not sufficiently scalable onto productive models due to the amount of different syntax used to express the same semantics.

[TDH11] focuses on challenges concerning the introduction of a product line approach in the context of AUTOSAR-based [AUT] development of applications for in-vehicle systems. Especially, it is pointed out that current approaches to introduce a product line and related

tool chains do not sufficiently support the product line life cycle. To do so, a seamless variant management integrating different artefacts is necessary. In this context annotations as presented in this paper can be helpful to identify relationships and, hence, to support the migration.

[BFH+10] elaborates on the general problem of tool isolation in practice. They explain which challenges evolve from tool isolation in the context of traceability and consistency of artefacts. As a cause for inconsistency they identify that current interaction among tools is not deep enough. That way possibilities of reuse are too limited. They consider a comprehensive modelling theory, an integrated architectural model and an integrated model engineering environment to be fundamental prerequisites to establish a seamless model-based development process. Our approach of artefact integration addresses the isolation problem while the annotation concept intends to make the interaction deeper.

[BDT10] elaborates on model transformations concerning safety-related embedded systems. They also perform analysis of embedded systems with model transformation. However, their objective is to automate translations from architecture description in the automotive domain into a safety analysis tool called HiP-HOPS.

The framework ToolNet [ADS02] was developed by DaimlerChrysler using MOFLON [TD] focusing on the creation, the management and consistency checks of traceability links between different tools. ToolNet assists the developer in the manual construction of traceability links. In contrast to that, our approach will support the developer by automatically creating the traceability links through the annotation information.

[ZMV+03] associates objects of a model with the corresponding objects of the model in the following design step. They use annotations only for free text documentation. This documentation could be accessed by keyword search but is not intended for automated analysis.

[SVSZ01] elaborates on the problem of losing knowledge during the development process. They developed a tool called Clockwork Enrich Tool to annotate individual blocks of a Simulink model with free text. In [MZV+03] they also introduce formalisations. However, they just support few formalised relations among artefacts. Our approach provides artefacts with more formalised knowledge to automatically analyse these relations.

5 Summary

The presented work intends to support the model-based development of an embedded software product line. Two of the main challenges in this context are to overcome the barrier of tool isolation and to allow for automated analyses of artefacts and their interrelationships considering both functional aspects and process-related meta information. To our mind, the concept which is best suitable to tackle these challenges is to integrate the different artefacts into one tool-independent model repository which offers interfaces to the current tools in use. To capture meta information about the development process on the different artefacts we presented an annotation concept. By defining a syntax for annotations and a central artefact-independent repository for annotations the concept allows for automatising artefact

analyses with respect to development-related and evolution-related issues. Moreover, the concept is to a large extent constructed tool-independent as both artefacts to analyse and the related annotations are kept in an external repository.

To analyse the information specified by these annotations we can export them from Matlab/Simulink into our external framework and link them with representatives of the Simulink model. We have exemplified one analysis of meta information with the help of annotations in Section 2.3.

Up to now, we are able to import Simulink models and requirements into the external framework. Furthermore, we implemented the annotation concept for Simulink models. However, in order to establish traceability using the annotation concept we have to integrate annotations into the other artefacts as well. Last but not least the annotation concept is currently just a prototype. In order to put the concept into practice it has to be extended and restructured to make annotations more expressive. For instance, annotations are currently appended to blocks in a linear, i. e. flattened, manner. That way information about relationships among themselves cannot be captured and hence, we loose information compared to the free text approach where information can be grouped textually. This requires a deeper analysis of the kinds of information to be annotated and a different way to specify them in a user-friendly manner. After restructuring annotations a graphical user interface will be needed to support the engineers during evolution of the product line.

Acknowledgement

This work was funded, in part, by the Excellence Initiative of the German federal and state governments as well as Daimler AG. Moreover, we would like to thank Christian Dziobek, Thorsten Stecker, Uwe Spieth and the anonymous reviewers for useful inputs and constructive feedback about the presented concepts.

References

[ADS02] F. Altheide, H. Dörr, and A. Schürr. Requirements to a Framework for Sustainable Integration of System Development Tools. In *EuSEC '02*, pages 53–57, 2002.

[AUT] AUTOSAR. AUTOSAR AUTomotive Open System ARchitecture. `http://www.autosar.org/`.

[BDT10] M. Biehl, C. DeJiu, and M. Törngren. Integrating safety analysis into the model-based development toolchain of automotive embedded systems. In *LCTES '10*, pages 125–132, 2010.

[BFH+10] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu. Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments. *Proceedings of the IEEE*, 98(4):526 –545, April 2010.

[dSp] dSpace. Target Link. `http://www.dspace.com/en/ltd/home/products/sw/pcgs/targetli.cfm`.

[EFa] Eclipse-Foundation. EMF - Eclipse Modeling Framework. http://eclipse.org/modeling/emf/.

[EFb] Eclipse-Foundation. GMP - Graphical Modelling Project. http://www.eclipse.org/modeling/gmp/.

[EFc] Eclipse-Foundation. Xpand. http://www.eclipse.org/modeling/m2t/?project=xpand.

[EFd] Eclipse-Foundation. Xtext. http://www.eclipse.org/Xtext/.

[IC] IBM-Corporation. IBM Rational DOORS. http://www-01.ibm.com/software/awdtools/doors/.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature Oriented Domain Analysis (FODA) Feasibility Study. SEI Technical Report CMU/SEI-90-TR-21, ADA 235785, Software Engineering Institute, 1990.

[Mata] The MathWorks, Inc. Function Reference (MATLAB). http://www.mathworks.de/help/techdoc/ref/f16-6011.html.

[Matb] The MathWorks, Inc. Simulink – Simulation and Model-Based Design. http://www.mathworks.de/products/simulink.

[MD08] T. Mens and S. Demeyer. *Software evolution*. Springer, 2008.

[MPBK11] D. Merschen, A. Polzer, G. Botterweck, and S. Kowalewski. Experiences of Applying Model-based Analysis to Support the Development of Automotive Software Product Lines. In *VaMoS '11*, pages 141–150, 2011.

[MZV+03] P. Mulholland, Z. Zdrahal, M. Valasek, P. Sainter, M. Koss, and L. Trejtnar. Supporting the sharing and reuse of modelling and simulation design knowledge. In *ICE 2003*, 2003.

[PMT+10] A. Polzer, D. Merschen, J. Thomas, B. Hedenetz, G. Botterweck, and S. Kowalewski. View-Supported Rollout and Evolution of Model-Based ECU Applications. In *MoMPES '10*, pages 37–44, 2010.

[Som07] I. Sommerville. *Software engineering*. International computer science series. Addison-Wesley, 2007.

[SVSZ01] P. Steinbauer, M. Valasek, Z. Sika, and Z. Zdrahal. Knowledge supported design and reuse of simulation models. In *MATLAB 2001*, pages 399–406, 2001.

[TD] TU-Darmstadt. MOFLON. http://www.moflon.org/.

[TDH11] J. Thomas, C. Dziobek, and B. Hedenetz. Variability management in the AUTOSAR-based development of applications for in-vehicle systems. In *VaMoS '11*, pages 137–140, 2011.

[WDR08] F. Wohlgemuth, C. Dziobek, and T. Ringler. Erfahrungen bei der Einführung der modellbasierten AUTOSAR-Funktionsentwicklung. In *MBEFF '08*, pages 1 – 15, 2008.

[ZMV+03] Z. Zdrahal, P. Mulholland, M. Valasek, P. Sainter, M. Koss, and L. Trejtnar. A Toolkit and Methodology to Support the Collaborative Development and Reuse of Engineering Models. In *DEXA 2003*, pages 856–865, 2003.

Einführung eines Produktlinienansatzes in die automotive Softwareentwicklung am Beispiel von Steuergerätesoftware

Christian Hopp, Fabian Wolf
Elektronik-Entwicklung
Volkswagen AG Braunschweig
http://www.volkswagen.de

Holger Rendel, Bernhard Rumpe
Lehrstuhl Software Engineering
RWTH Aachen
http://www.se-rwth.de

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

1 Einleitung

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

Eine Optimierung auf allen Schichten des Softwareentwicklungsprozesses kann vor allem durch Nutzung eines Software-Produktlinienansatzes erfolgen. Dieser erlaubt durch die gezielte Dokumentation und Nutzung von Variabilität eine schnelle und effiziente Entwicklung von Produktfamilien. Dadurch kann auch die Komplexität in zukünftigen

Softwaregenerationen verringert werden. Die mit einer Software-Produktlinie verbundenen Auswirkungen auf die Software-Architektur erlauben eine agilere Entwicklung.

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

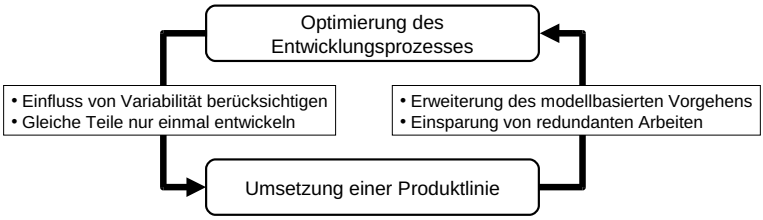


Abbildung 1: Zusammenhang zwischen Optimierung des Entwicklungsprozesses und Umsetzung von Produktlinien

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

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

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

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

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

2 Unterstützungsmöglichkeiten für Produktlinien

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

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

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

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

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

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

2.1 Identifikation oder Erstellung von Modellen

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

der Entwicklung die folgenden wesentlichen Eigenschaften [Sta73]:

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

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

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

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

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

2.2 Nutzung der Modelle im Produktlinienkontext

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

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

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

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

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

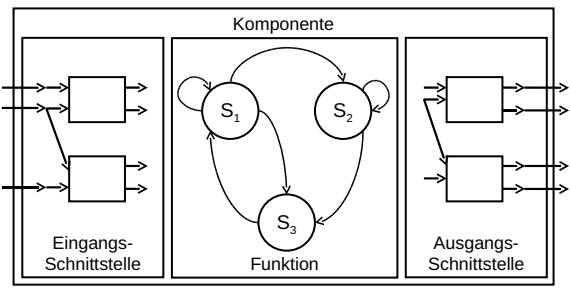


Abbildung 2: Aufteilung in Funktion und Schnittstelle

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

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

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

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

3 Industrieller Einsatz

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

3.1 Nutzung von Modellen und Erhöhung der Modellqualität

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

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

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

3.2 Modulaufteilung

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

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

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

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

3.3 Generierung von Test-Infrastruktur

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

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

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

4 Wirksamkeit der Maßnahmen

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

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

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

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

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

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

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

Tabelle 1: Vergleich vor und nach Modulaufteilung

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

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

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

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

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

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

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

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

5 Verwandte Arbeiten

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

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

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

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

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

6 Zusammenfassung

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

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

Literatur

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

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

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

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

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

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

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

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

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

[pur] pure systems. pure::variants.

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

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

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

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

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

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

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

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

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

Workshops

5. Arbeitstagung Programmiersprachen (ATPS 2012)

Frank Huch[1], Janis Voigtländer[2]

[1] Christian-Albrechts-Universität zu Kiel
Christian-Albrechts-Platz 4, 24118 Kiel
fhu@informatik.uni-kiel.de
[2] Universität Bonn
Römerstraße 164, 53117 Bonn
jv@informatik.uni-bonn.de

Die Tagung dient dem Austausch zwischen Forschern, Entwicklern und Anwendern, die sich mit Themen aus dem Bereich der Programmiersprachen beschaeftigen. Alle Programmierparadigmen sind von Interesse: imperative, objektorientierte, funktionale, logische, parallele, graphische Programmiersprachen, auch verteilte und nebenlaeufige Programmierung in Intra- und Internet-Anwendungen, sowie Konzepte zur Integration dieser Paradigmen. Die ersten vier Arbeitstagungen Programmiersprachen fanden im Rahmen von GI-Jahrestagungen statt (Aachen 1997, Paderborn 1999, Ulm 2004, Lübeck 2009) statt. Wegen der größeren inhaltlichen Nähe, findet sie in diesem Jahr zusammen mit der GI-Tagung Software Engineering statt. Typische, aber nicht ausschliessliche Themenbereiche sind:

- Entwurf von Programmiersprachen und anwendungsspezifischen Sprachen
- Analyse und Transformation von Programmen
- Typsysteme, Semantik, Spezifikationstechniken
- Modellierungssprachen, Objektorientierung
- Intra- und Internet-Programmierung
- Programm- und Implementierungsverifikation
- Werkzeuge, IDEs, Frameworks, Architekturen, generative Ansätze
- Implementierungs- und Optimierungstechniken
- Verbindung von Sprachen, Architekturen, Prozessoren

Ebenfalls von Interesse sind Arbeiten zu Techniken, Methoden, Konzepten oder Werkzeugen, mit denen Sicherheit und Zuverlaessigkeit bei der Ausfuehrung von Programmen erhoeht werden kann. Die Tagung richtet sich ausdruecklich auch an Interessenten aus Wirtschaft und Industrie. Neben neuen Arbeiten können auch existierende Arbeiten oder Projekte zusammengefasst bzw. aus einem anderen Blickwinkel präsentiert werden und so insbesondere einem deutschsprachigen Publikum präsentiert werden.

IT-Unterstützung für Public Safety & Security: Interdisziplinäre Anforderungsanalyse, Architekturen und Gestaltungskonzepte (IT4PSS 2012)

Birgitta König-Ries[1], Volkmar Pipek[2], Jens Pottebaum[3], Stefan Strohschneider[1]

[1] Friedrich-Schiller-Universität Jena
Ernst-Abbe-Platz 2, 07743 Jena
birgitta.koenig-ries, stefan.strohschneider@uni-jena.de
[2] Universität Siegen
Hölderlinstr. 3, 57076 Siegen
volkmar.pipek@uni-siegen.de
[3] Universität Paderborn,
Pohlweg 47-49, 33098 Paderborn
pottebaum@cik.upb.de

IT-Lösungen in nicht-traditionellen Anwendungsgebieten, wie etwa in der Sicherheitsforschung, erfordern die Bearbeitung äußerst heterogener Fragegestellungen und Einbeziehung aller relevanten Stakeholder. Diese Aspekte zählen auch zu den Leitlinien des vom Bundesministerium für Bildung und Forschung (BMBF) im Rahmen der Hightech-Strategie durchgeführten Programms Forschung für die zivile Sicherheit. Gerade im Themenschwerpunkt Schutz und Rettung von Menschen werden hier gemeinsam von Wissenschaft, Wirtschaft und Endanwendern interdisziplinär IT-Unterstützungen für die Menschenrettung aus Gefahrenlagen sowie den Brand- und Katastrophenschutz entwickelt. Die Interdisziplinarität von Forschungsaktivitäten bedeutet natürlich Chance und Herausforderung zugleich. Dies betrifft insbesondere die Erforschung der Anwendung von IT-Systemen, die den Kern vieler nationaler und internationaler Aktivitäten bilden.

In diesem Kontext unterliegen Entwicklung und Nutzung von IT-Systemen zur Unterstützung aller beteiligten Akteure in Szenarien der zivilen Sicherheit („alltägliche“ zivile Gefahrenabwehr, Bevölkerungsschutz, Katastrophenhilfe, etc.) besonderen Rahmenbedingungen durch ihre Öffentlichkeitswirksamkeit und Notwendigkeiten der Verlässlichkeit, Anpassbarkeit/Flexibilität, Sicherheitsrelevanz, Nachvollziehbarkeit und Interoperabilität solcher IT-Systeme. Neben dem besonderen Fokus auf die Anforderungsanalyse ist es daher Ziel des Workshops, Designer und Systementwickler aus diesem Bereich zusammenzubringen, um Erfahrungen und Ansätze für Architekturen, Anwendungen und Vorgehensweisen zu diskutieren. Die Innovativität der diskutierten Ansätze kann dabei sowohl auf der besonders gelungenen Umsetzung der Rahmenbedingungen als auch auf der Erschließung und Einbindung neuer technologischer Konzepte basieren.

Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation (PIK 2012)

Andreas Birk[1], Florian Markert[2], Sebastian Oster[2]

[1] Software.Process.Management
Usedomstraße 15, 70439 Stuttgart
andreas.birk@swpm.de
[2] TU Darmstadt
Merckstraße 25, 64283 Darmstadt
markert, oster@rs.tu-darmstadt.de

Der Workshop bietet ein Forum für die deutschsprachige Community zu Software-Produktlinien (SPL) und fördert den Erfahrungsaustausch zu SPL. Er verfolgt insbesondere die Ziele:

- Den Dialog zwischen Praxis und anwendungsorientierter Forschung fördern
- SPL-Erfahrungen und neue SPL-Technologien vorstellen
- Eine Standortbestimmung der SPL-Technologie in Forschung / Praxis vornehmen

Produktlinien sind heute in vielen Bereichen der Software-Industrie vertreten von eingebetteten Systemen bis zu betrieblichen Informationssystemen. Sie ermöglichen höhere Produktivität, steigern die Qualität und verbessern die strategischen Positionen der Unternehmen. Dennoch bergen Produktlinien für viele Unternehmen noch bedeutende Herausforderungen und Risiken. Die Gründe liegen teilweise im technischen Bereich. So sind viele Produktlinien-Technologien für den breiten Einsatz in der Praxis noch nicht genügend ausgereift und integriert. Die wohl größten Herausforderungen stellen sich in den Wechselwirkungen zwischen den technischen Verfahren mit den Prozessen sowie dem organisatorischen und geschäftlichen Kontext der Produktlinienentwicklung. Wie müssen die technologischen Ansätze auf diese Wechselwirkungen ausgerichtet sein? Welche Potenziale bieten neue technologische Entwicklungen in unterschiedlichen Einsatzfeldern?

PIK 2012 beleuchtet aktuelle Erfahrungen mit Produktlinien und fördert den Dialog zwischen Praxis und anwendungsorientierter Forschung. Im Mittelpunkt steht das Wechselspiel zwischen technischen Fragestellungen und den geschäftlichen, organisatorischen und prozessbezogenen Aspekten. Daneben werden auch neue technologische Entwicklungen vorgestellt und diskutiert.

Zertifizierung und modellgetriebene Entwicklung sicherer Software (ZeMoSS 2012)

Michaela Huhn [1], Stefan Gerken [2], Carsten Rudolph [3]

[1] TU Clausthal
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld
Michaela.Huhn@tu-clausthal.de
[2] Siemens AG
Ackerstr. 22, 38126 Braunschweig
Stefan.Gerken@siemens.com
[2] Fraunhofer-Institut für Sichere Informationstechnologie
Rheinstraße 75, 64295 Darmstadt
carsten.rudolph@sit.fraunhofer.de

Mit dem vielfältigen Einsatz softwaregesteuerter Produkte und Infrastrukturen in unserem Alltag wachsen die Software-Qualitätsanforderungen, insbesondere in den Bereichen funktionale Sicherheit und Informationssicherheit. In der Luft- und Raumfahrt, der Energieerzeugung und im Schienenverkehr, aber auch in der Medizintechnik, der Automobiltechnik und bei mobilen Systemen sind Zertifizierung und der Nachweis der Sicherheit kritischer Systeme und softwarespezifische Sicherheitsnormen international etabliert und bindend. Zwei aktuelle, domänenübergreifende Herausforderungen bei der Entwicklung sicherer Software sollen im Workshop adressiert werden:

(1) Modellgetriebene Entwicklung eingebetteter Software wird in der Industrie immer wichtiger und in ihren Grundlagen für höhere Sicherheitsanforderungsstufen seit langem in Sicherheitsnormen als dringend empfohlen klassifiziert. Da Normen aber immer nur die etablierten Regeln der Technik darstellen, entsteht für den Hersteller mit jedem Schritt hin zum erweiterten Einsatz modellgetriebener Methoden und Werkzeuge die Herausforderung, dass diese im Zertifizierungsprozess neu akzeptiert werden müssen, selbst wenn noch keine normativen Aussagen zu ihnen vorliegen.

(2) Durch die zunehmende Vernetzung kritischer Infrastrukturen und die Anbindung mobiler Endgeräte entstehen neue Risiken aus der wechselseitigen Abhängigkeit von Informationssicherheit und funktionaler Sicherheit. Hier sind eine Integration von Safety- und Security-Prozess und neue Methoden gefragt, die eine verbindende Behandlung von funktionaler Sicherheit und Informationssicherheit in der Risikoanalyse, der Entwicklung und beim Sicherheitsnachweis unterstützen.

Der Workshop soll den Austausch über gewonnene Erfahrungen und neue Ansätze zur Entwicklung zertifizierbarer, sicherheitskritischer Software und insbesondere zu den Herausforderungen Modellgetriebene Entwicklung und Integration von Informations- und funktionaler Sicherheit zwischen Teilnehmern aus Industrie und Forschung fördern.

analyses with respect to development-related and evolution-related issues. Moreover, the concept is to a large extent constructed tool-independent as both artefacts to analyse and the related annotations are kept in an external repository.

To analyse the information specified by these annotations we can export them from Matlab/Simulink into our external framework and link them with representatives of the Simulink model. We have exemplified one analysis of meta information with the help of annotations in Section 2.3.

Up to now, we are able to import Simulink models and requirements into the external framework. Furthermore, we implemented the annotation concept for Simulink models. However, in order to establish traceability using the annotation concept we have to integrate annotations into the other artefacts as well. Last but not least the annotation concept is currently just a prototype. In order to put the concept into practice it has to be extended and restructured to make annotations more expressive. For instance, annotations are currently appended to blocks in a linear, i. e. flattened, manner. That way information about relationships among themselves cannot be captured and hence, we loose information compared to the free text approach where information can be grouped textually. This requires a deeper analysis of the kinds of information to be annotated and a different way to specify them in a user-friendly manner. After restructuring annotations a graphical user interface will be needed to support the engineers during evolution of the product line.

Acknowledgement

This work was funded, in part, by the Excellence Initiative of the German federal and state governments as well as Daimler AG. Moreover, we would like to thank Christian Dziobek, Thorsten Stecker, Uwe Spieth and the anonymous reviewers for useful inputs and constructive feedback about the presented concepts.

References

[ADS02] F. Altheide, H. Dörr, and A. Schürr. Requirements to a Framework for Sustainable Integration of System Development Tools. In *EuSEC '02*, pages 53–57, 2002.

[AUT] AUTOSAR. AUTOSAR AUTomotive Open System ARchitecture. `http://www.autosar.org/`.

[BDT10] M. Biehl, C. DeJiu, and M. Törngren. Integrating safety analysis into the model-based development toolchain of automotive embedded systems. In *LCTES '10*, pages 125–132, 2010.

[BFH+10] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu. Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments. *Proceedings of the IEEE*, 98(4):526 –545, April 2010.

[dSp] dSpace. Target Link. `http://www.dspace.com/en/ltd/home/products/sw/pcgs/targetli.cfm`.

[EFa] Eclipse-Foundation. EMF - Eclipse Modeling Framework. http://eclipse.org/modeling/emf/.

[EFb] Eclipse-Foundation. GMP - Graphical Modelling Project. http://www.eclipse.org/modeling/gmp/.

[EFc] Eclipse-Foundation. Xpand. http://www.eclipse.org/modeling/m2t/?project=xpand.

[EFd] Eclipse-Foundation. Xtext. http://www.eclipse.org/Xtext/.

[IC] IBM-Corporation. IBM Rational DOORS. http://www-01.ibm.com/software/awdtools/doors/.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature Oriented Domain Analysis (FODA) Feasibility Study. SEI Technical Report CMU/SEI-90-TR-21, ADA 235785, Software Engineering Institute, 1990.

[Mata] The MathWorks, Inc. Function Reference (MATLAB). http://www.mathworks.de/help/techdoc/ref/f16-6011.html.

[Matb] The MathWorks, Inc. Simulink – Simulation and Model-Based Design. http://www.mathworks.de/products/simulink.

[MD08] T. Mens and S. Demeyer. *Software evolution*. Springer, 2008.

[MPBK11] D. Merschen, A. Polzer, G. Botterweck, and S. Kowalewski. Experiences of Applying Model-based Analysis to Support the Development of Automotive Software Product Lines. In *VaMoS '11*, pages 141–150, 2011.

[MZV+03] P. Mulholland, Z. Zdrahal, M. Valasek, P. Sainter, M. Koss, and L. Trejtnar. Supporting the sharing and reuse of modelling and simulation design knowledge. In *ICE 2003*, 2003.

[PMT+10] A. Polzer, D. Merschen, J. Thomas, B. Hedenetz, G. Botterweck, and S. Kowalewski. View-Supported Rollout and Evolution of Model-Based ECU Applications. In *MoMPES '10*, pages 37–44, 2010.

[Som07] I. Sommerville. *Software engineering*. International computer science series. Addison-Wesley, 2007.

[SVSZ01] P. Steinbauer, M. Valasek, Z. Sika, and Z. Zdrahal. Knowledge supported design and reuse of simulation models. In *MATLAB 2001*, pages 399–406, 2001.

[TD] TU-Darmstadt. MOFLON. http://www.moflon.org/.

[TDH11] J. Thomas, C. Dziobek, and B. Hedenetz. Variability management in the AUTOSAR-based development of applications for in-vehicle systems. In *VaMoS '11*, pages 137–140, 2011.

[WDR08] F. Wohlgemuth, C. Dziobek, and T. Ringler. Erfahrungen bei der Einführung der modellbasierten AUTOSAR-Funktionsentwicklung. In *MBEFF '08*, pages 1 – 15, 2008.

[ZMV+03] Z. Zdrahal, P. Mulholland, M. Valasek, P. Sainter, M. Koss, and L. Trejtnar. A Toolkit and Methodology to Support the Collaborative Development and Reuse of Engineering Models. In *DEXA 2003*, pages 856–865, 2003.

GI-Edition Lecture Notes in Informatics

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Lan-guage to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Business 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Er-fahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometrics and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment

P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society

P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications

P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven

P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik

P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004

P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration

P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Adhoc-Netze Medienintegration

P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government

P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen

P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality

P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Ranneberg, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments

P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für inforrmatische Bildung

P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen

P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit

P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and ist Applications

P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005

P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolffried Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web

P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik

P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)

P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)

P-69 Robert Hirschfeld, Ryszard Kowalcyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005

P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)

P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005

P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): "Heute schon das Morgen sehen"

P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology

P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture

P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz

P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006

P-78 K.-O. Wenkel, P. Wagner, M. Morgenstern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce

P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS´06

P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006

P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics

P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications

P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems

P-86 Robert Krimmer (Ed.): Electronic Voting 2006

P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik

P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006

P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur

P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006

P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006

P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1

P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2

P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen

P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies

P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices

P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik

P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik

P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten

P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments

P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)

P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme

P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007

P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops

P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications

P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures

P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1

P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2

P-111 Christian Eibl, Johannes Magenheim, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik

P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis

P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)

P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics

P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walther (Eds.) German conference on bioinformatics GCB 2007

P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007

P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and manegement The 4th International Conference on Grid Service Engineering and Management GSEM 2007

P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007

P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications

P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges

P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik

P-122 Walid Maalej, Bernd Bruegge (Hrsg.) Software Engineering 2008 - Workshopband Fachtagung des GI-Fachbereichs Softwaretechnik

P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)

P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Speciel Interest Groups PARS and PARVA

P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT: Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung

P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)

P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008

P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany

P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems

P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung

P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting. CC

P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik

P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 1

P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 2

P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik – Aktuelle Forschungsergebnisse

P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008

P-137 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures

P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008

P-139 Georg Herzwurm, Martin Mikusz (Hrsg.)
Industrialisierung des Software-Managements
Fachtagung des GI-Fachausschusses Management der Anwendungsentwick-lung und -wartung im Fachbereich Wirtschaftsinformatik

P-140 Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)
IMF 2008 - IT Incident Management & IT Forensics

P-141 Peter Loos, Markus Nüttgens, Klaus Turowski, Dirk Werth (Hrsg.)
Modellierung betrieblicher Informations-systeme (MobIS 2008)
Modellierung zwischen SOA und Compliance Management

P-142 R. Bill, P. Korduan, L. Theuvsen, M. Morgenstern (Hrsg.)
Anforderungen an die Agrarinformatik durch Globalisierung und Klimaveränderung

P-143 Peter Liggesmeyer, Gregor Engels, Jürgen Münch, Jörg Dörr, Norman Riegel (Hrsg.)
Software Engineering 2009
Fachtagung des GI-Fachbereichs Softwaretechnik

P-144 Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)
Datenbanksysteme in Business, Technologie und Web (BTW)

P-145 Knut Hinkelmann, Holger Wache (Eds.)
WM2009: 5th Conference on Professional Knowledge Management

P-146 Markus Bick, Martin Breunig, Hagen Höpfner (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Entwicklung, Implementierung und Anwendung
4. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2009)

P-147 Witold Abramowicz, Leszek Maciaszek, Ryszard Kowalczyk, Andreas Speck (Eds.)
Business Process, Services Computing and Intelligent Service Management
BPSC 2009 · ISM 2009 · YRW-MBP 2009

P-148 Christian Erfurth, Gerald Eichler, Volkmar Schau (Eds.)
9th International Conference on Innovative Internet Community Systems
I^{2}CS 2009

P-149 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
2. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-150 Jürgen Münch, Peter Liggesmeyer (Hrsg.)
Software Engineering
2009 - Workshopband

P-151 Armin Heinzl, Peter Dadam, Stefan Kirn, Peter Lockemann (Eds.)
PRIMIUM
Process Innovation for Enterprise Software

P-152 Jan Mendling, Stefanie Rinderle-Ma, Werner Esswein (Eds.)
Enterprise Modelling and Information Systems Architectures
Proceedings of the 3rd Int'l Workshop EMISA 2009

P-153 Andreas Schwill, Nicolas Apostolopoulos (Hrsg.)
Lernen im Digitalen Zeitalter
DeLFI 2009 – Die 7. E-Learning Fachtagung Informatik

P-154 Stefan Fischer, Erik Maehle Rüdiger Reischuk (Hrsg.)
INFORMATIK 2009
Im Focus das Leben

P-155 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2009:
Biometrics and Electronic Signatures
Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-156 Bernhard Koerber (Hrsg.)
Zukunft braucht Herkunft
25 Jahre »INFOS – Informatik und Schule«

P-157 Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, Peter Stadler (Eds.)
German Conference on Bioinformatics 2009

P-158 W. Claupein, L. Theuvsen, A. Kämpf, M. Morgenstern (Hrsg.)
Precision Agriculture Reloaded – Informationsgestützte Landwirtschaft

P-159 Gregor Engels, Markus Luckey, Wilhelm Schäfer (Hrsg.)
Software Engineering 2010

P-160 Gregor Engels, Markus Luckey, Alexander Pretschner, Ralf Reussner (Hrsg.)
Software Engineering 2010 – Workshopband
(inkl. Doktorandensymposium)

P-161 Gregor Engels, Dimitris Karagiannis Heinrich C. Mayr (Hrsg.)
Modellierung 2010

P-162 Maria A. Wimmer, Uwe Brinkhoff, Siegfried Kaiser, Dagmar Lück-Schneider, Erich Schweighofer, Andreas Wiebe (Hrsg.)
Vernetzte IT für einen effektiven Staat
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2010

P-163 Markus Bick, Stefan Eulgem, Elgar Fleisch, J. Felix Hampe, Birgitta König-Ries, Franz Lehner, Key Pousttchi, Kai Rannenberg (Hrsg.)
Mobile und Ubiquitäre Informationssysteme
Technologien, Anwendungen und Dienste zur Unterstützung von mobiler Kollaboration

P-164 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2010: Biometrics and Electronic Signatures Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-165 Gerald Eichler, Peter Kropf, Ulrike Lechner, Phayung Meesad, Herwig Unger (Eds.)
10th International Conference on Innovative Internet Community Systems (I^2CS) – Jubilee Edition 2010 –

P-166 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
3. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-167 Robert Krimmer, Rüdiger Grimm (Eds.)
4th International Conference on Electronic Voting 2010
co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-168 Ira Diethelm, Christina Dörge, Claudia Hildebrandt, Carsten Schulte (Hrsg.)
Didaktik der Informatik
Möglichkeiten empirischer Forschungsmethoden und Perspektiven der Fachdidaktik

P-169 Michael Kerres, Nadine Ojstersek Ulrik Schroeder, Ulrich Hoppe (Hrsg.)
DeLFI 2010 - 8. Tagung der Fachgruppe E-Learning der Gesellschaft für Informatik e.V.

P-170 Felix C. Freiling (Hrsg.)
Sicherheit 2010
Sicherheit, Schutz und Zuverlässigkeit

P-171 Werner Esswein, Klaus Turowski, Martin Juhrisch (Hrsg.)
Modellierung betrieblicher Informationssysteme (MobIS 2010)
Modellgestütztes Management

P-172 Stefan Klink, Agnes Koschmider Marco Mevius, Andreas Oberweis (Hrsg.)
EMISA 2010
Einflussfaktoren auf die Entwicklung flexibler, integrierter Informationssysteme
Beiträge des Workshops der GI-Fachgruppe EMISA (Entwicklungsmethoden für Informationssysteme und deren Anwendung)

P-173 Dietmar Schomburg, Andreas Grote (Eds.)
German Conference on Bioinformatics 2010

P-174 Arslan Brömme, Torsten Eymann, Detlef Hühnlein, Heiko Roßnagel, Paul Schmücker (Hrsg.)
perspeGKtive 2010
Workshop „Innovative und sichere Informationstechnologie für das Gesundheitswesen von morgen“

P-175 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 1

P-176 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 2

P-177 Witold Abramowicz, Rainer Alt, Klaus-Peter Fähnrich, Bogdan Franczyk, Leszek A. Maciaszek (Eds.)
INFORMATIK 2010
Business Process and Service Science – Proceedings of ISSS and BPSC

P-178 Wolfram Pietsch, Benedikt Krams (Hrsg.)
Vom Projekt zum Produkt
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschafts-informatik (WI-MAW), Aachen, 2010

P-179 Stefan Gruner, Bernhard Rumpe (Eds.)
FM+AM`2010
Second International Workshop on Formal Methods and Agile Methods

P-180 Theo Härder, Wolfgang Lehner, Bernhard Mitschang, Harald Schöning, Holger Schwarz (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW)
14. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS)

P-181 Michael Clasen, Otto Schätzel, Brigitte Theuvsen (Hrsg.)
Qualität und Effizienz durch informationsgestützte Landwirtschaft, Fokus: Moderne Weinwirtschaft

P-182 Ronald Maier (Hrsg.)
6th Conference on Professional Knowledge Management
From Knowledge to Action

P-183 Ralf Reussner, Matthias Grund, Andreas Oberweis, Walter Tichy (Hrsg.)
Software Engineering 2011
Fachtagung des GI-Fachbereichs Softwaretechnik

P-184 Ralf Reussner, Alexander Pretschner, Stefan Jähnichen (Hrsg.)
Software Engineering 2011
Workshopband
(inkl. Doktorandensymposium)

P-185 Hagen Höpfner, Günther Specht, Thomas Ritz, Christian Bunse (Hrsg.) MMS 2011: Mobile und ubiquitäre Informationssysteme Proceedings zur 6. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2011)

P-186 Gerald Eichler, Axel Küpper, Volkmar Schau, Hacène Fouchal, Herwig Unger (Eds.) 11th International Conference on Innovative Internet Community Systems (I^2CS)

P-187 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.) 4. DFN-Forum Kommunikations-technologien, Beiträge der Fachtagung 20. Juni bis 21. Juni 2011 Bonn

P-188 Holger Rohland, Andrea Kienle, Steffen Friedrich (Hrsg.) DeLFI 2011 – Die 9. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. 5.–8. September 2011, Dresden

P-189 Thomas, Marco (Hrsg.) Informatik in Bildung und Beruf INFOS 2011 14. GI-Fachtagung Informatik und Schule

P-190 Markus Nüttgens, Oliver Thomas, Barbara Weber (Eds.) Enterprise Modelling and Information Systems Architectures (EMISA 2011)

P-191 Arslan Brömme, Christoph Busch (Eds.) BIOSIG 2011 International Conference of the Biometrics Special Interest Group

P-192 Hans-Ulrich Heiß, Peter Pepper, Holger Schlingloff, Jörg Schneider (Hrsg.) INFORMATIK 2011 Informatik schafft Communities

P-193 Wolfgang Lehner, Gunther Piller (Hrsg.) IMDM 2011

P-194 M. Clasen, G. Fröhlich, H. Bernhardt, K. Hildebrand, B. Theuvsen (Hrsg.) Informationstechnologie für eine nachhaltige Landbewirtschaftung Fokus Forstwirtschaft

P-198 Stefan Jähnichen, Axel Küpper, Sahin Albayrak (Hrsg.) Software Engineering 2012 Fachtagung des GI-Fachbereichs Softwaretechnik

The titles can be purchased at:

Köllen Druck + Verlag GmbH
Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn
Fax: +49 (0)228/9898222
E-Mail: druckverlag@koellen.de