

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 co-operation 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 978-388579-621-3

This volume contains the contributions of the Software Engineering 2014 conference held from February 25<sup>th</sup> to February 28<sup>th</sup> 2014 in Kiel, Germany. The conference series SE is the German-speaking conference of the special interest group software engineering of the Gesellschaft für Informatik e.V. (GI) focussing on software engineering.

These proceedings contain entries from the scientific program, the technology transfer program, the software & systems engineering essentials, the doctoral symposium, as well as entries from workshops, tutorials, and software engineering ideas.



# GI-Edition

## Lecture Notes in Informatics

**Wilhelm Hasselbring,  
Nils Christian Ehmke (Hrsg.)**

## Software Engineering 2014

**Fachtagung des GI-Fachbereichs  
Softwaretechnik**

**25. Februar – 28. Februar 2014  
Kiel, Deutschland**

Software Engineering 2014

227

# Proceedings









Wilhelm Hasselbring, Nils Christian Ehmke (Hrsg.)

## **Software Engineering 2014**

**Fachtagung des GI-Fachbereichs Softwaretechnik**

**25. Februar – 28. Februar 2014  
in Kiel, Deutschland**

Gesellschaft für Informatik e.V. (GI)

## **Lecture Notes in Informatics (LNI) - Proceedings**

Series of the Gesellschaft für Informatik (GI)

Volume P-227

ISBN 978-388579-621-3

ISSN 1617-5468

### **Volume Editors**

Prof. Dr. Wilhelm Hasselbring

Arbeitsgruppe Software Engineering

Institut für Informatik, Christian-Albrechts-Universität zu Kiel

24118 Kiel, Deutschland

Email: hasselbring@email.uni-kiel.de

M.Sc. Nils Christian Ehmke

Arbeitsgruppe Software Engineering

Institut für Informatik, Christian-Albrechts-Universität zu Kiel

24118 Kiel, Deutschland

Email: nils.ehmke@email.uni-kiel.de

### **Series Editorial Board**

Heinrich C. Mayr, Alpen-Adria-Universität Klagenfurt, Austria

(Chairman, mayr@ifit.uni-klu.ac.at)

Dieter Fellner, Technische Universität Darmstadt, Germany

Ulrich Flegel, Hochschule für Technik, Stuttgart, Germany

Ulrich Frank, Universität Duisburg-Essen, Germany

Johann-Christoph Freytag, Humboldt-Universität zu Berlin, Germany

Michael Goedicke, Universität Duisburg-Essen, Germany

Ralf Hofestädt, Universität Bielefeld, Germany

Michael Koch, Universität der Bundeswehr München, Germany

Axel Lehmann, Universität der Bundeswehr München, Germany

Peter Sanders, Karlsruher Institut für Technologie (KIT), Germany

Sigrid Schubert, Universität Siegen, Germany

Ingo Timm, Universität Trier, Germany

Karin Vosseberg, Hochschule Bremerhaven, Germany

Maria Wimmer, Universität Koblenz-Landau, Germany

### **Dissertations**

Steffen Hölldobler, Technische Universität Dresden, Germany

### **Seminars**

Reinhard Wilhelm, Universität des Saarlandes, Germany

### **Thematics**

Andreas Oberweis, Karlsruher Institut für Technologie (KIT), Germany

© Gesellschaft für Informatik, Bonn 2014

**printed by** Köllen Druck+Verlag GmbH, Bonn

# Vorwort

Willkommen zur Tagung Software Engineering 2014 an der Kieler Förde!

Software Engineering ist eine praxisorientierte Wissenschaftsdisziplin. Die Ergebnisse der Software Engineering Forschung sollten in die Praxis der Softwareentwicklung einfließen, gleichzeitig können die relevanten Fragen der Praxis den Anstoß für innovative Forschungsprojekte geben. Wissens- und Technologietransfer ist ein bidirektionaler Prozess. Um diesen Transfer zu befördern, bietet die Software Engineering 2014 ein Forum für die deutschsprachige Software Engineering Community. In parallelen Vortragssitzungen werden Highlights aus der Wissenschaft, aus dem praktizierten Technologietransfer und aus der industriellen Praxis berichtet. Diese Vortragssitzungen werden eingerahmt von hochkarätigen Keynote-Vorträgen. Das diesjährige Tagungsmotto lautet konsequenterweise

„Transfer zwischen Wissenschaft und Wirtschaft“

Um diesen Transfer zu befördern, finden im Hauptprogramm der Tagung parallele Sitzungen zum wissenschaftlichen Programm, zum Technologietransfer, zu Software & Systems Engineering Essentials und zum Industrieprogramm in parallelen und teils gemischten Sitzungen statt. Der Austausch soll dann insbesondere durch die gemeinsamen Pausen im Foyer gefördert werden.

Die Konferenzserie SE ist die deutschsprachige Konferenz zum Thema Software Engineering des Fachbereichs Softwaretechnik der Gesellschaft für Informatik e. V. (GI). Die Software Engineering 2014 wird gemeinsam vom Lehrstuhl für Software Engineering im Institut für Informatik der Technischen Fakultät der Christian-Albrechts-Universität zu Kiel, dem Verein der Digitalen Wirtschaft Schleswig-Holstein e.V. (DiWiSH), der Gesellschaft für Informatik e. V. (GI), der IHK Kiel und dem Kompetenzverbund Software Systems Engineering (KoSSE) organisiert.

Das Programm umfasst in diesem Jahr die folgenden Elemente:

- Drei Keynotes aus Industrie, Technologietransfer und Wissenschaft.
- Eine Podiumsdiskussion zu Ausgründungen in der Softwaretechnik.
- Das wissenschaftliche Programm, mit neuem Format.  
(Leitung: Andreas Zeller, Universität des Saarlandes, Saarbrücken)
- Die Software Engineering Ideen (Leitung: Bernd Brügge, TU München)
- Die Software & Systems Engineering Essentials  
(Leitung: Marc Sihling, 4Soft GmbH)

- Das Technologietransferprogramm (Leitung: Ralf Reussner, KIT / FZI)
- Das Industrieprogramm (Leitung: Wilhelm Hasselbring, Universität Kiel)
- Das Doktorandensymposium (Leitung: Rainer Koschke, Universität Bremen)
- Die Tutorials (Leitung: Klaus Schmid, Universität Hildesheim)
- Die Workshops (Leitung: Klaus Schmid, Universität Hildesheim)
- Das Studierendenprogramm (Leitung: Dirk Nowotka, Universität Kiel)
- Der Software Engineering Preis (Ernst-Denert-Stiftung)

Insbesondere das neue Format für das wissenschaftliche Programm hat sich als Erfolgskonzept herausgestellt, sehen Sie sich dazu auch die Einleitung von Andreas Zeller an. Im Gesamtbild mit den weiteren Programmelementen bietet die Software Engineering 2014 Ihnen ein zugleich hochkarätiges und vielfältiges Programm. Ich möchte den oben aufgelisteten Koordinatoren der jeweiligen Programmelemente dafür danken, dass wir gemeinsam dieses Programm aus den vielen Einreichungen zusammenstellen konnten (insgesamt konnte nur die Hälfte der Vorschläge berücksichtigt werden).

Mein besonderer Dank gilt auch allen Studierenden, Mitarbeitern und Unterstützern, die in unterschiedlicher Weise zum Gelingen der Tagung beitragen.

Kiel, im Dezember 2013

Wilhelm Hasselbring



---

GOLD SPONSOREN

---

**Accso**  
Accelerated Solutions

**adesso** | business.  
people.  
technology.

**b+m**  
Informatik AG  
AN ALLGEIER COMPANY

**C1 WPS**

**imbus**

---

SILBER SPONSOREN

---

**msg**  
systems

---

BRONZE SPONSOREN

---

**ABB**

**Cap3**  
Softwarekonzeption

**ESN**  
Power für Ihr Business

**itemis**

**IVU** TRAFFIC  
TECHNOLOGIES  
AG

---

Medienpartner

---

**dpunkt.verlag**

**KoSSe**  
KOMPETENZVERBUND  
SOFTWARE SYSTEMS & SERVICES

**DiWiSH**  
DIGITALE WIRTSCHAFT  
SCHLESWIG-HOLSTEIN

**CAU**  
Christian-Albrechts-Universität zu Kiel

**IHK** Kiel

# Inhaltsverzeichnis

## Wissenschaftliches Programm

**Andreas Zeller**

*Vorwort Wissenschaftliches Programm der SE 2014* ..... 21

## Software Analytics

**Thomas Zimmermann, Nachiappan Nagappan**

*Software Analytics for Digital Games* ..... 23

**Widura Schwittek, Stefan Eicker**

*A Study on Third Party Component Reuse in Java Enterprise Open Source Software ..* 25

**Ingo Scholtes, Marcelo Serrano Zanetti, Claudio Juan Tessone, Frank Schweitzer**

*Categorizing Bugs with Social Networks: A Case Study on Four Open Source Software Communities* ..... 27

**Walid Maalej, Martin Robillard**

*Patterns of Knowledge in API Reference Documentation* ..... 29

## Quality of Service

**Franz Brosch, Heiko Kozirolek, Barbora Buhnova, Ralf Reussner**

*Architecture-Based Reliability Prediction with the Palladio Component Model* ..... 31

**Norbert Siegmund, Sergiy Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, Gunter Saake**

*Performance Prediction in the Presence of Feature Interactions*..... 33



**Jons-Tobias Wamhoff, Etienne Rivière, Gilles Muller, Christof Fetzer, Pascal Felber**

*FastLane: Software Transactional Memory Optimized for Low Numbers of Threads...* 35

**Lars Grunske, Ayman Amin**

*Reactive vs. Proactive Detection of Quality of Service Problems .....* 37

**Verification**

**Antonio Filieri, Corina S. Păsăreanu, Willem Visser**

*Reliability Analysis in Symbolic Pathfinder: A brief summary.....* 39

**Dirk Beyer, Stefan Löwe, Evgeny Novikov, Andreas Stahlbauer, Philipp Wendler**

*Precision Reuse in CPAchecker .....* 41

**Christian Hammer**

*Detecting Deadlock in Programs with Data-Centric Synchronization .....* 43

**Volodymyr Kuznetsov, Johannes Kinder, Stefan Bucur, George Candea**

*Efficient State Merging in Symbolic Execution .....* 45

**Comprehension**

**Tobias Roehm, Rebecca Tiarks, Rainer Koschke, Walid Maalej**

*How Do Professional Developers Comprehend Software? .....* 47

**Zoya Durdik, Ralf Reussner**

*On the Appropriate Rationale for Using Design Patterns and Pattern Documentation* 49

**Domenico Bianculli, Carlo Ghezzi, Cesare Pautasso, Patrick Senti**

*Specification Patterns from Research to Industry: A Case Study in Service-Based Applications .....* 51

**Dominik Rost, Matthias Naab, Crescencio Lima, Christina von Flach Chavez**

*Software Architecture Documentation for Developers: A Survey .....* 53

## Evolution

**Vasilios Andrikopoulos**

*On the (Compatible) Evolution of Services* ..... 55

**Timo Kehrer**

*Generierung konsistenzhaltender Editierskripte im Kontext der Modellversionierung* 57

**Klaus Schmid**

*Ein formal fundierter Entscheidungs-Ansatz zur Behandlung von Technical Debt* ..... 59

## Synthesis

**Gerd Kainz, Christian Buckl, Alois Knoll**

*Tool Support for Integrated Development of Component-based Embedded Systems* .... 61

**Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe**

*Synthesis of Component and Connector Models from Crosscutting Structural Views*... 63

**Thomas Thüm**

*Modular Reasoning for Crosscutting Concerns with Contracts* ..... 65

**Daniel Wonisch, Alexander Schremmer, Heike Wehrheim**

*Programs from Proofs – Approach and Applications* ..... 67

## Modeling

**Stefan Wagner**

*Software-Produktqualität modellieren und bewerten: Der Quamoco-Ansatz* ..... 69

**Richard Pohl, Vanessa Stricker, Klaus Pohl**

*Messung der Strukturellen Komplexität von Feature-Modellen* ..... 71

**Robert Reicherdt, Sabine Glesner**

*Methods of Model Quality in the Automotive Area* ..... 73

**Lars Hamann, Martin Gogolla, Oliver Hofrichter**

*Zur Integration von Struktur- und Verhaltensmodellierung mit OCL* ..... 75

## **Software Architecture and Specification**

**Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolk, Indika Meedeniya**

*Software Architecture Optimization Methods: A Systematic Literature Review* ..... 77

**Christian Hammer**

*Flexible Access Control for JavaScript* ..... 79

## **Static Analysis**

**Eric Bodden, Társis Tolêdo, Márcio Ribeiro, Claus Brabrand, Paulo Borba, Mira Mezini**

*SPL<sup>LIFT</sup> – Statically Analyzing Software Product Lines in Minutes Instead of Years..* 81

**Marco Trudel**

*C nach Eiffel: Automatische Übersetzung und objektorientierte Umstrukturierung von Legacy Quelltext* ..... 83

**Ahmed Bouajjani, Egor Derevenetc, Roland Meyer**

*Robustness against Relaxed Memory Models* ..... 85

**Sebastian Eder, Maximilian Junker, Benedikt Hauptmann, Elmar Juergens, Rudolf Vaas, Karl-Heinz Prommer**

*How Much Does Unused Code Matter for Maintenance?* ..... 87

## **Specification**

**Jan Jürjens, Kurt Schneider**

*The SecReq approach: From Security Requirements to Secure Design while Managing Software Evolution* ..... 89

**James J. Hunt, Maarten De Mol, Arend Rensink**

*Noninvasive regelbasierte Graphtransformation für Java* ..... 91

**Kaituo Li, Christoph Reichenbach, Yannis Smaragdakis, Michal Young**

*Second-Order Constraints in Dynamic Invariant Inference* ..... 93

## **Testing**

**Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, Kurt Schneider**

*Revisited: Testing Culture on a Social Coding Site* ..... 95

**Dirk Beyer, Andreas Holzer, Michael Tautschnig, Helmut Veith**

*Reusing Information in Multi-Goal Reachability Analyses* ..... 97

**Alexander Wert, Jens Happe, Lucia Happe**

*Supporting Swift Reaction: Automatically Uncovering Performance Problems by Systematic Experiments* ..... 99

**Azadeh Farzan, Andreas Holzer, Niloofar Razavi, Helmut Veith**

*Concolic Testing of Concurrent Programs* ..... 101

## **Software Engineering Ideen**

**Bernd Brügge**

*Vorwort Software Engineering Ideen Programm der SE 2014* ..... 105

## **Mensch Maschine Interaktion**

**Anke Tallig**

*Sozial empathische Systeme coram publico* ..... 107

**Marc Paul, Amelie Roenspieß, Tilo Mentler, Michael Herczeg**

*The Usability Engineering Repository (UsER)* ..... 113

## **Tools**

**David Georg Reichelt, Lars Braubach**

*Sicherstellung von Performanzeigenschaften durch kontinuierliche Performanztests mit dem KoPeMe Framework ..... 119*

**Oliver Siebenmarck**

*Visualizing cross-tool ALM projects as graphs with the Open Service for Lifecycle Collaboration ..... 125*

**Martin Otto Werner Wagner**

*ACCD – Access Control Class Diagram ..... 131*

## **Code Generierung und Verifikation**

**Thorsten Ehlers, Dirk Nowotka, Philipp Sieweck, Johannes Traub**

*Formal software verification for the migration of embedded code from single- to multicore systems ..... 137*

**Malte Brunnlieb, Arnd Poetzsch-Heffter**

*Architecture-driven Incremental Code Generation for Increased Developer Efficiency ..... 143*

## **Technologietransferprogramm**

**Ralf Reussner**

*Vorwort Technologietransfer-Programm der SE 2014 ..... 151*

## **Transfer Softwarearchitektur**

**Balthasar Weitzel, Matthias Naab, Mathias Scheffe**

*Agilität braucht Architektur! ..... 153*

**Heiko Koziolk, Thomas Goldschmidt**

*Tool-Driven Technology Transfer to Support Software Architecture Decisions..... 159*

**Benjamin Klatt, Klaus Krogmann, Michael Langhammer**

*Individual Code Analyses in Practice* ..... 165

**Transferprozesse**

**Stefan Hellfeld**

*FZI House of Living Labs - interdisziplinärer Technologietransfer 2.0* ..... 171

**Andreas Metzger, Philipp Schmidt, Christian Reinartz, Klaus Pohl**

*Management operativer Logistikprozesse mit Future-Internet-Leitständen: Erfahrungen aus dem LoFIP-Projekt* ..... 177

**Steffen Kruse, Philipp Gringel**

*Ein gutes Bild erfordert mindestens 1000 Worte - Daten-Visualisierungen in der Praxis* ..... 183

**Software & Systems Engineering Essentials**

**SEE Softwareprojekte**

**Katrin Heymann**

*Releasemanagement in einem sehr komplexen Projekt* ..... 191

**Christian Werner, Ulrike Schneider**

*Open Source als Triebfeder für erfolgreiche Softwareprojekte in der öffentlichen Verwaltung* ..... 193

**Ralf Leonhard, Gerhard Pews, Simon Spielmann**

*Effiziente Erstellung von Software-Factories* ..... 195

**SEE Softwaretest**

**Gerald Zincke**

*Sieben Strategien gegen beißende Hunde* ..... 197

## **Matthias Daigl**

*Gegen den Trend? Neue Software-Teststandards ISO/IEC/IEEE 29119*..... 199

## **Workshops**

### **Robert Heinrich, Reiner Jung, Marco Konersmann, Thomas Ruhroth, Eric Schmieders**

*1st Collaborative Workshop on Evolution and Maintenance of Long-Living-Systems (EMLS14)*..... 203

### **Volker Stolz, Baltasar Trancón Widemann**

*7. Arbeitstagung Programmiersprachen (ATPS 2014)*..... 205

### **Michaela Huhn, Stefan Gerken, Carsten Rudolph**

*CeMoSS – Certification and Model-Driven Development of Safe and Secure Software* 207

### **Pit Pietsch, Udo Kelter, Jan Oliver Ringert**

*International Workshop on Comparision and Versioning of Software Models (CVSM 2014)*..... 209

### **Andrea Herrmann, Anne Hoffmann, Dieter Landes, Rüdiger Weißbach**

*Workshop „Lehre für Requirements Engineering“ (LehRE)* ..... 211

### **Ottmar Bender, Wolfgang Böhm, Stefan Henkler, Oliver Sander, Andreas Vogelsang, Thorsten Weyer**

*4. Workshop zur Zukunft der Entwicklung softwareintensiver eingebetteter Systeme (ENVISON2020)* ..... 213

## **Tutorien**

### **Thorsten Keuler, Jens Knodel, Matthias Naab**

*Tutorial: Zukunftssichere Software Systeme mit Architekturbewertung: Wann, Wie und Wieviel?*..... 217

### **Guido Gryczan, Henning Schwentner**

*Der Werkzeug-und-Material-Ansatz für die Entwicklung interaktiver Software-Systeme* ..... 219

**Simon Grapenthin, Matthias Book, Volker Gruhn**

*Früherkennung fachlicher und technischer Projektrisiken mit dem Interaction Room..* 221

**Doktorandensymposium**

**Michaela Gluchow**

*AGREEMENT - An Approach for Agile Rationale Management .....* 225

**Christian Wulf**

*Pattern-Based Detection and Utilization of Potential Parallelism in Software Systems .....* 229

**Max E. Kramer**

*Synchronizing Heterogeneous Models in a View-Centric Engineering Approach.....* 233

**Emitza Guzman**

*Summarizing, Classifying and Diversifying User Feedback.....* 237







## **Wissenschaftliches Programm**



## **Vorwort Wissenschaftliches Programm der SE 2014**

Mit dem neuen Format für das wissenschaftliche Programm bläst ein frischer Wind durch die deutschsprachige SE-Konferenz. Mit 41 Vorträgen aus den Spitzenkonferenzen und Fachzeitschriften der Softwaretechnik erwartet Sie ein spannendes Programm, das die ganze Breite und Tiefe aktueller Forschung abdeckt und es in jeder Hinsicht mit den besten internationalen Konferenzen aufnehmen kann.

Diese Qualitätsoffensive kommt nicht von ungefähr. Lange hatte die SE um Original-Beiträge gebeten und eingeladen, und trotz wohlgemeinter Aufrufe und Ermahnungen nie die gleiche Qualität der Einreichungen erhalten, wie wir sie von Spitzenkonferenzen und -Zeitschriften kennen. Für die SE 2014 haben wir deshalb einen neuen Ansatz gewagt – und gewonnen. Ein “Best-Of” sollte es sein, ein Schaufenster, in dem sich die besten SE-Beiträge der Community sammeln würden. Wer immer in den letzten zwei Jahren einen Beitrag auf einer der Spitzenkonferenzen oder einem der Spitzenjournale der Softwaretechnik veröffentlicht hatte, sollte Gelegenheit haben, ihre oder seine Arbeit noch einmal der Community in Kiel zu präsentieren.

Die Regeln für die Einreichung waren schnell aufgestellt: Der Vortragsvorschlag musste sich auf einen Beitrag beziehen, der auf einer internationalen Konferenz (unter Beteiligung der ACM SIGSOFT) oder IEEE TSE oder ACM TOSEM erschienen war; vergleichbare Konferenzen und Zeitschriften waren ebenfalls zugelassen. Neben dem Beitrag mussten die Autoren lediglich eine kurze Vortragszusammenfassung von maximal 200 Wörtern einreichen. Die Hürde für bereits erfolgreiche Autoren war so denkbar niedrig.

Wir identifizierten 268 Beiträge, deren Autoren als Einreicher in Frage kamen (Danke an Andrey Tarasevich für seine Mithilfe!), und sandten Ankündigungen an alle 455 Autorinnen und Autoren. Insgesamt erhielten wir 58 Vortragsvorschläge (von denen sich einige auf mehrere Beiträge bezogen). Die meisten kamen, wie erwartet, von SE-Konferenzen; wir erhielten aber auch spannende Einreichungen aus Spitzenkonferenzen der SE nahestehenden Communities wie Programmiersprachen oder Mensch-Maschine-Interaktion.

Nun schlug die Stunde des Programmkomitees, das aus diesen 58 Beiträgen auswählen durfte. Aber auch hier war der Prozess selten unkompliziert: Harald Gall, Willi Hasselbring, Mira Mezini, Klaus Pohl, Ralf Reussner, Wilhelm Schäfer und Walter Tichy durften jeweils bis zu 15 Vortragsvorschläge nominieren, die sei gerne auf der Konferenz sehen würden. Es stellte sich heraus, dass wir alle 41 der 58 Beiträge, die wenigstens eine Nominierung erhielten, im Programm unterbringen können würden – und so war die Arbeit schneller getan als erwartet.

Insgesamt spiegelt das Programm aktuelle Trends der internationalen SE-Konferenzen wider: Analyse, Evolution und Architektur, alle eingesetzt, um Qualität und Produktivität

zu steigern. Was mich besonders freut: Viele der Autorinnen und Autoren zeigen sich zum ersten Mal auf der SE. Damit können wir hoffentlich nicht nur die bestehende Community, sondern eine ganz neue Generation für die Forschung im deutschsprachigen Raum begeistern – und nicht zuletzt weit außerhalb unseres Gebietes für Softwaretechnik werben.

Ich freue mich auf spannende Vorträge, aktuelle Themen, und inspirierende Gespräche in Kiel. Lang lebe die Forschung der Softwaretechnik – lang lebe die SE!

Andreas Zeller

Leiter des Programmkomitees, SE 2014

# Software Analytics for Digital Games

Thomas Zimmermann, Nachiappan Nagappan

Microsoft Research  
One Microsoft Way, Redmond, WA 98052, USA  
tzimmer@microsoft.com  
nachin@microsoft.com

**Abstract:** In this talk, we will summarize our effort in the area of software analytics with a special focus on digital games. We will first introduce software analytics and show important information needs for data scientists working on software data. Next we will present, several examples of how analytics can be used on digital games such as how players are engaged in Project Gotham Racing and how skill develops over time in Halo Reach. We will also point out important differences between games development and traditional software development.

## 1 Software Analytics

Software Analytics is a subfield of analytics with the focus on software data. Davenport, Harris, and Morison [DHM10] define analytics “*as the use of analysis, data, and systematic reasoning to make decisions.*” Software data can take many forms such as source code, changes, bug reports, code reviews, execution data, user feedback, and telemetry information. Analysis of software data has a long tradition [MT13] in empirical software engineering, software reliability, and mining software repositories communities.

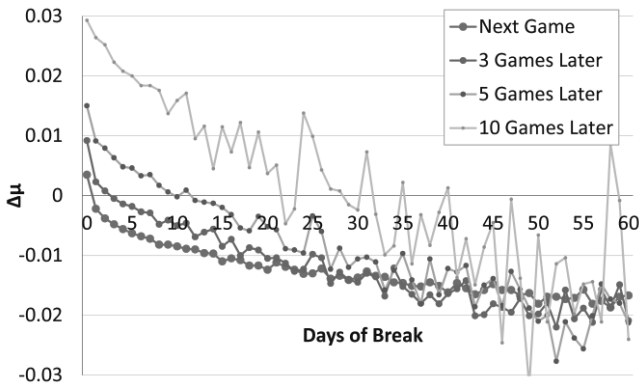
Since its inception in 2005, the mission of the Empirical Software Engineering (ESE) group at Microsoft Research is to “*empower software development teams to make sound data-driven decisions by deploying novel analytics tools and methods based on ESE’s empirical research on products, process, people, and customers*” [ESE13]. The work by ESE includes specific guidelines and recommendations for analytics tools [BZ12] and data scientists [AB13]. In this talk, we focus on analytics for digital games: player engagement in Project Gotham Racing [Hu12] and skill development in Halo Reach [Hu13].

## 2 Skill in Halo Reach

As an example of analytics for digital games, we describe a project on characterizing the skill of players in Halo Reach. We selected a cohort of 3.2 million players who started playing the game in the release week. To quantify the skill of each player we used the TrueSkill rating. We analyzed the skill time series for different groups of players in the cohort, focussing on play intensity, *play breaks* (summarized below), other titles played, as well as skill change and retention. The complete findings are in a CHI paper [Hu13].

Figure 1 shows behaviors that players exhibit after breaks. The change in skill from before the break to after the break is illustrated by the 4 lines representing the next 1, 3, 5, and 10 matches after the break. When players are not taking breaks (breaks of 0 days), skill generally increases, as evidenced by the climbing intercepts on the y-axis. Breaks of 1–2 days correlate with a small drop in skill in the next match played after the break, but have little long-term effect. Longer breaks correlate with larger skill decreases, but the relationship is not linear. More concretely, a 30 day break correlates with a skill drop of 10 matches of play; this is shown by the intersection of the 10 games later line with the x-axis. Thus, the amount of time required to regain skill following a 30 day break is only about 3 hours of gameplay (matches are typically 15 minutes). Insights like this can be used to improve matchmaking algorithms as well as retention of players.

**Acknowledgements.** Thanks to Tim Menzies, Emerson Murphy-Hill, Ken Hullett, Sauvik Das, Jeff Huang, Thomas Debeauvais, and Gifford Cheung as well as our collaborators at Xbox and IEB.



**Figure 1.** Skill change after different lengths of breaks for the next match, 3 matches after, 5 matches after, and 10 matches after the break.

## References

- [DHM10] Thomas H. Davenport, Jeanne G. Harris, Robert Morison. Analytics at Work: Smarter Decisions, Better Results. Harvard Business Review Press, 2010.
- [MT13] Tim Menzies and Thomas Zimmermann. Software Analytics: So What? IEEE Software, 30, 4 (July 2013), 31-37.
- [Hu12] Kenneth Hullett, Nachi Nagappan, Eric Schuh, and John Hopson, Empirical analysis of user data in game software development, in Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM), 2012.
- [Hu13] Jeff Huang, Thomas Zimmermann, Nachiappan Nagappan, Charles Harrison, and Bruce Phillips, Mastering the Art of War: How Patterns of Gameplay Influence Skill in Halo, in Proceedings of the Intl. Conf. on Human Factors in Computing Systems (CHI), 2013.
- [ESE13] Empirical Software Engineering Group (ESE), <http://aka.ms/esegroup>
- [BZ12] Raymond P.L. Buse and Thomas Zimmermann, Information Needs for Software Development Analytics, in Proc. of Intl. Conf. on Software Engineering (ICSE), 2012.
- [AB13] Andrew Begel and Thomas Zimmermann, Analyze This! 145 Questions for Data Scientists in Software Engineering, Tech. Report, no. MSR-TR-2013-111, 28 Oct 2013.

# A Study on Third Party Component Reuse in Java Enterprise Open Source Software

Widura Schwittek, Stefan Eicker

paluno – The Ruhr Institute for Software Technology  
University of Duisburg-Essen  
Universitätsstr. 9  
45141 Essen

widura.schwittek@paluno.uni-due.de  
stefan.eicker@paluno.uni-due.de

**Abstract:** Recent studies give empirical evidence that much of today's software is to a large extent built on preexisting software, such as commercial-off-the-shelf (COTS) and open source software components. In this exploratory study we want to contribute to this small but increasing body of knowledge by investigating third party component reuse in 36 Java web applications that are open source and are meant to be used in an enterprise context. Our goal is to get a better understanding on how third party components are reused in web applications and how to better support it. The results are in line with existing research in this field. 70 third party components are being reused on average. 50 percent of the 40 most reused third party components are maintained by the Apache Foundation. Further research questions based on the study results were generated and are presented at the end of this paper.

## Summary

Reusing third party components has become a key success factor in software development projects [Ga08] leading to reduced costs, faster time-to-market and better software quality [Li09]. Recent studies ([He11], [RvV12]) give empirical evidence that much of today's software is indeed to a large extent build on preexisting software, such as commercial-off-the-shelf (COTS) and open source software components. This especially holds true for web applications, where architects and developers are faced with the steady proliferation of new technologies and standards [Ro08].

In this exploratory study which is described in more detail in [SE13] we want to contribute to this small but increasing body of knowledge of third party component reuse as part of the larger field of software reuse. For this we have analyzed 36 end-user products such as content management (e.g. Liferay, Magnolia) or business intelligence systems (e.g. Pentaho). We focused on black-box reuse in Open Source Java web applications that are meant to be used in an enterprise context. As such, all identified third party components (e.g. Apache Commons, dom4j, log4j, Lucene) also follow an open source license which excludes COTS components from this study. In this first step we did not consider references to components which are part of the runtime environment



such as the JDK or application servers. We also excluded non-Java based components such as the JavaScript library JQuery. A small tool has been developed in order to retrieve the reuse data by extracting, transforming and analyzing deployment artifacts in an automatic fashion. In short, it looks for characteristics such as the filenames and the checksum of files to identify the name and the version of a reused third party component.

During the identification of third party components in 36 Java based web applications 3311 different artifacts were considered leading to 651 unique third party components. 70 third party components are reused on average ranging from a minimum of 16 (Agorum) to a maximum of 161 components (Alfresco) per web application. The results are comparable to the existing studies, underlining that software reuse happens in this specific way of third party component reuse, independent from the kind of reuse (calling, extending, inheriting, instantiating).

As part of this exploratory study these questions were raised from looking at the data which are worthwhile to be asked as part of our future research: Do integrators keep track of the huge amount of reused third party components, especially with respect to bugfix and security updates and what are the risks of not doing so? Specific findings in the data might be of interest to integrators during their selection process: Some third party components are reused by almost all of the analyzed web applications. 50 percent of the 40 most reused third party components are maintained by the Apache Foundation. Certain reuse characteristics were sepecific to particular web application domains. How can this knowledge be used to support the selection process of third party components? Having laid the foundation with this work, we further look into how valuable findings in the data set can be made available to integrators. We already started to create component recommendations to support the component identification and selection phase [SE12].

## References

- [Ga08] The Evolving Open-source Software Model. Predicts from December 2008. Gartner.
- [He11] Heinemann, L. et al.: On the Extent and Nature of Software Reuse in Open Source Java Projects. In (Schmid, K. Ed.): Top Productivity through Software Reuse. ICSR 2011, Pohang, South Korea, June 13-17, 2011. Proceedings. Springer Berlin Heidelberg, 2011; pp. 207–222.
- [Li09] Li, J. et al.: Development with Off-the-Shelf Components: 10 Facts. In IEEE Software, 2009, 26; pp. 80–87.
- [Ro08] Rossi, G.: Web engineering. Modelling and implementing web applications. Springer, London, 2008.
- [RvV12] Raemaekers, S.; van Deursen, A.; Visser, J.: An Analysis of Dependence on Third-party Libraries in Open Source and Proprietary Systems: Proceedings of the Sixth International Workshop on Software Quality and Maintainability (SQM 2012), 2012.
- [SE12] Schwittek, W.; Eicker, S.: Decision Support for Off-the-Shelf Software Selection in Web Development Projects. In (Grossniklaus, M. Ed.): Current trends in web engineering. Springer, Berlin, 2012; pp. 238–243.
- [SE13] Schwittek, W.; Eicker, S.: A study on third party component reuse in Java enterprise open source software. In (Kruchten, P.; Giannakopoulou, D.; Tivoli, M. Eds.): Proceedings of the 16th International ACM Sigsoft symposium on Component-based software engineering. ACM, New York, NY, 2013; pp. 75–80.

# Categorizing Bugs with Social Networks: A Case Study on Four Open Source Software Communities

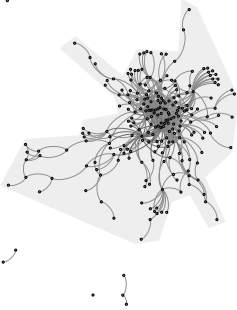
Ingo Scholtes, Marcelo Serrano Zanetti, Claudio Juan Tessone, Frank Schweitzer

Chair of Systems Design  
ETH Zürich, Switzerland  
{ischoltes, mzanetti, tessonec, fschweitzer}@ethz.ch

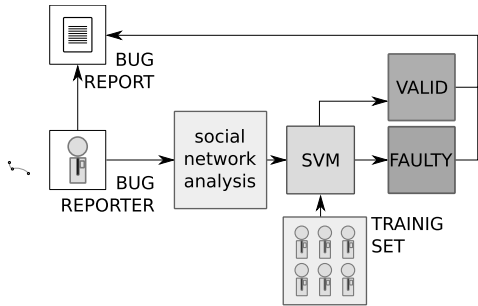
**Abstract:** Efficient bug triaging procedures are an important precondition for successful collaborative software engineering projects. Summarizing the results of a recent study [ZSTS13], in this paper we present a method to automatically identify *valid* bug reports which a) contain enough information to be reproduced, b) refer to actual software issues, and c) are not duplicates. Focusing on the social dimension of bug handling communities, we use network analytic measures to quantify the position of bug reporters in the collaboration networks of Open Source Software (OSS) communities. Based on machine learning techniques we then use these measures to predict whether bugs reported by users will eventually be identified as *valid*. A study on a large-scale data set covering more than 700,000 bug reports that have been collected from the BUGZILLA installations of four major OSS communities shows that our method achieves a remarkable precision of up to 90%.

In large collaborative software engineering projects, the process of triaging, categorizing and prioritizing bug reports can become a laborious and difficult task that consumes considerable resources. The magnitude of this problem calls for (semi-)automated techniques that assist bug handling communities in the filtering of important bug reports. Due to the importance for practitioners, different approaches for the automated classification of bug reports have been studied, most of which have been focused on the information provided in the bug report itself. Fewer studies have focused on human aspects like, e.g., coordination patterns or the reputation of bug reporters. Based on data covering the full history of 700,000 bug reports in the BUGZILLA installation of the four OSS projects ECLIPSE, NETBEANS, FIREFOX and THUNDERBIRD, in this work we study whether quantitative measures for the position of bug reporters in a project's social organization can be used to predict whether reported bugs will eventually be classified as helpful by the community. Our approach is based on the extraction of evolving collaboration networks from time-stamped collaboration events between community members that can be inferred from a forwarding of information (i.e. updates in the CC field of bug reports) as well as the assignment of tasks (i.e. updates of the ASSIGNEE field of bug reports). By means of a sliding time window with a width of 30 days and an increment of one day, we build evolving monthly collaboration networks for each of the four studied communities. An example for such a monthly collaboration network can be seen in Figure 1(a). For each user reporting a bug at time  $t$  we then compute nine quantitative measures that capture the social position of the reporting user in the collaboration network for the 30 days pre-

ceding  $t$ . Based on their final status when they were closed, we categorize all bug reports either as *valid* (final status FIXED or WONTFIX) or *faulty* (final status INVALID, INCOMPLETE or DUPLICATE). Based on a random subset of 5% of all reports we use this information to train a support vector machine and - using the nine network-analytic measures - utilize the trained machine to predict which of the remaining bug reports will eventually be identified as *valid* by the community. We then use the ground truth in our data set to evaluate the precision and recall of our prediction. The evaluation results of this method are shown in Table 1. Our prediction method achieves a remarkable high precision ranging between 78 and 90 percent. Remarkably, for communities in which the fraction of *valid* reports is as low as 21%, our classifier still achieves a precision of more than 80%. For a detailed description of our methods and data sets, the network measures used in our study, as well as the full discussion of results we refer the reader to [ZSTS13].



(a) Collaboration network covering June 2006 in the NETBEANS Bugzilla community



(b) Outline of the prediction methodology

Table 1: Precision ( $p$ ) and recall ( $r$ ) of prediction of valid bug reports based on social networks

	FIREFOX	THUNDERBIRD	ECLIPSE	NETBEANS
Valid reports	21.0%	23.3%	74.3%	62.4%
$p$	82.5%	90.3%	88.7%	78.9%
$r$	44.5%	38.9%	91.0%	87.0%

In summary, we show that the social layer of support infrastructures like BUGZILLA contains valuable information about the reputation and/or abilities of community members that can be used to efficiently mitigate the information overload in bug handling communities. Our study highlights the potential of quantitative measures of social organization in collaborative software engineering and opens interesting perspectives for the integration of network analysis in the design of support infrastructures and social information systems.

## References

- [ZSTS13] Marcelo Serrano Zanetti, Ingo Scholtes, Claudio Juan Tessone, and Frank Schweitzer. Categorizing bugs with social networks: A case study on four open source software communities. In *Proceedings of the 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 1032–1041, 2013.

# Patterns of Knowledge in API Reference Documentation

Walid Maalej  
Fachbereich Informatik  
Universität Hamburg  
maalej@informatik.uni-hamburg.de

Martin P. Robillard  
School of Computer Science  
McGill University  
martin@cs.mcgill.ca

**Abstract:** Das Lesen der Referenzdokumentation ist ein wichtiger Teil der Entwicklungsarbeit mit API Programmierschnittstellen (Application Programming Interface). Die Referenzdokumentation stellt zusätzliche wichtige Informationen zur Verfügung, die nicht direkt aus der Syntax der API abgeleitet werden können. Um die Qualität der Referenzdokumentationen zu verbessern und die Effizienz des Zugriffes auf die darin enthaltenen Informationen zu steigern, muss zuerst der Inhalt dieser Dokumentation untersucht und verstanden werden. Diese Studie untersucht ausführlich, die Natur und die Organisation vom Wissen, welches in der Referenzdokumentation von Hunderten APIs der Haupttechnologien Java SDK 6 und Net 4.0 beinhaltet ist. Unsere Studie besteht zum einen aus der Entwicklung einer Taxonomie der Wissensarten in Referenzdokumentationen mit Hilfe einer systematischen Auswertung qualitativer Daten (Grounded Theory) und zum anderen aus der unabhängigen empirischen Validierung. Siebzehn trainierte Coders haben die Taxonomie verwendet um insgesamt 5574 zufällig ausgewählte Dokumentationseinheiten auszuwerten. Untersucht wurde hauptsächlich, ob die Dokumentationseinheit bestimmte Wissensarten aus der Taxonomie beinhaltet. Die Ergebnisse bieten eine ausführliche Analyse von Wissensmustern in Referenzdokumentationen. Dazu zählen Beobachtungen über die Wissensarten und wie diese über die Gesamtdokumentation verteilt sind. Sowohl die Taxonomie als auch die Wissensmuster können verwendet werden, um Entwicklern zu helfen, die Inhalte ihrer Referenzdokumentation zu bewerten, besser zu organisieren und überflüssige Inhalte zu vermeiden. Zusätzlich, stellt die Studie ein Vokabular zur Verfügung, das verwendet werden kann, um Diskussionen über die API- Schnittstellen zu strukturieren und effizienter zu führen.

## Literatur

- [MR13] Walid Maalej und Martin P. Robillard. Patterns of Knowledge in API Reference Documentation. *IEEE Transactions on Software Engineering*, 39(9):1264–1282, September 2013.



# **Architecture-Based Reliability Prediction with the Palladio Component Model**

Franz Brosch, Heiko Koziol, Barbora Buhnova, Ralf Reussner

FZI Karlsruhe, Germany ABB Corporate Research, Germany  
Masaryk University, Czech Republic  
Karlsruhe Institute of Technology, Germany  
heiko.koziol@de.abb.com

Software-intensive systems are increasingly used to support critical business and industrial processes, such as in business information systems, e-business applications, or industrial control systems. The reliability of a software system is defined as the probability of failure-free operation of a software system for a specified period of time in a specified environment. To manage reliability, reliability engineering gains its importance in the development process. Reliability is compromised by faults in the system and its execution environment, which can lead to different kinds of failures during service execution: Software failures occur due to faults in the implementation of software components, hardware failures result from unreliable hardware resources, and network failures are caused by message loss or problems during inter-component communication.

To support fundamental design decisions early in the development process, architecture-based reliability prediction can be employed to evaluate the quality of system design, and to identify reliability-critical elements of the architecture. Existing approaches suffer from the following drawbacks that limit their applicability and accuracy.

First, many approaches do not explicitly model the influence of the system usage profile (i.e., sequences of system calls and values of parameters given as an input to these calls) on the control and data flow throughout the architecture, which in turn influences reliability. For example, if faulty code is never executed under a certain usage profile, no failures occur, and the system is perceived as reliable by its users. Existing models encode a system usage profile implicitly into formal models, typically in terms of transition probabilities in the Markov Models characterizing the execution flow among components. Since the models are tightly bound to the selected usage profile, evaluating reliability for a different usage profile requires repeating much of the modeling effort.

Second, many approaches do not consider the reliability impact of a systems execution environment. Even if the software is totally free of faults, failures can occur due to unavailability of underlying hardware resources and communication failures across network links. Neglecting these factors tends to result in less accurate and overoptimistic reliability prediction. On the other hand, approaches that do consider the execution environment typically offer no means to model application-level software failures, which also results in a limited view of software system reliability.

Third, many approaches use Markov models as their modeling notation, which is not aligned with concepts and notations typically used in software engineering (e.g., UML or SysML). They represent the system through a low-level set of states and transition probabilities between them, which obscures the original software-engineering semantics. Direct creation and interpretation of Markov models without any intermediate notation may be uncomfortable and hard to accomplish for software developers, especially when it is to be done repeatedly during the development process.

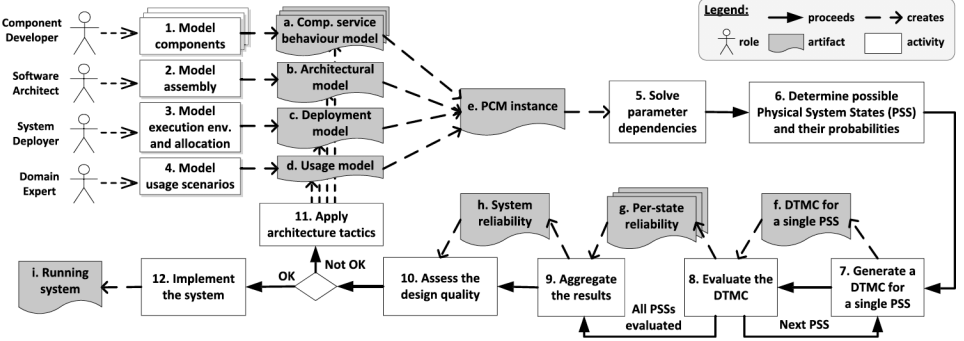


Figure 1: Palladio Component Model Reliability Prediction Approach

Our contribution is a novel technique for architecture-based software reliability modeling and prediction that explicitly considers and integrates the discussed reliability-relevant factors [BKBR12]. The technique offers usage profile separation and propagation through the concept of parameter dependencies [Koz08] and accounts for hardware unavailability through reliability evaluation of service execution under different hardware availability states. We realize the approach as an extension of the Palladio Component Model (PCM) [BKR09], which offers a UML-like modeling notation. We provide tool support for an automated transformation of PCMs into Markov chains and space-effective evaluation of these chains. We discuss how software engineers can use architecture tactics to systematically improve the reliability of the software architecture. Furthermore, we validate the approach in two case studies.

## References

- [BKBR12] Franz Brosch, Heiko Kozirolek, Barbora Buhnova, and Ralf Reussner. Architecture-Based Reliability Prediction with the Palladio Component Model. *IEEE Transactions on Software Engineering*, 38(6):1319–1339, November 2012.
- [BKR09] Steffen Becker, Heiko Kozirolek, and Ralf Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, January 2009.
- [Koz08] Heiko Kozirolek. *Parameter Dependencies for Reusable Performance Specifications of Software Components*. PhD thesis, University of Oldenburg, Germany, March 2008.

# Performance Prediction in the Presence of Feature Interactions

– Extended Abstract –

Norbert Siegmund,<sup>1</sup> Sergiy Kolesnikov,<sup>1</sup> Christian Kästner,<sup>2</sup> Sven Apel,<sup>1</sup> Don Batory,<sup>3</sup>  
Marko Rosenmüller,<sup>4</sup> and Gunter Saake<sup>4</sup>

<sup>1</sup>University of Passau, Germany, <sup>2</sup>Carnegie Mellon University, USA

<sup>3</sup>University of Texas at Austin, USA, <sup>4</sup>University of Magdeburg, Germany

**1 Introduction.** Customizable programs and program families provide user-selectable *features* allowing users to tailor the programs to the application scenario. Beside functional requirements, users are often interested in non-functional requirements, such as a binary-size limit, a minimized energy consumption, and a maximum response time. To tailor a program to non-functional requirements, we have to know in advance which feature selection, that is, configuration, affects which non-functional properties. Due to the combinatorial explosion of possible feature selections, a direct measurement of all of them is infeasible.

In our work, we aim at *predicting* a configuration’s non-functional properties for a specific workload based on the user-selected features [SRK<sup>+</sup>11, SRK<sup>+</sup>13]. To this end, we quantify the influence of each selected feature on a non-functional property to compute the properties of a specific configuration. Here, we concentrate on performance only. Unfortunately, the accuracy of performance predictions may be low when considering features only in isolation, because many factors influence performance. Usually, a property is program-wide: it emerges from the presence and interplay of multiple features. For example, database performance depends on whether a search index or encryption is used and how both features interplay. If we knew how the combined presence of two features influences performance, we could predict a configuration’s performance more accurately. Two features *interact* (i.e., cause a performance interaction) if their simultaneous presence in a configuration leads to an unexpected performance, whereas their individual presences do not.

We improve the accuracy of predictions in two steps: (i) We detect which features interact and (ii) we measure to what extent they interact. In our approach, we aim at finding the sweet spot between prediction accuracy, measurement effort, and generality in terms of being independent of the application domain and the implementation technique. The distinguishing property of our approach is that we neither require domain knowledge, source code, nor complex program-analysis methods, and our approach is not limited to special implementation techniques, programming languages, or domains.

Our evaluation is based on six real-world case studies from varying domains (e.g., databases, encoding libraries, and web servers) using different configuration techniques. Our experiments show an average prediction accuracy of 95 percent, which is a 15 percent improvement over an approach that takes no interactions into account [SKK<sup>+</sup>12].



**2 Approach.** We detect feature interactions in two steps: (a) We identify which features interact and (b) with heuristics, we search for the combination of these *interacting features* to pin down the actual feature interactions. Next, we give an overview of both steps.

**Detecting Interacting Features.** To identify which features interact, we quantify the performance contribution of each feature. Our idea is as follows: First, we determine a feature’s performance contribution in isolation (i.e., how a feature influences a program’s performance when no other feature is present) – called minimal delta. Second, we determine a feature’s contribution when combined with all other features – called maximum delta. Finally, we compare for each feature its minimal and maximal delta. Our assumption is, if the deltas differ, then there must be, at least, one other feature that is responsible for this change. After applying this approach to all features, we know which features interact (but not in which specific combinations). The remaining task is to determine which combinations of these interacting features cause an actual feature interaction.

**Heuristics to Detect Feature Interactions.** To pin down performance feature interactions, we developed three heuristics based on our experience with product lines and previous experiments. We identify a feature interaction by predicting the performance of a certain feature combination and comparing the prediction against the actually measured performance. If the difference exceeds a certain threshold (e.g., to compensate for measurement bias), we found a feature interaction. Next, we shortly describe these heuristics.

- **Pair-Wise Interactions (PW)** – We assume that pair-wise interactions are the most common form of non-functional feature interactions. Hence, we measure all pair-wise combinations of interacting features (i.e., *not* all features) and compare them with our predictions to detect interactions.
- **Higher-Order Interactions (HO)** – We assume that triple-wise feature interactions can be predicted by analyzing already detected pair-wise interactions. The rationale is, if three features interact pair wise in any combination, they likely participate also in a triple-wise interaction.
- **Hot-Spot Features (HS)** – We assume the existence of *hot-spot* features. In previous experiments, we found that there are usually few features that interact with many features and there are many features that interact only with few features. Hence, we perform additional measurements to locate interactions of hot-spot features.

We performed a series of experiments with the six real-world case studies Berkeley DB Java, Berkeley DB C, SQLite, Apache web server, LLVM compiler infrastructure, and x264 video encoder. We found that applying these heuristics improves prediction accuracy from 80 %, on average, to 95 %, on average, which is within the measurement error.

## References

- [SKK<sup>+</sup>12] N. Siegmund, S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saake. Predicting Performance via Automated Feature-Interaction Detection. In *Proc. ICSE*, pages 167–177. IEEE, 2012.
- [SRK<sup>+</sup>11] N. Siegmund, M. Rosenmüller, C. Kästner, P. Giarrusso, S. Apel, and S. Kolesnikov. Scalable Prediction of Non-functional Properties in Software Product Lines. In *Proc. SPLC*, pages 160–169. IEEE, 2011.
- [SRK<sup>+</sup>13] N. Siegmund, M. Rosenmüller, C. Kästner, P. Giarrusso, S. Apel, and S. Kolesnikov. Scalable Prediction of Non-functional Properties in Software Product Lines: Footprint and Memory Consumption. *Information and Software Technology*, 55(3):491–507, 2013.

# FASTLANE: Software Transactional Memory Optimized for Low Numbers of Threads

Jons-Tobias Wamhoff  
Christof Fetzer  
*Technische Universität Dresden,  
Germany*  
first.last@tu-dresden.de

Etienne Rivière  
Pascal Felber  
*Université de Neuchâtel,  
Switzerland*  
first.last@unine.ch

Gilles Muller  
*INRIA, France*  
first.last  
@inria.fr

Software transactional memory (STM) can lead to scalable implementations of concurrent programs, as the relative performance of an application increases with the number of threads that support it. However, the absolute performance is typically impaired by the overheads of transaction management and instrumented accesses to shared memory. This often leads a STM-based program with a low thread count to perform worse than a sequential, non-instrumented version of the same application.

We propose FASTLANE [WFF<sup>+</sup>13], a new STM system that bridges the performance gap between sequential execution and classical STM algorithms when running on few cores (see Figure 1). FASTLANE seeks to reduce instrumentation costs and thus performance degradation in its target operation range. We introduce a family of algorithms that differentiate between two types of threads: One thread (the *master*) is allowed to commit transactions without aborting, thus with minimal instrumentation and management costs and at nearly sequential speed, while other threads (the *helpers*) execute speculatively. Helpers typically run slower than STM threads, as they should contribute to the application progress without impairing on the performance of the master (in particular, helpers never cause aborts for the master’s transactions) in addition to performing the extra bookkeeping associated with memory accesses.

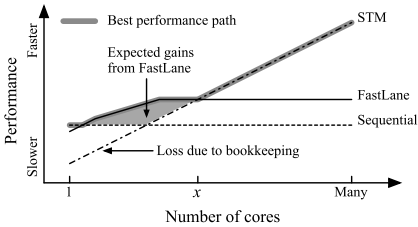


Figure 1: Bridging the gap between sequential and STM performance for few threads.

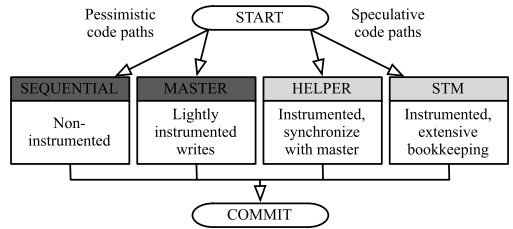


Figure 2: Code path for transaction selected dynamically at start by runtime system.

FASTLANE is implemented within a state-of-the-art STM runtime and compiler. Multiple code paths are generated for execution (see Figure 2): sequential on a single core, FASTLANE (master and helper) for few cores, and STM for many cores. Applications can

dynamically select a variant at runtime, depending on the number of cores available for execution.

FASTLANE almost systematically wins over a classical STM in the 2-4 threads range, and often performs better than sequential execution of the non-instrumented version of the same application. Figure 3 shows our results for the STAMP Vacation benchmark, an online travel reservation system. We compare against TML, which is based on a global versioned readers-writer lock, NOREC, which extends TML with buffered updates and value-based validation, and TINYSTM, an implementation of the lazy snapshot algorithm with time-based validation using an array of versioned locks.

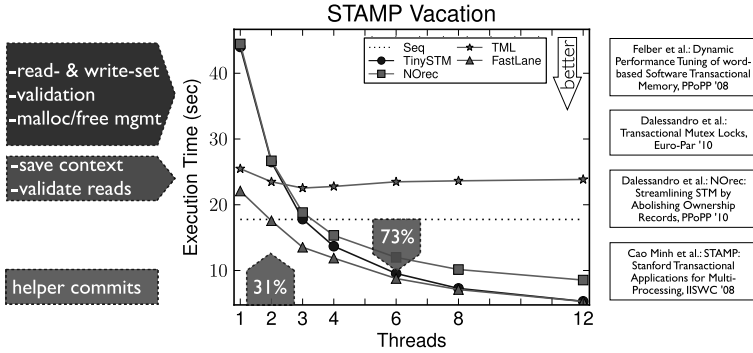


Figure 3: Comparison against different STM implementation with the STAMP Vacation benchmark.

The overheads compared to the single master thread origin from saving the context at transaction start and validating all reads during the execution of the transaction (for TML, NOREC and TINYSTM). Fully optimistic STM implementation additionally suffer from overheads due to maintaining a read-set and write-set, which must be validated for conflicts with other threads at commit time, and tracking dynamically managed memory (for NOREC and TINYSTM). The FASTLANE master must only acquire a global lock and reflect all its updates in the meta-data and achieves a performance close to the sequential uninstrumented execution.

The scalability for few threads is achieved by activating FASTLANE’s speculative helper threads that maintain a read-set and write-set. Figure 3 shows their increasing share of the total number of commits the more threads are enabled. TML does not scale because all threads abort and wait as long as a single update transaction is active.

## References

- [WFF<sup>+</sup>13] Jons-Tobias Wamhoff, Christof Fetzer, Pascal Felber, Etienne Rivière, and Gilles Muller. FastLane: Improving Performance of Software Transactional Memory for Low Thread Counts. In *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP ’13, pages 113–122, New York, NY, USA, 2013. ACM.

# Reactive vs. Proactive Detection of Quality of Service Problems

Lars Grunske<sup>a</sup>, Ayman Amin<sup>b</sup>

<sup>a</sup>Institute of Software Technology, University of Stuttgart, Germany

<sup>b</sup>Faculty of ICT, Swinburne University of Technology, Australia

**Abstract:** This paper summarizes our earlier contributions on reactive and proactive detection of quality of service problems. The first contribution is applying statistical control charts to reactively detect QoS violations. The second contribution is applying time series modeling to proactively detect potential QoS violations.

## 1 Introduction

Software systems may suffer at runtime from changes in their operational environment or/and requirements specification, so they need to be adapted to satisfy the changed environment or/and specifications [CGK<sup>+</sup>11]. The research community has developed a number of approaches to building adaptive systems that respond to these changes such as Rainbow [GCH<sup>+</sup>04]. Currently, several approaches have been proposed to monitor QoS attributes at runtime with the goal of reactively detecting QoS violations (e.g. [MP11]). We will present our reactive and proactive techniques in the following.

## 2 Reactive detection of QoS violations based on control charts

The reactive approaches detect QoS violations by observing the running system and determining QoS values [MP11] and checking if they exceed a predefined threshold. The main limitation of these approaches is that they do not have statistical confidence in detecting QoS violations. To address this limitation, we propose a statistical approach [ACG11, ACG12b] based on control charts for the runtime detection of QoS violations. This approach consists of four phases: (1) Estimating the running software system capability (current normal behavior) in terms of descriptive statistics, i.e. mean, std, and confidence interval of the QoS attributes; (2) Building a control chart (esp. CUSUM) using given QoS requirements; (3) After each new QoS observation, updating the chart statistic and checking for statistically significant violations; (4) In case of detecting violations, providing warning signals.

## 3 Proactive Detection of QoS violations based on time series modeling

Predicting future values of Quality of Service (QoS) attributes can assist in the control of software intensive systems by preventing QoS violations before they happen. Currently, many approaches prefer ARIMA models for this task, and assume the QoS attributes' be-

havior can be linearly modeled. However, our analysis of real QoS datasets shows that they are characterized by a highly dynamic and mostly nonlinear behavior with volatility clustering (time-varying variation) to the extent that existing ARIMA models cannot guarantee accurate QoS forecasting, which can introduce crucial problems such as proactively triggering unrequired adaptations and thus leading to follow-up failures and increased costs. To address this limitation, we propose two automated forecasting approaches based on time series modeling. The first forecasting approach [AGC12] addresses the nonlinearity characteristic of QoS values. This forecasting approach integrates linear and nonlinear time series models [BJ76] and automatically, without human intervention, selects and constructs the best suitable forecasting model to fit the QoS attributes' dynamic behavior and provide accurate forecasting for QoS measures and violations. The second forecasting approach [ACG12a] addresses the QoS volatility by exploiting the ability of generalized autoregressive conditional heteroscedastic (GARCH) models to model the high volatility [Eng82]. This approach basically integrates ARIMA and GARCH models to capture the QoS volatility and provide accurate forecasting for QoS measures and violations. Using real-world QoS datasets of Web services we evaluate the accuracy and performance aspects of the proposed forecasting approaches.

## References

- [ACG11] Ayman Amin, Alan Colman, and Lars Grunske. Using Automated Control Charts for the Runtime Evaluation of QoS Attributes. In *Proc. of the 13th IEEE Int. High Assurance Systems Engineering Symposium*, pages 299–306. IEEE Computer Society, 2011.
- [ACG12a] Ayman Amin, Alan Colman, and Lars Grunske. An Approach to Forecasting QoS Attributes of Web Services Based on ARIMA and GARCH Models. In *Proc. of the 19th Int. Conf. on Web Services*, pages 74–81. IEEE, 2012.
- [ACG12b] Ayman Amin, Alan Colman, and Lars Grunske. Statistical Detection of QoS Violations Based on CUSUM Control Charts. In *Proc. of the 3rd ACM/SPEC Int. Conf. on Performance Engineering*, pages 97–108. ACM, 2012.
- [AGC12] Ayman Amin, Lars Grunske, and Alan Colman. An automated approach to forecasting QoS attributes based on linear and non-linear time series modeling. In *Proc. of the 27th IEEE/ACM Int. Conf. on Automated Software Engineering*, pages 130–139. IEEE, 2012.
- [BJ76] George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. HoldenDay, San Francisco, 1976.
- [CGK<sup>+</sup>11] Radu Calinescu, Lars Grunske, Marta Z. Kwiatkowska, Raffaella Mirandola, and Giordano Tamburrelli. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Trans. Software Eng.*, 37(3):387–409, 2011.
- [Eng82] R.F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica*, pages 987–1007, 1982.
- [GCH<sup>+</sup>04] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.
- [MP11] Raffaella Mirandola and Pasqualina Potena. A QoS-based framework for the adaptation of service-based systems. *Scalable Computing: Practice and Experience*, 12(1), 2011.

# Reliability Analysis in Symbolic Pathfinder: A brief summary\*

Antonio Filieri<sup>a</sup>, Corina S. Păsăreanu<sup>b</sup>, and Willem Visser<sup>c</sup>

<sup>a</sup>Institute of Software Technology, University of Stuttgart, Stuttgart, Germany

<sup>b</sup>Carnegie Mellon Silicon Valley, NASA Ames, Moffet Field, CA, USA

<sup>c</sup>Stellenbosch University, Stellenbosch, South Africa

**Abstract:** Designing a software for critical applications requires a precise assessment of reliability. Most of the reliability analysis techniques perform at the architecture level, driving the design since its early stages, but are not directly applicable to source code. We propose a general methodology based on symbolic execution of source code for extracting failure and success paths to be used for probabilistic reliability assessment against relevant usage scenarios. Under the assumption of finite and countable input domains, we provide an efficient implementation based on Symbolic PathFinder that supports the analysis of sequential and parallel Java programs, even with structured data types, at the desired level of confidence. We validated our approach on both NASA prototypes and other test cases showing a promising applicability scope.

Design and implementation of software systems for critical applications is stressing the need for methodologies and tools to assess and certify its reliability. Different definitions of reliability are introduced within different domains. In this paper we generically refer to reliability as the probability of the software to successfully accomplish its assigned task when requested (Che80). In reality most of the software we use daily is defective in some way, though it can most of the time do its job. Indeed, the presence of a defect in the code may never be realized if the input does not activate the fault (ALRL04). For this reason, the reliability of a software heavily depends on the actual *usage profile* the software is required to deal with.

Most of the approaches for software reliability assessment have been based on the analysis of formal models derived from architectural abstractions (GPMT01; IN08). Model-driven techniques have often been used to keep design models synchronized with the implementation (IN08) and with analysis models. To deal with code, black-box (Mus93) or some ad-hoc reverse engineering approaches have been proposed, e.g. (GPHP05).

In (FPV13), we proposed the systematic and fully automated use of symbolic execution (Kin76; APV07) to extract logical models of failure and successful execution paths directly from source code. Each execution path is fully characterized by a *path condition*, i.e. a set of constraints on the inputs that, if satisfied by the input values, make the execution follow the specific path through the code. In our approach, we label the (terminating) execution paths as either *success* or *failure*. The set of path conditions produced by symbolic

---

\*This paper reports a summary of (FPV13). Please refer to the original paper for a complete exposition.

execution is a complete partition of the input domain (Kin76). Hence, given a probability distribution on the input values, the reliability of the software can be formalized as the probability of satisfying any of the successful path conditions. We take the probability distribution over the input domain as the formalization of the usage profile. Furthermore, we assume the inputs to account for all the external interactions of the software, i.e. with the users, external resources, or third-party applications. Non-termination in presence of loops or recursion is handled by bounded symbolic execution (APV07). In this case interrupted execution paths are labeled as *grey*. For an input satisfying a grey path condition we cannot predict success nor failure. Thus, the probability for an input value to satisfy a grey path condition can be used to define a precise confidence measure to assess the impact of the execution bounds and the consequent quality of the reliability prediction.

As for (FPV13), we focused on inputs ranging over finite domains. This restriction allows us to make use of model counting procedures for efficiently computing the probability of execution paths. Our implementation, based on Symbolic PathFinder (APV07), supports linear integer arithmetic, complex data-structures, loops, and also concurrency. For multi-threaded programs the actual reliability depends both on the usage profile and on the scheduling policy. In this case we identify the best and worst schedule for a given usage profile, that respectively lead to the highest and lowest reliability achievable for that usage.

We evaluated our approach on both examples from the Literature and on NASA's On-board Abort Executive, a real-life complex software from the aerospace domain. Both the accuracy and the analysis time revealed a promising applicability scope for our approach.

## References

- [ALRL04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secure Comput.*, 1(1):11—33, 2004.
- [APV07] S. Anand, C. S. Păsăreanu, and W. Visser. JPF-SE: A Symbolic Execution Extension to Java PathFinder. volume 4424 of *LNCS*, pages 134—138. Springer, 2007.
- [Che80] R.C. Cheung. A User-Oriented Software Reliability Model. *IEEE Trans. Soft. Eng.*, SE-6(2):118—125, 1980.
- [FPV13] A. Filieri, C. S. Păsăreanu, and W. Visser. Reliability analysis in symbolic pathfinder. *ICSE*, pages 622—631. IEEE, 2013.
- [GPHP05] K. Goseva-Popstojanova, M. Hamill, and R. Perugupalli. Large empirical case study of architecture-based software reliability. In *ISSRE*, pages 52—61, Nov 2005.
- [GPMT01] K. Goseva-Popstojanova, A.P. Mathur, and K.S. Trivedi. Comparison of architecture-based software reliability models. In *ISSRE*, pages 22—31, 2001.
- [IN08] A. Immonen and E. Niemela. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software and Systems Modeling*, 7:49—65, 2008.
- [Kin76] J. C. King. Symbolic execution and program testing. *Commun. ACM*, 19(7):385—394, Jul 1976.
- [Mus93] J. Musa. Operational Profiles in Software-Reliability Engineering. *IEEE Software*, 10(2):14—32, March 1993.

# Precision Reuse in CPAchecker \*

Dirk Beyer<sup>1</sup>, Stefan Löwe<sup>1</sup>, Evgeny Novikov<sup>2</sup>, Andreas Stahlbauer<sup>1</sup>, and Philipp Wendler<sup>1</sup>

<sup>1</sup> University of Passau, Innstr. 33, 94032 Passau, Germany

<sup>2</sup> ISPRAS, A. Solzhenitsyn St. 25, 109004 Moscow, Russia

**Abstract:** Continuous testing during development is a well-established technique for software-quality assurance. Continuous model checking from revision to revision is not yet established as a standard practice, because the enormous resource consumption makes its application impractical. Model checkers compute a large number of verification facts that are necessary for verifying if a given specification holds. We have identified a category of such intermediate results that are easy to store and efficient to reuse: *abstraction precisions*. The precision of an abstract domain specifies the level of abstraction that the analysis works on. Precisions are thus a precious result of the verification effort and it is a waste of resources to throw them away after each verification run. In particular, precisions are reasonably small and thus easy to store; they are easy to process and have a large impact on resource consumption. We experimentally show the impact of precision reuse on industrial verification problems created from 62 Linux kernel device drivers with 1 119 revisions.

## Overview

Verification tools spend much effort on computing intermediate results that are needed to check if the specification holds. In most uses of model checking, these intermediate results are erased after the verification process — wasting precious information (in failing and succeeding runs). There are several directions to reuse (intermediate) results [BW13]. *Conditional model checking* [BHKW12] outputs partial verification results for later re-verification of the same program by other verification approaches. *Regression verification* [HJMS03, SG08, HKM<sup>+</sup>96] outputs intermediate results (or checks differences) for re-verification of a changed program by the same verification approach.

In program analysis, e.g., predicate analysis, shape analysis, or interval analysis, the respective abstract domain defines the kind of abstraction that is used to automatically construct the abstract model. The *precision* for an abstract domain defines the level of abstraction in the abstract model, for example, which predicates to track in predicate analysis [BHT08], or which pointers to track in shape analysis [BHT06]. Such precisions can be obtained automatically; interpolation is an example for a technique that extracts precisions for predicate analysis from infeasible error paths.

We propose to reuse precisions as intermediate verification results. Precisions are costly to compute and represent precious intermediate verification results. We treat these abstraction precisions as reusable verification facts, because precisions are easy to extract from model checkers that automatically construct an abstract model of the program (e.g., CEGAR), have a small memory footprint, are tool-independent, and are easy to use for regression verification because they are rather insensitive to changes in the program source code (compared to previous approaches).

---

\*This is a summary of a full article on this topic that appeared in Proc. ESEC/FSE 2013 [BLN<sup>+</sup>13].



The technical insight of our work is that reusing precisions drastically reduces the number of refinements. The effort spent on analyzing spurious counterexamples and re-exploring the abstract state space in search for a suitable abstract model is significantly reduced. We implemented precision reuse in the open-source verification framework CPACHECKER<sup>1</sup> [BK11] (a supplementary web page is also available<sup>2</sup>) and confirmed the effectiveness and efficiency (significant impact in terms of performance gains and increased number of solvable verification tasks) of our approach with an extensive experimental study on industrial code. The benchmark verification tasks were extracted from the Linux kernel, which is an important application domain [BP12], and prepared for verification using the LDV toolkit [MMN<sup>+</sup>12]. Our study consisted of a total of 16 772 verification runs for 4 193 verification tasks that are available online<sup>3</sup>, composed from a total of 1 119 revisions (spanning more than 5 years) of 62 Linux drivers from the Linux-kernel repository. Precision reuse is applicable to all verification approaches that are based on abstraction and automatically computing the precision of the abstract model (including CEGAR). Both the efficiency and effectiveness of such approaches can be increased by reusing precisions. As a result of our experiments, a previously unknown bug in the Linux kernel was discovered by the LDV team, and a fix was submitted to and accepted by the maintainers<sup>4</sup>.

## References

- [BHKW12] D. Beyer, T. A. Henzinger, M. E. Keremoglu, P. Wendler. Conditional Model Checking: A Technique to Pass Information between Verifiers. In *Proc. FSE*. ACM, 2012.
- [BHT06] D. Beyer, T. A. Henzinger, and G. Théoduloz. Lazy Shape Analysis. In *Proc. CAV*, LNCS 4144, pages 532–546. Springer, 2006.
- [BHT08] D. Beyer, T. A. Henzinger, and G. Théoduloz. Program Analysis with Dynamic Precision Adjustment. In *Proc. ASE*, pages 29–38. IEEE, 2008.
- [BK11] D. Beyer and M. E. Keremoglu. CPACHECKER: A Tool for Configurable Software Verification. In *Proc. CAV*, LNCS 6806, pages 184–190. Springer, 2011.
- [BLN<sup>+</sup>13] D. Beyer, S. Löwe, E. Novikov, A. Stahlbauer, and P. Wendler. Precision reuse for efficient regression verification. In *Proc. ESEC/FSE*, pages 389–399. ACM, 2013.
- [BP12] D. Beyer and A. K. Petrenko. Linux Driver Verification. In *Proc. ISoLA*, LNCS 7610, pages 1–6. Springer, 2012.
- [BW13] D. Beyer and P. Wendler. Reuse of Verification Results - Conditional Model Checking, Precision Reuse, and Verification Witnesses. In *Proc. SPIN*, pages 1–17, 2013.
- [HJMS03] T. A. Henzinger, R. Jhala, R. Majumdar, and M. A. A. Sanvido. Extreme model checking. In *Proc. Verification: Theory and Practice*, pages 332–358. Springer, 2003.
- [HKM<sup>+</sup>96] R. H. Hardin, R. P. Kurshan, K. L. McMillan, J. A. Reeds, and N. J. A. Sloane. Efficient Regression Verification. In *Proc. WODES*, pages 147–150, 1996.
- [MMN<sup>+</sup>12] M. U. Mandrykin, V. S. Mutilin, E. M. Novikov, A. V. Khoroshilov, and P. E. Shved. Using Linux device drivers for static verification tools benchmarking. *Programming and Computer Software*, 38(5):245–256, 2012.
- [SG08] O. Strichman and B. Godlin. Regression Verification — A Practical Way to Verify Programs. In *Proc. Verified Software: Theories, Tools, Experiments*, pages 496–501. Springer, 2008.

---

<sup>1</sup><http://cpachecker.sosy-lab.org>

<sup>2</sup><http://www.sosy-lab.org/~dbeyer/cpa-reuse/>

<sup>3</sup><http://www.sosy-lab.org/~dbeyer/cpa-reuse/regression-benchmarks/>

<sup>4</sup><https://patchwork.kernel.org/patch/2204681/>

# Detecting Deadlock in Programs with Data-Centric Synchronization

Christian Hammer

CISPA, Saarland University  
Campus E1.1  
66123 Saarbrücken  
hammer@cs.uni-saarland.de

## Extended Abstract

Writing concurrent programs that operate on shared memory is error-prone as it requires reasoning about the possible interleavings of threads that access shared locations. If programmers make mistakes, two kinds of software faults may occur. *Data races* and *atomicity violations* may arise when shared locations are not consistently protected by locks. *Deadlock* may occur as the result of undisciplined lock acquisition, preventing an application from making progress. Previously [VTD06, VTD<sup>+</sup>10, DHM<sup>+</sup>12], we proposed a data-centric approach to synchronization to raise the level of abstraction in concurrent object-oriented programming and prevent concurrency-related errors.

With data-centric synchronization, fields of classes are grouped into *atomic sets*. Each atomic set has associated *units of work*, code fragments that preserve the consistency of their atomic sets. Our compiler inserts synchronization that is sufficient to guarantee that, for each atomic set, the associated units of work are serializable [HDVT08], thus preventing data races and atomicity violations *by construction*. Our previous work reported on the implementation of atomic sets as an extension of Java called **AJ**: we demonstrated that atomic sets enjoy low annotation overhead and that realistic Java programs can be refactored into **AJ** without significant loss of performance [DHM<sup>+</sup>12].

However, our previous work did not address the problem of deadlock, which may arise in **AJ** when two threads attempt to execute the units of work associated with different atomic sets in different orders. This talk presents a static analysis for detecting possible deadlock in **AJ** programs. The analysis is a variation on existing deadlock-prevention strategies [Mas93, EA03] that impose a global order on locks and check that all locks are acquired in accordance with that order. However, we benefit from the declarative nature of data-centric synchronization in **AJ** to infer the locks that threads may acquire. We rely on two properties of **AJ**: (i) all locks are associated with atomic sets, and (ii) the memory locations associated with different atomic sets will be disjoint unless they are explicitly merged by the programmer. Our algorithm computes a partial order on atomic sets. If such an order can be found, a program is deadlock-free. For programs that use recursive data

structures, the approach is extended to take into account a programmer-specified ordering between different instances of an atomic set.

We implemented this analysis and evaluated it on 10 **AJ** programs. These programs were converted from Java as part of our previous work [DHM<sup>+</sup>12], and cover a range of programming styles. The analysis was able to prove all 10 programs deadlock-free. Minor refactorings were needed in 2 cases, and a total of 4 ordering annotations were needed, all in 1 program.

In summary, this talk presents the following contributions of our latest work [MHD<sup>+</sup>13]:

- We present a static analysis for detecting possible deadlock in **AJ** programs. It leverages the declarative nature of atomic sets to check that locks are acquired in a consistent order. If so, the program is guaranteed to be deadlock-free. Otherwise, possible deadlock is reported.
- To handle recursive data structures, we extend **AJ** with ordering annotations that are enforced by a small extension of **AJ**'s type system. We show how these annotations are integrated with our analysis in a straightforward manner.
- We implemented the analysis and evaluated it on a set of **AJ** programs. The analysis found all programs to be deadlock-free, requiring minor refactorings in two cases. Only 4 ordering annotations were needed, in 1 program.

## References

- [DHM<sup>+</sup>12] Julian Dolby, Christian Hammer, Daniel Marino, Frank Tip, Mandana Vaziri, and Jan Vitek. A Data-centric Approach to Synchronization. *ACM TOPLAS* 34(1):4, 2012.
- [EA03] Dawson R. Engler and Ken Ashcraft. RacerX: effective, static detection of race conditions and deadlocks. In *SOSP*, pages 237–252, 2003.
- [HDVT08] Christian Hammer, Julian Dolby, Mandana Vaziri, and Frank Tip. Dynamic detection of atomic-set-serializability violations. In *ICSE*, pages 231–240, 2008.
- [Mas93] Stephen P. Masticola. *Static Detection of Deadlocks in Polynomial Time*. PhD thesis, Rutgers University, 1993.
- [MHD<sup>+</sup>13] Daniel Marino, Christian Hammer, Julian Dolby, Mandana Vaziri, Frank Tip, and Jan Vitek. Detecting Deadlock in Programs with Data-Centric Synchronization. In *In ICSE*, pages 322–311, May 2013.
- [VTD06] Mandana Vaziri, Frank Tip, and Julian Dolby. Associating synchronization constraints with data in an object-oriented language. In *POPL*, pages 334–345, 2006.
- [VTD<sup>+</sup>10] Mandana Vaziri, Frank Tip, Julian Dolby, Christian Hammer, and Jan Vitek. A Type System for Data-Centric Synchronization. In *ECOOP*, pages 304–328, 2010.

# Efficient State Merging in Symbolic Execution

## (Extended Abstract)

Volodymyr Kuznetsov<sup>1</sup> Johannes Kinder<sup>1,2</sup> Stefan Bucur<sup>1</sup> George Candea<sup>1</sup>

<sup>1</sup>École Polytechnique Fédérale de Lausanne (EPFL)  
{vova.kuznetsov,stefan.bucur,george.candea}@epfl.ch

<sup>2</sup>Royal Holloway, University of London  
johannes.kinder@rhul.ac.uk

Recent tools [CDE08, GLM08, CKC11] have applied symbolic execution to automated test case generation and bug finding with impressive results—they demonstrate that symbolic execution brings unique practical advantages. First, such tools perform dynamic analysis and actually execute a target program, including any external calls; this broadens their applicability to many real-world programs. Second, like static analysis, these tools can simultaneously reason about multiple program behaviors. Third, symbolic execution is fully precise, so it generally does not have false positives.

While recent advances in SMT solving have made symbolic execution tools significantly faster, they still struggle to achieve scalability due to path explosion: the number of possible paths in a program is generally exponential in its size. *States* in symbolic execution encode the history of branch decisions (the *path condition*) and precisely characterize the value of each variable in terms of input values (the *symbolic store*), so path explosion becomes synonymous with state explosion. Alas, the benefit of not having false positives in bug finding comes at the cost of having to analyze an exponential number of states.

**State merging.** One way to reduce the number of states is to merge states that correspond to different paths. Consider, for example, the program *if* ( $x < 0$ ) { $x=0$ ; } *else* { $x=5$ ; } with input  $X$  assigned to  $x$ . We denote with  $(pc, s)$  a state that is reachable for inputs obeying path condition  $pc$  and in which the symbolic store  $s = [v_0 = e_0, \dots, v_n = e_n]$  maps variable  $v_i$  to expression  $e_i$ , respectively. In this case, the two states  $(X < 0, [x = 0])$  and  $(X \geq 0, [x = 5])$ , which correspond to the two feasible paths, can be merged into one state  $(\text{true}, [x = \text{ite}(X < 0, 0, 5)])$ . Here,  $\text{ite}(c, p, q)$  denotes the if-then-else operator that evaluates to  $p$  if  $c$  is true, and to  $q$  otherwise.

State merging effectively decreases the number of paths that have to be explored [God07, HSS09], but also increases the size of the symbolic expressions describing variables. Merging introduces disjunctions, which are notoriously difficult for SMT solvers. Merging also converts differing concrete values into symbolic expressions, as in the example above: the value of  $x$  was concrete in the two separate states, but symbolic ( $\text{ite}(X < 0, 0, 5)$ ) in the merged state. If  $x$  were to appear in branch conditions or array indices later in the execution, the choice of merging the states may lead to more solver invocations than without merging. This combination of larger symbolic expressions and extra solver invocations

can drown out the benefit of having fewer states to analyze, leading to an actual *decrease* in the overall performance of symbolic execution [HSS09].

Furthermore, state merging conflicts with important optimizations in symbolic execution: search-based symbolic execution engines, like the ones used in test case generators and bug finding tools, employ *search strategies* to prioritize searching of “interesting” paths over “less interesting” ones, e.g., with respect to maximizing line coverage given a fixed time budget. To maximize the opportunities for state merging, however, the engine would have to traverse the control flow graph in topological order, which typically contradicts the strategy’s path prioritization policy.

**Our solution.** In this work (published as [KKBC12]), we describe a solution to these two challenges that yields a net benefit in practice. We combine the state space reduction benefits of merged exploration with the constraint solving benefits of individual exploration, while mitigating the ensuing drawbacks. Our main contributions are the introduction of query count estimation and dynamic state merging. *Query count estimation* is a way to statically approximate the number of times each variable will appear in future solver queries after a potential merge point. We then selectively merge two states only when we expect differing variables to appear infrequently in later solver queries. Since this selective merging merely groups paths instead of pruning them, inaccuracies in the estimation do not hurt soundness or completeness. *Dynamic state merging* is a merging algorithm specifically designed to interact favorably with search strategies. The algorithm explores paths independently of each other and uses a similarity metric to identify on-the-fly opportunities for merging, while preserving the search strategy’s privilege of dictating exploration priorities.

Experiments on all 96 GNU COREUTILS show that employing our approach in a symbolic execution engine achieves speedups over the state of the art that are exponential in the size of symbolic input, and can cover up to 11 orders of magnitude more paths. Our code and experimental data are publicly available at <http://cloud9.epfl.ch>.

## References

- [CDE08] C. Cadar, D. Dunbar, and D. Engler. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In *Proc. 8th USENIX Symp. Oper. Syst. Design and Implem. (OSDI 2008)*, pages 209–224. USENIX, 2008.
- [CKC11] V. Chipounov, V. Kuznetsov, and G. Candea. S2E: A platform for in-vivo multi-path analysis of software systems. In *Proc. 16th. Int. Conf. Architectural Support for Prog. Lang. and Oper. Syst. (ASPLOS 2011)*, pages 265–278. ACM, 2011.
- [GLM08] P. Godefroid, M. Levin, and D. Molnar. Automated Whitebox Fuzz Testing. In *Proc. Network and Distributed Syst. Security Symp. (NDSS 2008)*. The Internet Society, 2008.
- [God07] P. Godefroid. Compositional Dynamic Test Generation. In *34th ACM SIGPLAN-SIGACT Symp. Principles of Prog. Lang. (POPL 2007)*, pages 47–54. ACM, 2007.
- [HSS09] T. Hansen, P. Schachte, and H. Søndergaard. State Joining and Splitting for the Symbolic Execution of Binaries. In *9th Int. Workshop Runtime Verification (RV 2009)*, volume 5779 of *LNCS*, pages 76–92. Springer, 2009.
- [KKBC12] V. Kuznetsov, J. Kinder, S. Bucur, and G. Candea. Efficient state merging in symbolic execution. In *Proc. ACM SIGPLAN Conf. Prog. Lang. Design and Implem. (PLDI 2012)*, pages 193–204. ACM, 2012.

# How Do Professional Developers Comprehend Software?

Tobias Roehm  
Lst. Angew. Softwaretechnik  
TU München  
roehm@in.tum.de

Rainer Koschke  
Arbeitsgruppe Softwaretechnik  
Universität Bremen  
koschke@informatik.uni-bremen.de

Rebecca Tiarks  
Arbeitsbereich Mobile Services  
Universität Hamburg  
tiarks@informatik.uni-hamburg.de

Walid Maalej  
Arbeitsbereich Mobile Services  
Universität Hamburg  
maalej@informatik.uni-hamburg.de

**Abstract:** Das Gebiet des Programmverstehens wurde in den letzten zwei Jahrzehnten ausgiebig erforscht. Allerdings ist wenig darüber bekannt, wie Entwickler in der Praxis unter Zeit- und Projekt-Druck Programmverstehen praktizieren und welche von Wissenschaftlern entwickelte Methoden und Werkzeuge sie dabei einsetzen. In diesem Beitrag präsentieren wir die Ergebnisse einer Beobachtungsstudie mit 28 professionellen Software-Entwicklern aus sieben Unternehmen. Wir untersuchen, wie diese Entwickler Programmverstehen praktizieren, und fokussieren uns dabei auf die verwendeten Verstehensstrategien, die benötigten Informationen sowie die benutzten Werkzeuge. Unsere Ergebnisse zeigen, dass sich Entwickler beim Programmverstehen durch Inspektion von Benutzerschnittstellen in die Rolle von Nutzern hineinversetzen. Entwickler versuchen wo möglich Programmverstehen zu vermeiden und verwenden wiederkehrende, strukturierte Verstehens-Strategien, die vom aktuellen Kontext abhängen. Standards und Erfahrung erleichtern die Aufgabe des Programmverstehens. Programmverstehen wird von Entwicklern als Teil von anderen Wartungsaufgaben verstanden und nicht als eigenständige Aufgabe betrachtet. Entwickler bevorzugen direkte, persönliche Kommunikation gegenüber Dokumentation. Insgesamt offenbaren unsere Ergebnisse eine Lücke zwischen Forschung und Praxis im Gebiet des Programmverstehen, da wir keine Nutzung von modernen Programmverstehenswerkzeugen beobachtet haben und Entwickler diese nicht zu kennen scheinen. Unsere Ergebnisse decken die Notwendigkeit für eine weitere, sorgfältige Analyse sowie eine Anpassung von Forschungsschwerpunkten auf.

## Literatur

- [RTKM12] Tobias Roehm, Rebecca Tiarks, Rainer Koschke und Walid Maalej. How Do Professional Developers Comprehend Software? In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, Seiten 255–265, Piscataway, NJ, USA, 2012. IEEE Press.



# On the Appropriate Rationale for Using Design Patterns and Pattern Documentation

Zoya Durdik, Ralf H. Reussner

Institute for Program Structures and Data Organization  
Karlsruhe Institute of Technology (KIT)  
76131 Karlsruhe, Germany  
zoya.durdik@kit.edu  
ralf.reussner@kit.edu

**Abstract:** Software design patterns are proven solutions for recurring design problems. Therefore, one could expect that decisions to use patterns are beneficial and well documented in practice. However, our survey showed that 90% of the software engineers have encountered problems while applying patterns, understanding applied patterns or with their documentation. We address these problems in our paper “On the Appropriate Rationale for Using Design Patterns and Pattern Documentation” published at the “Quality of Software Architecture 2013 (QoSA)” conference. There we present an approach based on a new type of pattern catalogue enriched with question annotations, and the results of a survey with 21 software engineers as a validation of our idea and of exemplary entries of the pattern catalogue.

## 1 Short summary

Software design patterns are proven and widely used solutions for recurring design problems. However, there are several problems connected to the application of patterns, the modification of applied patterns and the documentation of decisions on pattern application. This is also confirmed by 90% of the academic and industrial software engineers who participated in our survey.

Some of the reasons why the use of design patterns is problematic and decisions on their usage are not well documented are: An overly intuitive application of design patterns, the lack of a standard to document design decisions on pattern application, and a burden of documentation effort, in particular when designs are informal and unstable in an early phase of software design.

In our paper “On the Appropriate Rationale for Using Design Patterns and Pattern Documentation” published at the “Quality of Software Architecture 2013 (QoSA)” conference [DR13] we analyse these problems and present an approach to address them. The approach supports the decisions on the appropriate use of design patterns, and the documentation of such decisions together with their rationale.

The approach we propose is based on a pattern catalogue. The major difference to other patterns catalogues is the inclusion of generic question annotations to each pattern to eval-



uate and to document decisions on the use of a pattern. However, the pattern catalogue is not intended to be used as an expert system. Instead, when answering the general questions to a pattern, software engineers learn whether the use of a pattern is appropriate for the specific design problem they are working on. They semi-automatically generate rationale, which is then saved to explain the engineer's decision to apply or to discard a pattern. As answers to the questions stem from requirements, the relevant requirements are linked to a decision. Further more, if a question cannot be answered with existing requirements, the requirements elicitation can be driven by architectural design in pinpointing to needed requirements to justify architectural decisions.

The envisioned benefits of the approach are a more appropriate use of design patterns and pattern variants even by less experienced software engineers, and documented design decision on the use of patterns with semi-automated documentation of their rationale with positive effects on evolution. In addition, trace links from requirements to design decision rationales and from there to the concerned architectural elements are automatically generated and documented, which further helps in system evolution.

Furthermore, in [DR13] we present the results of a survey with 21 software engineers as a validation of the idea and of some entries of the proposed pattern catalogue. The results of the survey can be summarized as follows: About 90% of the survey participants have encountered problems while applying patterns, understanding the applied patterns or their documentation. About 90% of the participants estimated that the proposed approach can be helpful to solve one or several of the encountered problems. In particular, 71% were positive that the pattern catalogue could help clarifying properties and consequences of a pattern, and 52% were positive about it to solve documentation problems, if answers to the pattern questions are automatically co-documented. The provided question annotations were considered understandable in about 95% in average for the listed sample patterns. This means that even if the pattern questions are general and project-independent, they can be answered by engineers in their project-specific situations.

The opinions of the participants may be subjective, however, the results of the survey provide a positive and valuable indication on the potential usefulness of such a catalogue. More details on the proposed approach and on the survey are provided in the [DR13].

## Acknowledgements

This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution.

## References

- [DR13] Zoya Durdik and Ralf Reussner. On the Appropriate Rationale for Using Design Patterns and Pattern Documentation. In *Proceedings of the 9th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA 2013)*, pages 107–116, June 2013.

# Specification Patterns from Research to Industry: A Case Study in Service-Based Applications

[Extended Abstract]

Domenico Bianculli<sup>1</sup>, Carlo Ghezzi<sup>2</sup>, Cesare Pautasso<sup>3</sup>, and Patrick Senti<sup>4</sup>

<sup>1</sup>SnT Centre - University of Luxembourg, Luxembourg, Luxembourg

`domenico.bianculli@uni.lu`

<sup>2</sup>DEEPSE group - DEIB - Politecnico di Milano, Milano, Italy

`carlo.ghezzi@polimi.it`

<sup>3</sup>Faculty of Informatics - University of Lugano, Lugano, Switzerland

`cesare.pautasso@usi.ch`

<sup>4</sup>formerly with Credit Suisse AG, Zürich, Switzerland

`patrick.senti@gmail.com`

Specification patterns [DAC98] have been proposed as a means to express recurring properties in a generalized form, allowing developers to state system requirements precisely and map them to specification languages like temporal logics. The majority of past work has focused on the use of specification patterns in the context of concurrent and real-time systems, and has been limited to a research setting. In this presentation we report the results of our study [BGPS12] on the use of specification patterns in the context of service-based applications (SBAs); the study focused on industrial SBAs in the banking domain. The study collected and classified the requirements specifications of two sets of case studies. One set consisted of 104 cases extracted from research articles in the area of specification, verification and validation of SBAs published between 2002 and 2010. The other set included 100 service specifications developed by our industrial partner for its service-oriented information system over a similar time period. During the study, each requirement specification was matched against a specification pattern; in total, we analyzed and classified 290 + 625 requirements specifications from research and industrial data, respectively. The requirements specifications were classified according to four classes of property specification patterns. Three of them correspond to the systems of specification patterns proposed by Dwyer et al. [DAC98], by Konrad and Cheng [KC05], and by Gruhn and Laue [GL06]; these patterns have been widely used for the specification and verification of concurrent and real-time systems. The fourth group includes patterns that are specific to service provisioning and have emerged during the study; they are:

**Average response time (S1)** is a variant of the bounded response pattern [KC05] that uses the average operator to aggregate the response time over a certain time window.

**Counting the number of events (S2)** is used to express common non-functional requirements such as reliability (e.g., “number of errors in a given time window”) and throughput (e.g., “number of requests that a client is allowed to submit in a given time window”).

**Average number of events (S3)** is a variant of the previous pattern that states the average number of events occurred in a certain time interval within a certain time window, as in “the average number of client requests per hour computed over the daily business hours”.

**Maximum number of events (S4)** is a variant of pattern S3 that aggregates events using the maximum operator.

**Absolute time (S5)** indicates events that should occur at a time that satisfies an absolute time constraint, as in “if the booking is done in May, a discount is given”.

**Unbounded elapsed time (S6)** indicates the time elapsed since the last occurrence of a certain event.

**Data-awareness (S7)** is a pattern denoting properties that refer to the actual data content of messages exchanged between services as in “every ID present in a message cannot appear in any future message”.

The study showed that: a) the majority of requirements specifications stated in industrial settings referred to specific aspects of service provisioning, which led to the definition of the new class of specification patterns; b) the specification patterns proposed in the research literature [DAC98, KC05, GL06] were barely used in industrial settings.

Furthermore, the new class of specification patterns led to the definition of a new specification language able to express them; the language, introduced in [BGS13], is called SOLOIST (*SpecificatiOn Language fOr service compoSitions inTeractions*) and is a many-sorted first-order metric temporal logic with new temporal modalities that support aggregate operations on events occurring in a given time window.

**Acknowledgements.** This work has been partially supported by the European Community under the IDEAS-ERC grant agreement no. 227977-SMScom and by the National Research Fund, Luxembourg (FNR/P10/03).

## References

- [BGPS12] Domenico Bianculli, Carlo Ghezzi, Cesare Pautasso, and Patrick Senti. Specification Patterns from Research to Industry: a Case Study in Service-based Applications. In *Proc. of ICSE 2012*, pages 968–976. IEEE Computer Society, 2012.
- [BGS13] Domenico Bianculli, Carlo Ghezzi, and Pierluigi San Pietro. The Tale of SOLOIST: a Specification Language for Service Compositions Interactions. In *Proc. of FACS’12*, volume 7684 of *LNCS*, pages 55–72. Springer, 2013.
- [DAC98] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Property specification patterns for finite-state verification. In *Proc. of FMSP ’98*, pages 7–15. ACM, 1998.
- [GL06] Volker Gruhn and Ralf Laue. Patterns for Timed Property Specifications. *Electron. Notes Theor. Comput. Sci.*, 153(2):117–133, 2006.
- [KC05] Sascha Konrad and Betty H. C. Cheng. Real-time specification patterns. In *Proc. of ICSE ’05*, pages 372–381. ACM, 2005.

# Software Architecture Documentation for Developers: A Survey

Dominik Rost<sup>1</sup>, Matthias Naab<sup>1</sup>, Crescencio Lima<sup>2</sup>, Christina von Flach Chavez<sup>2</sup>

<sup>1</sup>Fraunhofer Institute for Experimental Software Engineering  
Kaiserslautern, Germany  
{dominik.rost, matthias.naab}@iese.fraunhofer.de

<sup>2</sup>Fraunhofer Project Center on Software and Systems Engineering  
Software Engineering Laboratory, Department of Computer Science  
Federal University of Bahia, Bahia, Brazil  
{crescencio, flach@dcc.ufba.br}

Software architecture has become an established discipline in industry and documentation is the key for its efficient and effective usage. However, many companies do not have any architecture documentation in place or, if they do, the available documentation is not perceived as adequate by developers to support them in their tasks. To complement our experiences from projects with industry customers and as a foundation for the improvement of methods and tools in architecture documentation, we conducted a survey among 147 industrial participants, investigating their current problems and wishes for the future. Participants from different countries in Europe, Asia, North and South America shared their experiences. This paper presents the results of the survey.

We identified five main findings. The results confirm the common belief that architecture documentation is most frequently outdated, updated only with strong delays, and inconsistent in detail and form and backed it up with data. Further, developers perceive difficulties with a “one-size-fits-all” architecture documentation, which does not adequately provide information for their specific tasks and contexts. Developers seek more interactive ways of working with architecture documentation that allow them to find needed information more easily with extended navigation and search possibilities. And finally, what developers perceive as relevant in terms of architecture information gives, in our opinion, a very complete and mature picture of what software architecture and its documentation should consist of.

Based on these results, we discuss directions for further research and the development of advanced methods and tools for architecture documentation in this paper. Centralization with powerful tooling and automated generation mechanisms are possible means for addressing the problems of outdated and unspecific architecture documentation. Additionally, a clear, easy-to-follow connection between architecture and code is a major concern of developers for which new techniques have to be created. To achieve increased uniformity, organizations should invest in internal standardization in terms of form, details, and terminology, as well as in establishing a single source of architecture

documentation to avoid inefficiency due to scattered information. We understand that static architecture documents must be replaced by new and interactive ways to convey the information that allow easy searching and navigation. And finally, to increase readability and understandability, we see an additional need for standardization and clarity through reduction of information.

# On the (Compatible) Evolution of Services<sup>\*</sup>

Vasilios Andrikopoulos

Institute of Architecture of Application Systems (IAAS)  
University of Stuttgart  
Universitätsstraße 38  
70569 Stuttgart  
{firstname.lastname}@iaas.uni-stuttgart.de

**Abstract:** In an environment of constant change and variation driven by competition and innovation, a software service can rarely remain stable. Being able to manage and control the evolution of services is therefore an important goal for the Service-Oriented paradigm. This work extends existing and widely-adopted theories from software engineering, programming languages, service oriented computing and other related fields to provide the fundamental ingredients required to guarantee that spurious results and inconsistencies that may occur due to uncontrolled service changes are avoided. The presented work provides a unifying theoretical framework for controlling the evolution of services that deals with structural, behavioral and QoS level-induced service changes in a type-safe manner. The goal of the work is to ensure correct version transitions so that previous and future clients can use a service in a consistent manner.

The evolution of software due to changing requirements, technological shifts and corrective actions has been a well documented challenge for software engineering (and in particular for the field of Software Configuration Management) in the last decades [Leh96, ELvdH<sup>+</sup>05]. With regards to distributed and by extension service-oriented systems, however, a number of additional to traditional software system engineering challenges rise [BR00]. More specifically, large service networks consist of a number of services that potentially may belong to more than one organizations, making the identification and scoping of what constitutes the evolving system (so that maintenance activities can take place) non-trivial. More importantly, service-orientation is by definition based on a model of distributed ownership of services enabled by the loose coupling design principle, in the sense that services may be fully or partially composed out of third-party services that lay beyond the control of the service provider. In this context, the application of well-established techniques like refactoring or impact analysis becomes problematic at the very least.

Towards addressing these challenges, as part of our work in the context of EU's Network of Excellence S-Cube<sup>1</sup>, and culminating in [And10] and [ABP12], we have proposed a theoretical framework to manage the evolution of service interfaces in a type safe manner. The goal of this work is to assist service designers and developers in ensuring that changes to their services do not affect service consumers in a disruptive manner. For this purpose

---

<sup>\*</sup>This work was partially funded by the FP7 EU-FET project 600792 Allow Ensembles.

<sup>1</sup>S-Cube: <http://www.s-cube-network.eu>.

we have adapted, reused and integrated some well established methods and techniques from both software engineering and computer science.

In particular with respect to [ABP12], the presented work starts with establishing a framework on which different compatibility definitions are positioned and connected with each other, distinguishing between two dimensions: horizontal/vertical (ie. interoperability vs. substitutability) and backward/forward (ie. provider- and consumer-oriented). A formal definition of compatibility is provided incorporating both identified dimensions based on type theory. For this purpose the meta-model for services first presented in [ABP08] is leveraged to facilitate type-based reasoning on service changes. The meta-model consists of elements and their relationships in three layers: structural (message-related), behavioral (w.r.t. the observable behavior of the services) and non-functional (QoS-related). The subtyping relation  $\tau \leq \tau'$  is defined for the elements and relationships in each layer with semantics that depend on their types.

Building on these tools, the concept of *T-shaped changes* is introduced as the set of change operations (add, delete, modify)  $\Delta S$  that when applied to a service  $S$  results in a fully compatible service description  $S'$ . Checking for (full) compatibility follows directly from definition and is realized as a short algorithm. Evaluation of the work focuses on two aspects: firstly showing how the proposed approach supersedes the established best practices for (Web) services evolution, and secondly demonstrating its efficacy and efficiency by means of a proof-of-concept realization, through which additional challenges were identified mainly w.r.t. the implementation technologies commonly used for Web services. Addressing a set of these challenges is the subject of ongoing work in the context of the Allow Ensembles EU project.

## References

- [ABP08] Vasilios Andrikopoulos, Salima Benbernou, and Mike P. Papazoglou. Managing the Evolution of Service Specifications. In *CAiSE'08*, pages 359–374. Springer-Verlag, 2008.
- [ABP12] Vasilios Andrikopoulos, Salima Benbernou, and Michael P. Papazoglou. On the Evolution of Services. *IEEE Transactions on Software Engineering*, 38:609–628, 2012.
- [And10] Vasilios Andrikopoulos. *A Theory and Model for the Evolution of Software Services*. Number 262 in CentER Dissertation Series. Tilburg University Press, 2010.
- [BR00] Keith H. Bennett and Vclav T. Rajlich. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 73–87, Limerick, Ireland, 2000. ACM.
- [ELvdH<sup>+</sup>05] Jacky Estublier, David Leblang, Andr van der Hoek, Reidar Conradi, Geoffrey Clemm, Walter Tichy, and Darcy Wiborg-Weber. Impact of software engineering research on the practice of software configuration management. *ACM Trans. Softw. Eng. Methodol.*, 14(4):383–430, 2005.
- [Leh96] M. M. Lehman. Laws of Software Evolution Revisited. In *Proceedings of the 5th European Workshop on Software Process Technology*, pages 108–124. Springer-Verlag, 1996.

# Generierung konsistenzerhaltender Editierskripte im Kontext der Modellversionierung

Timo Kehrer

Praktische Informatik, Universität Siegen  
kehrer@informatik.uni-siegen.de

**Abstract:** Modellbasierte Softwareentwicklung erfordert spezialisierte Werkzeuge für ein professionelles Versions- und Variantenmanagement von Modellen. Insbesondere Anwendungsfälle wie das Patchen oder Mischen von Modellen stellen sehr hohe Anforderungen an die Konsistenz der synthetisierten Modelle. Ein Lösungsansatz ist die Verwendung konsistenzerhaltender Editierskripte. Zentrale Herausforderung ist letzten Endes die Generierung solcher Editierskripte, welche wir in diesem Papier mit entsprechenden Hinweisen auf weiterführende Literatur kurz skizzieren.

## 1 Motivation

Modellbasierte Softwareentwicklung hat sich in einigen Applikationsdomänen inzwischen fest etabliert. Modelle sind hier primäre Artefakte, entwickeln sich daher ständig weiter und existieren im Laufe ihrer Evolution in zahlreichen Versionen und Varianten. In der Praxis zeigt sich sehr deutlich, dass man für Modelle die gleichen Versionsmanagement-Dienste benötigt, die man für textuelle Dokumente gewohnt ist, namentlich Werkzeugfunktionen zum Vergleichen, Patchen und Mischen von Modellen.

Derzeitig verfügbare Werkzeuge des Versions- und Variantenmanagements von Modellen arbeiten jedoch auf systemnahen, fallweise werkzeugspezifischen Repräsentationen von Modellen und unterstellen ferner generische Graphoperationen zur Beschreibung von Änderungen. Dies führt zu zwei wesentlichen Problemen:

1. Die Darstellung solcher “low-level” Änderungen ist meist unverständlich, ohne Kenntnisse der internen Repräsentation der Modelle teilweise sogar unmöglich.
2. Die Anwendung von low-level Änderungen in Patch- oder Mischszenarien birgt die Gefahr der Synthetisierung inkonsistenter Modelle, da i.d.R. nicht alle Änderungen auf das Zielmodell propagiert werden können. Im schlimmsten Fall kann ein Modell so inkorrekt werden, dass es nicht mehr mit Standard-Modelleditoren verarbeitet werden kann.



## 2 Modellversionierung auf Basis von Editieroperationen

Lösungsansätze für die vorstehend skizzierten Probleme werden im Forschungsprojekt MOCA<sup>1</sup> erarbeitet. Ziel des Projekts ist es, alle für das Versions- und Variantenmanagement relevanten Werkzeugfunktionen auf die Abstraktionsebene von Editieroperationen anzuheben, welche für einen Benutzer *verständlich* und bei der Anwendung auf ein Modell *konsistenzerhaltend* sind.

**Konsistenzerhaltende Editieroperationen.** Unter einer konsistenzerhaltenden Editieroperation verstehen wir eine in-place Transformationsvorschrift, welche ein konsistentes Modell in einen konsistenten Folgezustand überführt, unabhängig von den aktuellen Aufrufparametern der Operation. Ein Modell bezeichnen wir als konsistent, wenn es konform zum effektiven Metamodell der unterstellten Editierumgebung ist. Die Menge der zulässigen Editieroperationen hängt somit vom Modelltyp und letzten Endes auch von der gegebenen Editierumgebung ab.

**Editierskripte.** Bei der Propagation von Änderungen in Patch- oder Mischszenarien sollen Änderungsvorschriften benutzt werden, welche ausschließlich aus Aufrufen von Editieroperationen bestehen. Solch eine halbgeordnete Menge von Operationsaufrufen bezeichnen wir als *Editierskript* (engl.: *Edit Script*). Zentrale Grundlage für die Realisierung entsprechender Patch- und Mischwerkzeuge ist also die Generierung von Editierskripten.

**Generierung von Editierskripten.** Editierskripte sind in der Regel durch Vergleich zweier Versionen eines Modells möglichst effizient zu berechnen. Ausgangspunkt unseres Algorithmus ist eine gegebene low-level Differenz, welche mit existierenden Verfahren des Modellvergleichs erzeugt werden kann. Diese wird nachfolgend in mehreren Schritten zu einem Editierskript weiter verarbeitet: Zunächst werden Gruppen von low-level Änderungen identifiziert, wobei eine Gruppe (engl.: *Semantic Change Set*) den Aufruf einer Editieroperation repräsentiert [KKT11]. Anschließend werden die Argumente der identifizierten Operationsaufrufe extrahiert und die sequentiellen Abhängigkeiten der Operationsaufrufe analysiert [KKT13]. Der Algorithmus arbeitet generisch in dem Sinne, als dass er durch die Eingabe der Spezifikationen der zulässigen Editieroperationen konfiguriert und somit an beliebige Modelltypen und Editorumgebungen angepasst werden kann.

## Literatur

- [KKT11] T. Kehrer, U. Kelter und G. Taentzer. A Rule-Based Approach to the Semantic Lifting of Model Differences in the Context of Model Versioning. In *Proc. 26th IEEE/ACM Intl. Conf. on Automated Software Engineering*. IEEE Computer Society, 2011.
- [KKT13] T. Kehrer, U. Kelter und G. Taentzer. Consistency-Preserving Edit Scripts in Model Versioning. In *Proc. 28th IEEE/ACM Intl. Conf. on Automated Software Engineering*. IEEE, 2013.

---

<sup>1</sup><http://www.dfg-spp1593.de/index.php?id=46>

# Ein formal fundierter Entscheidungs-Ansatz zur Behandlung von Technical Debt

Klaus Schmid

Institut für Informatik  
Universität Hildesheim  
Marienburger Platz 22  
31141 Hildesheim  
schmid@sse.uni-hildesheim.de

**Zusammenfassung:** Der Begriff *Technical Debt* bezeichnet Entwicklungslasten, die in der Weiterentwicklung von Software zu Problemen führen können. Diese Problematik ist von großer praktischer Bedeutung, vor allem in inkrementellen Entwicklungsprozessen, und gewinnt aktuell auch in der wissenschaftlichen Community stark an Bedeutung. Hier betrachten wir die Frage welche Bedingungen eine optimale Strategie zur Behandlung von *Technical Debt* erfüllen muss, ob eine solche Strategie überhaupt möglich ist und wie realistische Strategien aussehen können. Dabei ergibt sich, dass eine optimale Strategie auf fundamentale Probleme stößt. Wir führen daher Approximationen ein, die eine optimale Auswahl ermöglichen. Daraus leiten wir einen Ansatz zur Auswahl von *Technical Debt* items ab. Dieser erlaubt die effiziente Behandlung von *Technical Debt* in beliebigen, insbesondere iterativen und gar agilen Entwicklungsprozessen.

## 1 Technical Debt

Mit dem Begriff *Technical Debt* werden Kostentreiber in der Softwareentwicklung bezeichnet, die auf frühere Entwicklungsentscheidungen zurückgehen. Oft haben diese Entscheidungen zuerst zu einer Vereinfachung und Beschleunigung der Entwicklung geführt, jedoch führen sie in der weiteren Evolution zu Zusatzaufwänden. Cunningham führte für solche kurzfristig beschleunigten, aber langfristig behindernden Entscheidungen den Begriff *Technical Debt* (technische Schulden) ein [Cun92]. Beispiele dafür können sein: die Nutzung eines Frameworks, das den Anforderungen im Verlauf der Weiterentwicklung nicht gewachsen ist, oder die Verwendung eines vereinfachten Designs, welches im Rahmen der Weiterentwicklung nicht mehr hinreichend skaliert. *Technical Debt* lässt sich zurückzahlen, indem die entsprechenden Entscheidungen revidiert werden, bspw. durch Ersetzen des Frameworks oder durch Refactoring des Designs.

Das Erkennen und Behandeln von *Technical Debt* ist ein Thema von hoher praktischer Relevanz: sowohl in allen inkrementellen Entwicklungsprozessen (wie bspw. agilen Ansätzen) als auch allgemein in der Evolution von Software. Seit kurzer Zeit erhält das Thema auch intensive Aufmerksamkeit in der Forschungsgemeinde (Veröffentlichungen, Sonderhefte, Workshops).

## 2 Ein Ansatz zur Entscheidungsfindung

Es gibt bereits eine Vielzahl von Arbeiten, die sich mit dem Thema *Technical Debt* beschäftigen. Oft werden dabei ad-hoc Kriterien und Ansätze verwendet [Sch13a]. In dem in [Sch13b, Sch13c] dargestellten Ansatz wurde daher der Fokus auf eine systematische Ableitung der Kriterien und des Entscheidungsverfahrens gelegt. Der Ansatz führt eine detaillierte und formale Definition von Technical Debt ein, die sich an den Kostenunterschieden zwischen Einsparungen durch das Eingehen von Technical Debt und den verursachten Kosten im weiteren Entwicklungsverlauf orientiert. Dabei werden die Kosten als Erwartungswert zukünftiger, möglicher Entwicklungspfade definiert. Auf dieser Basis ist eine präzise Definition möglich, was auch die Ableitung eines optimalen Entwicklungspfades erlauben würde. Eine praxisorientierte Umsetzung der Theorie stößt jedoch auf fundamentale Schwierigkeiten, da bspw. alle möglichen alternativen Architekturen bzgl. aller möglichen zukünftigen Entwicklungspfade bewertet werden müssten. Dies ist für realistische Systeme prinzipiell unmöglich. Man kann entsprechend davon ausgehen, dass die Bestimmung eines optimalen, praktisch anwendbaren Ansatzes zur Behandlung von Technical Debt fundamental unmöglich ist.

Ausgehend von der Unvereinbarkeit eines präzisen Ansatzes zur Technical-Debt-Behandlung und den Bedürfnissen eines praxisnahen Ansatzes werden verschiedene Approximationen eingeführt: der Vergleich mit einem optimalen Referenzsystem wird durch den Vergleich mit einem fixen Referenzsystem ersetzt, die Annahme der Unabhängigkeit der einzelnen Entwicklungsschritte wird eingeführt und die Betrachtung einer beliebig großen Menge von Entwicklungssequenzen wird durch die Betrachtung einer endlichen (beliebig kleinen) Repräsentantenmenge ersetzt. Prinzipiell lassen sich diese Approximationen beliebig kombinieren und nur einzelne davon verwenden. Wir betrachten jedoch beispielhaft den Ansatz, der entsteht wenn alle Approximationen gleichzeitig angewendet werden. Der resultierende Ansatz erweist sich als einfach anwendbar und in grundlegender Weise mit dem ursprünglichen Ansatz verwandt. Insbesondere besitzt der Ansatz eine systematische Grundlage mit definierten, einzelnen Approximationen, die explizit identifiziert und formalisiert sind. Damit hat der Ansatz fundamentale Vorteile gegenüber anderen publizierten Ansätzen. Weiterhin kann er einfach in inkrementelle, insbesondere agile Ansätze integriert werden.

## Literaturverzeichnis

- [Cun92] W. Cunningham. *The WyCash portfolio management systems*. In Object-oriented programming systems, languages, and applications (OOPSLA'92, Addend.), Seite 29–30, 1992. [www.c2.com/doc/oopsla92.html](http://www.c2.com/doc/oopsla92.html).
- [Sch13a] K. Schmid. *On the Limits of the Technical Debt Metaphor: Some Guidance on Going Beyond*, Proceedings of the 4<sup>th</sup> International Workshop on Technical Debt (MTD) at ICSE, Seite 63-66, 2013. DOI: 10.1109/MTD.2013.6608681
- [Sch13b] K. Schmid. *A Formal Approach to Technical Debt Decision Making*, Proceedings of the 9th international ACM Sigsoft conference on Quality Of Software Architectures (QoSA'13), Seite 153-162, ACM, 2013. DOI: 10.1145/2465478.2465492.
- [Sch13c] K. Schmid. *Technical Debt — From Metaphor to Engineering Guidance: A Novel Approach based on Cost Estimation*. Technical Report, University of Hildesheim, No. 1/13, 2013.

# Tool Support for Integrated Development of Component-based Embedded Systems

Gerd Kainz<sup>1</sup>, Christian Buckl<sup>1</sup>, and Alois Knoll<sup>2</sup>

<sup>1</sup> Cyber-Physical Systems  
fortiss  
Guerickestraße 25  
80805 München  
{kainz, buckl}@fortiss.org

<sup>2</sup> Robotics and Embedded Systems  
Technische Universität München  
Boltzmannstraße 3  
85748 Garching bei München  
knoll@in.tum.de

**Abstract:** Nowadays many applications are based on embedded systems and more and more tasks are implemented in software. This trend increases the need of embedded systems and raises their complexity. To deal with this situation we present a model-driven development approach supporting all developers (platform, component, and application) equally during the creation of component-based embedded systems.

## 1 Approach

The development of embedded systems becomes more challenging while more functions are implemented in software. In addition, formerly unrelated applications have to communicate with each other to form so called cyber-physical systems (CPSes).

Embedded systems are comprised beside application code of a high amount of platform code which consists code related to operating systems, components (including drivers), and glue code. The implementation of those parts requires the involvement of various experts.

We are going to present an overview of our previous work focusing on the design of a model-driven development approach. In contrast to existing approaches we do not only support the application developers but also try to support all other developers (component and application developers) equally. By our approach the tasks of the developers are clearly separated and the approach takes care of working dependencies between them.

Figure 1 shows an overview of the approach consisting of the three parts: modeling, model-to-model (M2M) transformations, and model-to-text (M2T) transformation (code generation). Those are described in the following.

The modeling concept is based on a multi-phase development process. In this process each developer (platform, component, and application) can concentrate on his tasks and the system takes care of a correct integration of the different parts. Thereby, the developers build upon the preceding developments and are guided by the approach. The approach uses a modular design principle based on instantiation and configuration. [KBSK10, KBK11]

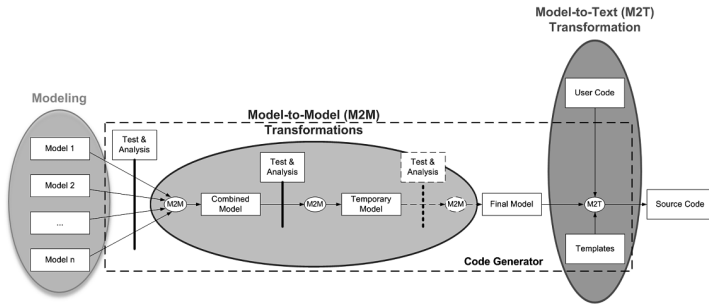


Abbildung 1: Main parts of the presented model-driven development approach for component-based embedded systems

For the transformations from input models to models used for code generation the approach provides a mechanism supporting exogenous M2M transformation chains. This allows breaking down the whole transformation into many small and modular transformations which are easier to understand and maintain. For the creation of the M2M transformation chain the developers only need to specify the structural differences between the input and output metamodels on the metamodel level and the changes to the data of the input model(s) on the model level. The remaining parts of the metamodels/models are handled by a provided tool, which creates the respective output metamodels and models and thereby, reduces any additional effort. [KBK12]

The M2T transformation (code generation) is also based on the multi-phase modeling approach and allows the definition of the various system parts independent of each other. To integrate the various system parts the developer can call functions providing the missing parts. During code generation these function calls are then resolved to the right system parts depending on the provided models. As result of the code generation the integrated system is available.

## Literatur

- [KBK11] Gerd Kainz, Christian Buckl und Alois Knoll. Automated Model-to-Metamodel Transformations Based on the Concepts of Deep Instantiation. In Jon Whittle, Tony Clark und Thomas Kühne, Hrsg., *Model Driven Engineering Languages and Systems*, Jgg. 6981 of *Lecture Notes in Computer Science*, Seiten 17–31. Springer Berlin / Heidelberg, 2011.
- [KBK12] Gerd Kainz, Christian Buckl und Alois Knoll. A Generic Approach Simplifying Model-to-Model Transformation Chains. In RobertB. France, Jürgen Kazmeier, Ruth Breu und Colin Atkinson, Hrsg., *Model Driven Engineering Languages and Systems*, Jgg. 7590 of *Lecture Notes in Computer Science*, Seiten 579–594. Springer Berlin Heidelberg, 2012.
- [KBSK10] Gerd Kainz, Christian Buckl, Stephan Sommer und Alois Knoll. Model-to-Metamodel Transformation for the Development of Component-Based Systems. In DorinaC. Petriu, Nicolas Rouquette und ystein Haugen, Hrsg., *Model Driven Engineering Languages and Systems*, Jgg. 6395 of *Lecture Notes in Computer Science*, Seiten 391–405. Springer Berlin / Heidelberg, 2010.

# Synthesis of Component and Connector Models from Crosscutting Structural Views (extended abstract)

Shahar Maoz  
School of Computer Science  
Tel Aviv University, Israel

Jan Oliver Ringert, Bernhard Rumpe  
Software Engineering  
RWTH Aachen University, Germany

**Abstract:** This extended abstract reports on [MRR13]. We presented component and connector (C&C) views, which specify structural properties of component and connector models in an expressive and intuitive way. C&C views provide means to abstract away direct hierarchy, direct connectivity, port names and types, and thus can crosscut the traditional boundaries of the implementation-oriented hierarchical decomposition of systems and sub-systems, and reflect the partial knowledge available to different stakeholders involved in a system's design.

As a primary application for C&C views we investigated the synthesis problem: given a C&C views specification, consisting of mandatory, alternative, and negative views, construct a concrete satisfying C&C model, if one exists. We showed that the problem is NP-hard and solved it, in a bounded scope, using a reduction to SAT, via Alloy. We further extended the basic problem with support for library components, specification patterns, and architectural styles. The result of synthesis can be used for further exploration, simulation, and refinement of the C&C model or, as the complete, final model itself, for direct code generation.

A prototype tool and an evaluation over four example systems with multiple specifications show promising results and suggest interesting future research directions towards a comprehensive development environment for the structure of component and connector designs.

Component and connector (C&C) models are used in many application domains, from cyber-physical and embedded systems to web services to enterprise applications. The structure of a C&C model consists of components at different containment levels, their typed input and output ports, and the connectors between them.

A system's C&C model is typically complex; it is not designed by a single engineer and is not completely described in a single document. Thus, we considered a setup where many different, incomplete, relatively small fragments of the model are provided by architects responsible for subsystems, for the implementation of specific features, use cases, or functionality, which crosscut the boundaries of components. Moreover, teams may have several, alternative solutions that address the same concern, and some knowledge about designs that must not be used. To move forward in the development process and enable implementation, these partial models and the intentions behind them should be integrated and then realized into a single, complete design. However, such an integration is a complex and challenging task.

In [MRR13] we presented *component and connector views*, which specify structural properties of component and connector models in an expressive and intuitive way. C&C views

provide means to abstract away direct hierarchy, direct connectivity, port names and types. Specifically, C&C views may not contain all components and connectors (typically a small subset related only to a specific use case or set of functions or features). They may contain (abstract) connectors between components at different, non-consecutive containment levels, and they may provide incomplete typing information, that is, components' ports may be un-typed. While the standard structural abstraction and specification mechanisms for C&C models rely on the traditional, implementation-oriented hierarchical decomposition of systems to sub-systems, C&C views allow one to specify properties that crosscut the boundaries of sub-systems. This makes them especially suitable to reflect the partial knowledge available to different stakeholders involved in a system's design.

As a primary application for C&C views we investigated the synthesis problem: given mandatory, alternative, and negative views, construct a concrete satisfying C&C model, if one exists. We have shown that the synthesis problem for C&C views specifications is NP-hard and solved it, in a bounded scope, using a reduction to Alloy [Jac06]. The input for the synthesis is a C&C views specification. Its output is a single C&C model that satisfies the specification and is complete, to allow implementation. When no solution exists (within a bounded scope), the technique reports that the input specification is unsatisfiable.

As a concrete language for C&C models we used MontiArc [HRR12], a textual ADL developed using MontiCore [KRV10], with support for direct Java code generation (including interfaces, factories, etc.). The C&C views are defined as an extension to general C&C models. The concrete syntax used in our implementation is an extension of MontiArc.

To further increase the usefulness of C&C views synthesis in practice, we extended the basic synthesis problem with support for three advanced features. First, support for integration with pre-defined or library components. Second, support for several high-level specification patterns. Third, support for synthesis subject to several architectural styles.

We implemented C&C views synthesis and evaluated it by applying it to four example systems. The implementation and example specifications are available from [www].

## References

- [HRR12] Arne Haber, Jan Oliver Ringert, and Bernhard Rumpe. MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems. Technical Report AIB-2012-03, RWTH Aachen, february 2012.
- [Jac06] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.
- [KRV10] Holger Krahn, Bernhard Rumpe, and Steven Völkel. MontiCore: a framework for compositional development of domain specific languages. *STTT*, 12(5):353–372, 2010.
- [MRR13] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Synthesis of Component and Connector Models from Crosscutting Structural Views. In Bertrand Meyer, Luciano Baresi, and Mira Mezini, editors, *ESEC/SIGSOFT FSE*, pages 444–454. ACM, 2013.
- [www] C&C Views Synthesis Evaluation Materials. <http://www.se-rwth.de/materials/cncvviews/>.

# Modular Reasoning for Crosscutting Concerns with Contracts

Thomas Thüm

University of Magdeburg  
thomas.thuem@ovgu.de

**Abstract:** Separation of concerns into modules is an active research area since four decades. Modularization is beneficial for complex software systems, as it enables a divide-and-conquer strategy to software development and maintenance. A key ingredient for modularization is that modules can be studied to a certain extent in isolation, which is important for program comprehension as well as for verification. Design by contract is a means to formalize implicit assumptions for module boundaries and thus facilitates modular reasoning. While design by contract was initially proposed for object-oriented programming, we focus on the modularization of crosscutting concerns. We discuss several approaches to combine design by contract with modularization techniques for crosscutting concerns. While some of these approaches have been discussed previously, we unify them to achieve synergies. Our experience with case studies suggests that we can achieve fine-grained trade-offs between openness to extensions by other modules and closeness for modular reasoning. We argue that our approach generalizes the open-closed principle known from object-oriented programming to crosscutting concerns.

In this talk, we give an overview on our experiences in specifying crosscutting concerns with modular contracts. For further reading and a list of all involved co-authors, we refer to previously published articles [TSKA11, STAL11, TSAH12, Thü12, TAZ<sup>+</sup>13, Thü13, SST13, AvRTK13].

## References

- [AvRTK13] Sven Apel, Alexander von Rhein, Thomas Thüm, and Christian Kästner. Feature-Interaction Detection based on Feature-Based Specifications. *Computer Networks*, 57(12):2399–2409, August 2013.
- [SST13] Reimar Schröter, Norbert Siegmund, and Thomas Thüm. Towards Modular Analysis of Multi Product Lines. In *Proc. Int’l Workshop Multi Product Line Engineering (MultiPLE)*, pages 96–99, New York, NY, USA, August 2013. ACM.
- [STAL11] Wolfgang Scholz, Thomas Thüm, Sven Apel, and Christian Lengauer. Automatic Detection of Feature Interactions using the Java Modeling Language: An Experience Report. In *Proc. Int’l Workshop Feature-Oriented Software Development (FOSD)*, pages 7:1–7:8, New York, NY, USA, August 2011. ACM.



- [TAZ<sup>+</sup>13] Thomas Thüm, Sven Apel, Andreas Zelend, Reimar Schröter, and Bernhard Möller. Subclack: Feature-Oriented Programming with Behavioral Feature Interfaces. In *Proc. Workshop Mechanisms for Specialization, Generalization and Inheritance (MASPEGHI)*, pages 1–8, New York, NY, USA, July 2013. ACM.
- [Thü12] Thomas Thüm. Verification of Software Product Lines Using Contracts. In *Doktorandentagung Magdeburger-Informatik-Tage (MIT)*, pages 75–82, Germany, July 2012. University of Magdeburg.
- [Thü13] Thomas Thüm. Product-Line Verification with Feature-Oriented Contracts. In *Proc. Int’l Symposium in Software Testing and Analysis (ISSTA)*, pages 374–377, New York, NY, USA, July 2013. ACM.
- [TSAH12] Thomas Thüm, Ina Schaefer, Sven Apel, and Martin Hentschel. Family-Based Deductive Verification of Software Product Lines. In *Proc. Int’l Conf. Generative Programming and Component Engineering (GPCE)*, pages 11–20, New York, NY, USA, September 2012. ACM.
- [TSKA11] Thomas Thüm, Ina Schaefer, Martin Kuhlemann, and Sven Apel. Proof Composition for Deductive Verification of Software Product Lines. In *Proc. Int’l Workshop Variability-intensive Systems Testing, Validation and Verification (VAST)*, pages 270–277, Washington, DC, USA, March 2011. IEEE.

# Programs from Proofs – Approach and Applications\*

Daniel Wonisch, Alexander Schremmer, Heike Wehrheim

Department of Computer Science  
University of Paderborn  
Germany  
wehrheim@upb.de

**Abstract:** Proof-carrying code approaches aim at the safe execution of untrusted code by having the code producer attach a safety proof to the code which the code consumer only has to validate. Depending on the type of safety property, proofs can however become quite large and their validation - though faster than their construction - still time consuming.

Programs from Proofs is a new concept for the safe execution of untrusted code. It keeps the idea of putting the time consuming part of proving on the side of the code producer, however, attaches no proofs to code anymore but instead uses the proof to *transform* the program into an *equivalent* but *more efficiently verifiable* program. Code consumers thus still do proving themselves, however, on a computationally inexpensive level only.

In case that the initial proving effort does not yield a conclusive result (e.g., due to a timeout), the very same technique of program transformation can be used to obtain a zero overhead runtime monitoring technique.

## 1 Overview

Proof-carrying code (PCC) as introduced by Necula [Nec97] is a technique for the safe execution of untrusted code. The general idea is that once a correctness proof has been carried out for a piece of code the proof is attached to the code, and code consumers successively only have to check the correctness of the proof. The technique is *tamperproof*, i.e., if the proof is not valid for the program and property at hand, the code consumer will actually detect it.

Within the Collaborative Research Center SFB 901 at the University of Paderborn we have developed an alternative concept for the safe execution of untrusted code called *Programs from Proofs* (PfP) [WSW13a]. Like PCC it is first of all a general concept with lots of possible instantiations, one of which we have already developed. Our concept keeps the general idea behind PCC: the potentially untrusted code producer gets the major burden in the task of ensuring safety while the consumer has to execute a time and space efficient procedure only. The approach works as follows. The code producer carries out a

---

\*This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre “On-The-Fly Computing” (SFB 901).

proof of correctness of the program with respect to a safety property. In our current instance of the PfP framework, the safety property is given as a protocol automaton and the correctness proof is constructed by the software verification tool CPACHECKER [BK11] using a predicate analysis. The information gathered in the proof (in our scenario an abstract reachability tree) is next used to *transform* the program into an *equivalent*, more efficiently verifiable (but usually larger wrt. lines of code) program. The transformed program is delivered to the consumer who – prior to execution – is also proving correctness of the program, however, with a significantly reduced effort. The approach remains tamper-proof since the consumer is actually verifying correctness of the delivered program. Experimental results show that the proof effort can be reduced by several orders of magnitude, both with respect to time and space.

Besides using it as a proof-simplifying method, PfP can also be used as a *runtime monitoring* technique [WSW13b]. This might become necessary when (fully-automatic) verification only yields inconclusive results, e.g., because of timeouts or insufficient memory. In this case, PfP will use the inconclusive proof, i.e., the inconclusive abstract reachability tree (ART), for the program transformation. An inconclusive ART still contains error states since the verification did not succeed in proving (or refuting) the property. When transforming this ART into a program, we insert HALT statements at these potential points of failure so that the program stops before running into an error state. The obtained program is thus safe by construction, and equivalent to the old program on its non-halting paths. Thus the Program From Proofs method in this application scenario gives us a zero-overhead runtime monitoring technique which needs no additional monitoring code (except for HALTs).

## References

- [BK11] Dirk Beyer and M. Erkan Keremoglu. CPACHECKER: A Tool for Configurable Software Verification. In G. Gopalakrishnan and S. Qadeer, editors, *CAV 2011*, volume 6806 of *LNCS*, pages 184–190. Springer-Verlag, Berlin, 2011.
- [Nec97] George C. Necula. Proof-carrying code. In *POPL 1997*, pages 106–119, New York, NY, USA, 1997. ACM.
- [WSW13a] Daniel Wonisch, Alexander Schremmer, and Heike Wehrheim. Programs from Proofs - A PCC Alternative. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 912–927. Springer, 2013.
- [WSW13b] Daniel Wonisch, Alexander Schremmer, and Heike Wehrheim. Zero Overhead Runtime Monitoring. In Robert M. Hierons, Mercedes G. Merayo, and Mario Bravetti, editors, *SEFM*, volume 8137 of *Lecture Notes in Computer Science*, pages 244–258. Springer, 2013.

# Software-Produktqualität modellieren und bewerten: Der Quamoco-Ansatz

Stefan Wagner

Institut für Softwaretechnologie  
Universität Stuttgart  
stefan.wagner@informatik.uni-stuttgart.de

**Abstract:** Existierende Software-Qualitätsmodelle bieten entweder abstrakte Qualitätscharakteristiken oder konkrete Messungen; eine Integration der beiden Aspekte fehlt. Im Projekt Quamoco haben wir einen umfassenden Ansatz entwickelt, um diese Lücke zu schließen.

Wir entwickelten eine Struktur für operationalisierbare Qualitätsmodelle, eine Bewertungsmethode und darauf aufbauend ein Basismodell, das wichtige Qualitätsfaktoren abdeckt. Zusätzliche, spezifische Modelle beschreiben Faktoren für spezielle Domänen. Eine empirische Untersuchung der Modelle und Methoden zeigt Potential für eine praktische Anwendung und weitere Forschung.

## 1 Einleitung

Trotz der Vielfalt von Software-Qualitätsmodelle verbleiben sie entweder abstrakt oder konzentrieren sich nur auf konkrete Messungen. Ein integrierter Ansatz von abstrakten Qualitätsfaktoren zu konkreten Messungen fehlt. Ansätze zur Qualitätsbewertung sind ebenfalls sehr spezifisch oder verbleiben abstrakt. Die Gründe dafür liegen in der Komplexität von Qualität und der Unterschiedlichkeit von Qualitätsprofilen. Operationalisierte Qualitätsmodelle sind deshalb schwer zu bauen. Im Projekt Quamoco haben wir einen umfassenden Ansatz entwickelt, um diese Lücke zu schließen. Die Details zum Quamoco-Ansatz sind in [WLH<sup>+</sup>12] zu finden.

## 2 Quamoco-Qualitätsmodelle

Im Projekt haben wir sowohl neue Konzepte zu Qualitätsmodellen und -bewertung erarbeitet als sie auch in Modelle umgesetzt. Wir entwickelten ein Meta-Qualitätsmodell, das die Struktur operationalisierter Qualitätsmodelle festlegt. Das darin enthaltene Konzept des Produktfaktors überbrückt die Lücke zwischen Messungen und Qualitätscharakteristiken. Darüberhinaus bietet das Metamodell die Möglichkeit Module für unterschiedliche Domänen zu entwickeln. Basierend auf dem Metamodell entwickelten wir einen konkreten Bewertungsansatz und entsprechende Werkzeugunterstützung.

Um den Aufwand und die Komplexität für das Bauen der Qualitätsmodelle für bestimmte Domänen zu reduzieren haben wir ein Basisqualitätsmodell entwickelt, das generelle, für viele Domänen relevante, Faktoren abdeckt. Es verwendet die Qualitätscharakteristiken der ISO/IEC 25010, die wir mit 300 Produktfaktoren und 500 Maßen für Java und C# verfeinert haben.

Im Metamodell und der Modularisierung sind wir davon ausgegangen, dass wir Qualitätsmodelle spezifisch für den Kontext bauen müssen, gerade dann, wenn sie operationalisiert sein sollen. Deshalb haben wir für einige Domänen das Basismodell um spezifische Faktoren erweitert. Es entstanden Qualitätsmodelle für SOA [GL11], Standardsoftware, Individualsoftware, Integrationsprojekte, Eingebettete Systeme [MPK<sup>+</sup>12] und Informationssysteme.

Schließlich führten wir empirische Validierungen des Ansatzes auf Open-Source- und Industrie-Systemen durch. Die Qualitätsbewertungsergebnisse stimmen gut mit Einschätzungen von Experten überein und Praktiker halten das Modell und die Bewertungsergebnisse für hilfreich, um einen Überblick über die Qualität zu bekommen. Damit können wir eine solide, frei verfügbare, Basis für die zukünftige Erweiterung, Validierung und den Vergleich mit anderen Ansätzen zur Verfügung stellen.

### 3 Zusammenfassung und Ausblick

Der Quamoco-Ansatz liefert einen vollständigen Ansatz zur Qualitätsmodellierung und -bewertung. Durch das Bereitstellen der Modelle und Werkzeuge als Open Source ist er eine ideale Grundlage für weitere Forschungsarbeiten in diesem Gebiet.

**Danksagung** Ich danke allen Mitgliedern des Quamoco-Projekts, insbesondere den Co-Autoren des ICSE-Papiers K. Lochmann, L. Heinemann, M. Kläs, A. Trendowicz, R. Plösch, A. Seidl, A. Göb und J. Streit. Diese Forschung wurde größtenteils gefördert vom BMBF unter dem Kennzeichen 01IS08023.

### Literatur

- [GL11] Andreas Goeb und Klaus Lochmann. A Software Quality Model for SOA. In *Proc. 8th International Workshop on Software Quality*, 2011.
- [MPK<sup>+</sup>12] Alois Mayr, Reinhold Plösch, Michael Kläs, Constanza Lampasona und Matthias Saft. A Comprehensive Code-Based Quality Model for Embedded Systems. In *Proc. 23th IEEE International Symposium on Software Reliability Engineering 2012*, 2012.
- [WLH<sup>+</sup>12] Stefan Wagner, Klaus Lochmann, Lars Heinemann, Michael Kläs, Adam Trendowicz, Reinhold Plösch, Andreas Seidl, Andreas Goeb und Jonathan Streit. The Quamoco Product Quality Modelling and Assessment Approach. In *Proc. 34th International Conference on Software Engineering*, 2012.

# Messung der Strukturellen Komplexität von Feature-Modellen

Richard Pohl, Vanessa Stricker und Klaus Pohl

paluno - The Ruhr Institute for Software Technology  
Universität Duisburg-Essen  
Gerlingstr. 16  
45127 Essen  
richard.pohl@paluno.uni-due.de  
vanessa.stricker@paluno.uni-due.de  
klaus.pohl@paluno.uni-due.de

**Abstract:** Die automatisierte Analyse von Feature-Modellen (FM) basiert größtenteils auf der Lösung der als NP-vollständig bekannten Probleme SAT und CSP. Trotz aktueller Heuristiken, die eine effizient Analyse in den meisten Fällen möglich machen, treten bei der Analyse dennoch Einzelfälle mit hoher Laufzeit auf. Diese Einzelfälle sind bisher nicht auf die Struktur der formalen Modelle zurückführbar und damit unvorhersehbar. Dieser Beitrag schlägt die Anwendung von Graphweitenmaßen aus der Graphentheorie auf die Formalisierung der FM-Analyse vor, um die strukturelle Komplexität der FM-Analyse abzuschätzen. Die Nützlichkeit der Abschätzung wurde in einem Experiment demonstriert. Daher kann sie künftig als Basis für eine einheitliche Methode zur systematischen Verbesserung von FM-Analysewerkzeugen dienen.

## 1 Einleitung

Feature-Modelle (FM) sind eine gängige Art zur Dokumentation von Variabilität in der Software-Produktlinienentwicklung. Die automatisierte Analyse von FM kann zur Gewinnung von Informationen über FM, z.B. im Rahmen der Korrektheitsprüfung, beitragen. FM werden dazu in ein formales Modell transformiert – oftmals konjunktive Normalform (Conjunctive Normal Form, CNF) oder ein Bedingungserfüllungsproblem (Constraint Satisfaction Problem, CSP). Diese formalen Modelle werden durch Werkzeuge (z.B. SAT-Solver) analysiert. Die Laufzeit dieser Verfahren ist von der Größe und der strukturellen Komplexität der formalen Modelle abhängig und in Einzelfällen unvorhersehbar. Während der Zusammenhang zwischen Laufzeit und Modellgröße bekannt ist [PLP11], ist der Zusammenhang mit der strukturellen Komplexität bislang ungeklärt.

Im Gegensatz dazu hat sich die Abschätzung der strukturellen Komplexität von Graphen für einzelne Analyseoperationen in der Graphentheorie mit Hilfe von Graphweitenmaßen (graph width measures, GWM) etabliert. GWM wurden bereits erfolgreich auf CNF und CSP angewandt. Daher stellt sich die Frage nach ihrer Anwendbarkeit und Nützlichkeit zur Abschätzung formaler Modelle der FM-Analyse.

## 2 Beitrag

Der Beitrag [PSP13] untersucht die Frage nach der Anwendbarkeit und Nützlichkeit von GWM auf die FM-Analyse experimentell. Zur Experimentdurchführung wurde das Comparison Framework for Feature Model Analysis Performance (CoFFeMAP)<sup>1</sup> entwickelt. Dieses stellt eine Infrastruktur für die Experimente bereit. Dazu ist zunächst eine Transformation von FM in drei formale Modelle notwendig, die dann von gängigen Werkzeugen analysiert werden können. Zur Experimentdurchführung wurden sowohl drei Mengen mit je 180 Modellen des Generators BeTTy<sup>2</sup> als auch der SPLOT-Benchmark<sup>3</sup> herangezogen.

Auf den drei Formalisierungen wurden mit einer existierenden Bibliothek<sup>4</sup> jeweils vier Varianten der *tree-width*, einem etablierten GWM, berechnet, wodurch sich für jedes formale Modell drei (insgesamt zwölf) strukturelle Komplexitätsmetriken ergaben. Im Experiment wurden die formalen Modelle auf mindestens drei aktuellen Solvern der üblicherweise zur FM-Analyse genutzten Kategorien BDD, SAT und CSP, gelöst.

Für alle berechneten GWM und die gemessenen Lösungszeiten von allen Experimentläufen wurde die Korrelation untersucht. Die Ergebnisse aller Experimente zeigen durch starke und signifikante Korrelationen, dass sich Graphweitenmaße eignen, um die strukturelle Komplexität in Bezug auf die verwendete Formalisierung abzuschätzen. Weiterhin wurde eine Kombination aus Graphweitenmaß und Formalisierung (die untere Abschätzung der *tree-width* des *order encodings*) gefunden, die zur Laufzeit nahezu aller Solvern korreliert und sich somit zur Abschätzung der Komplexität auch ohne Wissen über die eigentlich genutzte Formalisierung eignet.

## Danksagung

Die Autoren bedanken sich bei Sergio Segura und Ana B. Sánchez für die Modelle aus BeTTy. Dieser Beitrag entstand im Rahmen des DFG-Projekts KOPI (PO 607/4-1 KOPI).

## Literatur

- [PLP11] Richard Pohl, Kim Lauenroth und Klaus Pohl. A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In *Proc. 26th IEEE/ACM Int. Conf. on Automated Software Engineering*, Seiten 313–322, Washington, DC, USA, 2011. IEEE Computer Society.
- [PSP13] Richard Pohl, Vanessa Stricker und Klaus Pohl. Measuring the Structural Complexity of Feature Models. In *Proc. 28th IEEE/ACM International Conference on Automated Software Engineering*, Seiten 454–464. IEEE, November 2013.

---

<sup>1</sup><http://www.sse.uni-due.de/de/278>

<sup>2</sup><http://www.isa.us.es/betty/try-new-betty-line-fm-generator>

<sup>3</sup><http://splot-research.org>

<sup>4</sup><http://www.treewidth.com>

# Methods of Model Quality in the Automotive Area

Robert Reicherdt and Sabine Glesner

Software Engineering of Embedded Systems  
Technische Universität Berlin  
Ernst-Reuter-Platz 7  
10587 Berlin  
robert.reicherdt@tu-berlin.de  
sabine.glesner@tu-berlin.de

**Abstract:** MATLAB Simulink is the most widely used industrial tool for developing complex embedded systems in the automotive industry. These models often consist of multiple thousands of blocks and a large number of hierarchy levels. The increasing complexity leads to challenges for quality assurance as well as for maintenance of the models. In this article, we present a novel approach for slicing Simulink Models using dependence graphs and demonstrate its efficiency using case studies from the automotive and avionics domain. With slicing we can reduce the complexity of the models removing unrelated elements, thus paving the way for subsequent static quality assurance methods. Moreover, slicing enables the use of techniques like change impact analysis on the model level.

Model-based development (MBD), especially based on MATLAB/Simulink, is a well established technique in automotive area and widely used in industry. However, despite the increasing significance of MBD, quality assurance and maintenance techniques are quite immature compared to the techniques present for classical software development. In our Project *MeMo - Methods of Model Quality* [HWS<sup>+</sup>11], we have aimed to develop new analysis approaches and techniques to increase the quality of MATLAB/Simulink models. This was joint work together with two industrial partners from the area of model and software quality assurance.

One major result of the MeMo project is our novel approach [RG12] for the slicing of MATLAB/Simulink models. This approach consists of two parts: (1) a dependence analysis for MATLAB/Simulink models and (2) the slicing based on a reachability analysis. The dependence analysis in our approach is based on the sequential simulation semantics of MATLAB/Simulink models. While we can derive data dependences directly from the lines representing signal flow in the model, we derive control dependences from the *conditional execution contexts* of the models. MATLAB/Simulink uses conditional execution contexts for the conditional execution of blocks in the model. However, these contexts cannot be extracted directly, neither from the model file nor via the MATLAB/Simulink API. Hence, we have reimplemented the calculation using safe overapproximations for cases in which the informal and incomplete MATLAB/Simulink documentation does not contain sufficient information. Subsequent to the dependence analysis, we create a dependence graph and calculate the slices using a forward or backward reachability analysis.



We have implemented our approach into a tool which is able to parse a model, to store it in a database and to slice the model. To get the necessary information, we have implemented our parser in two phases. In the first phase we parse the models from the model files to build a skeleton. In the second phase we then use the MATLAB/Simulink API to extract additional run-time information (e.g., port widths and signal dimensions) from the models, which are not available directly from the model file.

In the evaluation on a number of cases studies we were able to show an average reduction of 45% in size of the models using our original approach. By now, we have extended our approach with a more precise analysis of data dependences in bus systems, which led to average slice sizes around 37%. This is another reduction of around 12% compared to our original approach.

In the last decade, only few approaches for the slicing of modeling notations for reactive systems have been published. However, most of these approaches have aimed at state-based notations such as extended finite state machines or state charts. A comprehensive overview for these approaches is given in [ACH<sup>+</sup>10]. To the best of our knowledge, our approach is the first approach published about the slicing of MATLAB/Simulink models.

Besides the ability to reduce the complexity of a model for a specific point of interest (i.e. a block or a signal) this approach offers new possibilities in the development and maintenance of models in MBD. Like in classical software development, in MBD it is important to track changes and especially their impact on the modeled systems. With our slicing technique, we are now able to lift slicing-based *change impact analysis* from the level of classical software development to the model level.

To do so, we have started a new project, the CISM<sub>o</sub> project. Together with an industrial partner, we aim to develop a novel slicing-based change impact analysis for MATLAB/Simulink models. Besides the development of this analysis, the project also aims to enhance this analysis with heuristics and a parametrization to increase scalability and tailor it to the needs of industrial applications. Moreover, we still plan to extend our slicing approach to *Stateflow* which is often used in MATLAB/Simulink models and is a state-based notation. With this extension, we aim to gain even more precision.

## References

- [ACH<sup>+</sup>10] K. Androutsopoulos, D. Clark, M. Harman, J. Krinke, and L. Tratt. Survey of slicing finite state machine models. Technical report, Technical Report RN/10/07, University College London, 2010.
- [HWS<sup>+</sup>11] Wei Hu, Joachim Wegener, Ingo Stürmer, Robert Reicherdt, Elke Salecker, and Sabine Glesner. MeMo - Methods of Model Quality. In *Dagstuhl-Workshop MBEES: Modell-basierte Entwicklung eingebetteter Systeme VII*, pages 127–132, 2011.
- [RG12] Robert Reicherdt and Sabine Glesner. Slicing MATLAB Simulink models. In *34th International Conference on Software Engineering (ICSE)*, pages 551–561, 2012.

# **Zur Integration von Struktur- und Verhaltensmodellierung mit OCL**

Lars Hamann, Martin Gogolla, Oliver Hofrichter  
AG Datenbanksysteme  
Universität Bremen  
Postfach 330440  
28334 Bremen

(lhamann|gogolla|hofrichter)@informatik.uni-bremen.de

Das werkzeuggestützte Validieren von in der Unified Modeling Language (UML) spezifizierten Modellen ist zu einem wichtigen Forschungszweig in der modellgetriebenen Softwareentwicklung geworden. Durch eine frühzeitige Validierung sollen Designfehler bereits zu Beginn des Entwicklungsprozesses aufgedeckt werden, um spätere aufwändige Korrekturen zu vermeiden. Herkömmliche Werkzeuge in diesem Kontext verwenden UML-Klassendiagramme in Verbindung mit textuellen Einschränkungen, die in der Object Constraint Language (OCL) definiert sind. Eher unbeachtet sind weitergehende Modellierungselemente wie zum Beispiel Zustandsautomaten, die zwar von der UML bereitgestellt werden, aber in OCL-basierten Validierungswerkzeugen bisher kaum Einzug gefunden haben.

Der hier zusammengefasste Beitrag [HHG12b] zeigt, wie die in der UML vorhandenen Protokollzustandsautomaten (Protocol State Machines; PSMs) die Modellierungs- und Validierungsmöglichkeiten erweitern. Alle vorgestellten Konzepte sind in einem UML/OCL-Werkzeug [USE, GBR07], das international in der Praxis sowie zu Forschungsprojekten und in der Lehre eingesetzt wird, umgesetzt.

Während Klassendiagramme dazu verwendet werden, um die Struktur eines Systems z. B. mit Hilfe von Klassen und Assoziationen zu beschreiben, können Zustandsautomaten dazu verwendet werden das Verhalten zu beschreiben. PSMs verfolgen dabei einen rein deklarativen Ansatz, indem sie gültige Reihenfolgen von Operationsaufrufen für eine Klasse festlegen. Sie beschreiben also ein oder mehrere Protokolle einer Klasse. Den PSMs gegenüber stehen Behavioral State Machines, die mit Hilfe einer Action Language zustandsverändernde Aktionen in einem Objekt ausführen können.

Unser Beitrag befasst sich mit Protokollzustandsautomaten, da diese unter anderem durch ihre Zustandsfolgen im Gegensatz zu Vor- und Nachbedingungen von Operationen mehr als nur einen Zustandswechsel berücksichtigen. Es wird dargestellt, wie PSMs während des Designprozesses genutzt werden können, um gültige Programmabläufe zu spezifizieren und an welchen Stellen OCL diesen Vorgang unterstützen kann. So können z. B. für Transitionen Vorbedingungen (Guards) in OCL definiert werden. Diese Guards erlauben es, dass ein Aufruf einer Operation zu unterschiedlichen Zuständen führen kann. Die auszuführende Transition kann hierbei anhand des Wahrheitswerts der einzelnen Guards gewählt werden.

Diese und weitere Funktionen der Automaten werden dabei ebenso ausführlich beschrieben, wie das Laufzeitverhalten während der Modellausführung. Das gesamte Vorgehen wird anhand von Beispielen erläutert, die zeigen, wie Designfehler frühzeitig entdeckt werden können. Die Möglichkeiten der Fehleranalyse mit dem vorgestellten UML/OCL-Werkzeug durch unterschiedliche Sichten auf das Modell und auf den aktuellen Systemzustand, wie z. B. Objektdiagramme, Sequenzdiagramme und Zustandsautomaten, die stets synchronisiert sind, werden ebenfalls gezeigt.

Zusätzlich werden besondere Funktionen, die im Kontext der Laufzeitverifikation von Implementierungen benötigt werden diskutiert. So erlaubt z. B. der in [HHG12a] beschriebene Ansatz zur Laufzeitverifikation, dass die Verifikation einer Anwendung zu einem beliebigen Zeitpunkt der Ausführung starten kann. Dies führt dazu, dass die bisher erfolgten Zustandsübergänge nicht beobachtet werden konnten. Dadurch sind die jeweiligen Automateninstanzen nicht mit der Anwendung synchronisiert. Um eine nachträgliche Synchronisation zu ermöglichen, zeigen wir einen Ansatz, der die für einzelne Zustände definierten Invarianten (State Invariants) verwendet; sind diese geeignet definiert, kann auch ohne Wissen über die vorher auf einer Instanz aufgerufenen Operationen der jeweilige Zustand eines Automaten ermittelt werden.

## Literatur

- [GBR07] Martin Gogolla, Fabian Büttner und Mark Richters. USE: A UML-Based Specification Environment for Validating UML and OCL. *Science of Computer Programming*, 69:27–34, 2007.
- [HHG12a] Lars Hamann, Oliver Hofrichter und Martin Gogolla. OCL-Based Runtime Monitoring of Applications with Protocol State Machines. In Antonio Vallecillo, Juha-Pekka Tolvanen, Ekkart Kindler, Harald Störrle und Dimitrios S. Kolovos, Hrsg., *ECMFA*, Jgg. 7349 of *Lecture Notes in Computer Science*, Seiten 384–399. Springer, 2012.
- [HHG12b] Lars Hamann, Oliver Hofrichter und Martin Gogolla. On Integrating Structure and Behavior Modeling with OCL. In Robert B. France, Jürgen Kazmeier, Ruth Breu und Colin Atkinson, Hrsg., *Model Driven Engineering Languages and Systems - 15th International ACM/IEEE Conference, MODELS 2012, Innsbruck, Austria, September 30-October 5, 2012. Proceedings*, Jgg. 7590 of *Lecture Notes in Computer Science*, Seiten 235–251. Springer, 2012.
- [USE] A UML-based Specification Environment. <http://sourceforge.net/projects/useocl/>.

# Software Architecture Optimization Methods: A Systematic Literature Review

Aldeida Aleti\*, Barbora Buhnova<sup>†</sup>, Lars Grunske<sup>‡</sup>,  
Anne Kozirolek<sup>§</sup>, Indika Meedeniya<sup>¶</sup>

aldeida.aleti@monash.edu, buhnova@fi.muni.cz,  
lars.grunske@informatik.uni-stuttgart.de, kozirolek@kit.edu, indikamee@gmail.com

Architecture specifications and models [ISO07] are used to structure complex software systems and to provide a blueprint that is the foundation for later software engineering activities. Thanks to architecture specifications, software engineers are better supported in coping with the increasing complexity of today's software systems. Thus, the architecture design phase is considered one of the most important activities in a software engineering project [BCK03]. The decisions made during architecture design have significant implications for economic and quality goals. Examples of architecture-level decisions include the selection of software and hardware components, their replication, the mapping of software components to available hardware nodes, and the overall system topology.

**Problem Description and Motivation.** Due to the increasing system complexity, software architects have to choose from a combinatorially growing number of design options when searching for an optimal architecture design with respect to a defined (set of) quality attribute(s) and constraints. This results in a design space search that is often beyond human capabilities and makes the architectural design a challenging task [GLB<sup>+</sup>06]. The need for automated design space exploration that improves an existing architecture specification has been recognized [PBKS07] and a plethora of architecture optimization approaches based on formal architecture specifications have been developed. To handle the complexity of the task, the optimization approaches restrict the variability of architectural decisions, optimizing the architecture by modifying one of its specific aspects (allocation, replication, selection of architectural elements etc.). Hence the research activities are scattered across many research communities, system domains (such as embedded systems or information systems), and quality attributes. Similar approaches are proposed in multiple domains without being aware of each other.

**Research Approach and Contribution.** To connect the knowledge and provide a comprehensive overview of the current state of the art, we provided a systematic literature review of the existing architecture optimization approaches [ABG<sup>+</sup>13]. As a result, a gateway

---

\*Monash University, Australia

<sup>†</sup>Masaryk University, Czech Republic

<sup>‡</sup>Universität Stuttgart, Germany

<sup>§</sup>Karlsruher Institut für Technologie, Germany

<sup>¶</sup>The Portland House Group, Australia

to new approaches of architecture optimization can be opened, combining different types of architectural decisions during the optimization or using unconventional optimization techniques. Moreover, new trade-off analysis techniques can be developed by combining results from different optimization domains. All this can bring significant benefits to the general practice of architecture optimization. In general, with the survey we aimed to achieve the following objectives:

- Provide a basic classification framework in form of a taxonomy to classify existing architecture optimization approaches.
- Provide an overview of the current state of the art in the architecture optimization domain.
- Point out current trends, gaps, and directions for future research.

We examined 188 papers from multiple research sub-areas, published in software-engineering journals and conferences. Initially, we derived a taxonomy by performing a formal content analysis. More specifically, based on the initial set of keywords and defined inclusion and exclusion criteria, we collected a set of papers, which we iteratively analyzed to identify the taxonomy concepts. The taxonomy was then used to classify and analyze the papers, which provided a comprehensive overview of the current research in architecture optimization. The data was then used to perform a cross analysis of different concepts in the taxonomy and derive gaps and possible directions for further research.

The full paper has been published in the IEEE Transactions on Software Engineering [ABG<sup>+</sup>13].

## References

- [ABG<sup>+</sup>13] Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziulek, and Indika Meedeniya. Software Architecture Optimization Methods: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 39(5):658–683, 2013.
- [BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. AddisonWesley, second edition, 2003.
- [GLB<sup>+</sup>06] Lars Grunske, Peter A. Lindsay, Egor Bondarev, Yiannis Papadopoulos, and David Parker. An Outline of an Architecture-Based Method for Optimizing Dependability Attributes of Software-Intensive Systems. In Rogério de Lemos, Cristina Gacek, and Alexander B. Romanovsky, editors, *Architecting Dependable Systems*, volume 4615 of *Lecture Notes in Computer Science*, pages 188–209. Springer, 2006.
- [ISO07] International-Standard-Organization. ISO/IEC Standard for Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems. *ISO/IEC 42010 IEEE Std 1471-2000 First edition 2007-07-15*, pages c1 –24, 6 2007.
- [PBKS07] Alexander Pretschner, Manfred Broy, Ingolf H. Krüger, and Thomas Stauner. Software Engineering for Automotive Systems: A Roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 55–71. IEEE Computer Society, 2007.

# Flexible Access Control for JavaScript

Christian Hammer

CISPA, Saarland University  
Campus E1.1  
66123 Saarbrücken  
hammer@cs.uni-saarland.de

## Extended Abstract

Many popular Web applications mix content from different sources, such as articles coming from a newspaper, a search bar provided by a search engine, advertisements served by a commercial partner, and included third-party libraries to enrich the user experience. The behavior of such a web site depends on *all* of its parts working, especially so if it is financed by ads. Yet, not all parts are equally trusted. Typically, the main content provider is held to a higher standard than the embedded third-party elements. A number of well publicized attacks have shown that ads and third-party components can introduce vulnerabilities in the overall application. Taxonomies of these attacks are emerging [JJLS10]. Attacks such as *cross site scripting*, *cookie stealing*, *location hijacking*, *clickjacking*, *history sniffing* and *behavior tracking* are being catalogued, and the field is rich and varied.<sup>1</sup>

This paper proposes a novel security infrastructure for dealing with this threat model. We extend JavaScript objects with *dynamic ownership* annotations and break up a web site's computation at ownership changes, that is to say when code belonging to a different owner is executed, into *delimited histories*. Subcomputations performed on behalf of untrusted code are executed under a special regime in which most operations are recorded into histories. Then, at the next ownership change, or at other well defined points, these histories are made available to user-configurable *security policies* which can, if the history violates some safety rule, issue a *revocation* request. Revocation undoes all the computational effects of the history, reverting the state of the heap to what it was before the computation. Delimiting histories is crucial for our technique to scale to real web sites. While JavaScript pages can generate millions of events, histories are typically short, and fit well within the computation model underlying Web 2.0 applications: once the history of actions of an untrusted code fragment is validated, the history can be discarded. Histories allow policies to reason about the impact of an operation within a scope by giving policies a view on the outcome of a sequence of computational steps. Consider storing a secret into an object's field. This could be safe if the modification was subsequently overwritten and replaced by the field's original value. Traditional access control policies would reject

---

<sup>1</sup>[www.webappsec.org/projects/threat](http://www.webappsec.org/projects/threat), [www.owasp.org/index.php/Category:Attack](http://www.owasp.org/index.php/Category:Attack).

the first write, but policies in our framework can postpone the decision and observe if this is indeed a leak. While policies of interest could stretch all the way to dynamic information flow tracking, we focus on access control in this talk and present the following contributions [RHZN<sup>+</sup>13]:

- *A novel security infrastructure:* Access control decisions for untrusted code are based on delimited histories. Revocation can restore the program to a consistent state. The enforceable security policies are a superset of [Sch00] as revocation allows access decisions based on future events.
- *Support of existing JavaScript browser security mechanisms:* All JavaScript objects are owned by a principal. Ownership is integrated with the browser's same origin principle for backwards compatibility with Web 2.0 applications. Code owned by an untrusted principal is executed in a controlled environment, but the code has full access to the containing page. This ensures compatibility with existing code.
- *Browser integration:* Our system was implemented in the WebKit library. We instrument all means to create scripts in the browser at runtime, so if untrusted code creates another script we add its security principal to the new script as well. Additionally, we treat the `eval` function as untrusted and always monitor it.
- *Flexible policies:* Our security policies allow enforcement of semantic properties based on the notion of security principals attached to JavaScript objects, rather than mere syntactic properties like method or variable names that previous approaches generally rely on. Policies can be combined, allowing for both provider-specified security and user-defined security.
- *Empirical Evaluation:* We validated our approach on 50 real web sites and two representative policies. The results suggest that our model is a good fit for securing web ad content and third-party extensions, with less than 10% of sites' major functionality broken. Our policies have successfully prevented dangerous operations performed by third-party code. The observed performance overheads were between 11% and 106% in the interpreter.

## References

- [JJLS10] Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. An empirical study of privacy-violating information flows in JavaScript web applications. In *CSS*, pages 270–283. ACM, 2010.
- [RHZN<sup>+</sup>13] Gregor Richards, Christian Hammer, Francesco Zappa Nardelli, Suresh Jagannathan, and Jan Vitek. Flexible Access Control for JavaScript. In *OOPSLA*, pages 305–322. ACM, October 2013.
- [Sch00] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3:30–50, February 2000.

# SPL<sup>LIFT</sup>— Statically Analyzing Software Product Lines in Minutes Instead of Years

Eric Bodden<sup>1</sup> Társis Tolêdo<sup>3</sup> Márcio Ribeiro<sup>3,4</sup>  
Claus Brabrand<sup>2</sup> Paulo Borba<sup>3</sup> Mira Mezini<sup>1</sup>

<sup>1</sup>EC SPRIDE, Technische Universität Darmstadt, Darmstadt, Germany

<sup>2</sup>IT University of Copenhagen, Copenhagen, Denmark

<sup>3</sup>Federal University of Pernambuco, Recife, Brazil

<sup>4</sup>Federal University of Alagoas, Maceió, Brazil

bodden@acm.org, {tw, phmb}@cin.ufpe.br, marcio@ic.ufal.br  
brabrand@itu.dk, mira.mezini@cased.de

**Abstract:** A software product line (SPL) encodes a potentially large variety of software products as variants of some common code base. Up until now, re-using traditional static analyses for SPLs was virtually intractable, as it required programmers to generate and analyze all products individually. In this work, however, we show how an important class of existing inter-procedural static analyses can be transparently lifted to SPLs. Without requiring programmers to change a single line of code, our approach SPL<sup>LIFT</sup> automatically converts any analysis formulated for traditional programs within the popular IFDS framework for inter-procedural, finite, distributive, subset problems to an SPL-aware analysis formulated in the IDE framework, a well-known extension to IFDS. Using a full implementation based on Heros, Soot, CIDE and JavaBDD, we show that with SPL<sup>LIFT</sup> one can reuse IFDS-based analyses without changing a single line of code. Through experiments using three static analyses applied to four Java-based product lines, we were able to show that our approach produces correct results and outperforms the traditional approach by several orders of magnitude.

A Software Product Line (SPL) describes a set of software products as variations of a common code base. Variations, so-called features, are typically expressed through compiler directives such as the well-known `#ifdef` from the C pre-processor or other means of conditional compilation.

Static program analyses are a powerful tool to find bugs in program code [GPT<sup>+</sup>11, FYD<sup>+</sup>08] or to conduct static optimizations [SHR<sup>+</sup>00], and it is therefore highly desirable to apply static analyses also to software product lines. With existing approaches, though, it is often prohibitively expensive to reuse existing static analyses. The problem is that traditional static analyses cannot be directly applied to software product lines. Instead they have to be applied to pre-processed programs. But for an SPL with  $n$  optional features, there are  $2^n$  possible products, which therefore demands thousands of analysis runs even for small product lines. This exponential blowup is particularly annoying because many of those analysis runs will have large overlaps for different feature combinations. It therefore seems quite beneficial to share analysis information wherever possible.



In this work we introduce  $\text{SPL}^{\text{LIFT}}$ , a simple but very effective approach to re-using existing static program analyses without an exponential blowup.  $\text{SPL}^{\text{LIFT}}$  allows programmers to transparently lift an important class of existing static analyses to software product lines. Our approach is fully inter-procedural. It works for any analysis formulated for traditional programs within Reps, Horwitz and Sagiv’s popular IFDS [RHS95] framework for inter-procedural, finite, distributive, subset problems.  $\text{SPL}^{\text{LIFT}}$  automatically converts any such analysis to a feature-sensitive analysis that operates on the entire product line in one single pass. The converted analysis is formulated in the IDE framework [SRH96] for inter-procedural distributed environment problems, an extension to IFDS. In cases in which the original analysis reports that a data-flow fact  $d$  may hold at a given statement  $s$ , the resulting converted analysis reports a feature constraint under which  $d$  may hold at  $s$ .

At <http://bodden.de/spllift/> we make available our full implementation as open source, along with all data and scripts to reproduce our empirical results. To summarize, our approach presents the following original contributions:

- a mechanism for automatically and transparently converting any IFDS-based static program analysis to an IDE-based analysis over software product lines,
- a full open-source implementation for Java, and
- a set of experiments showing that our approach yields correct results and outperforms the traditional approach by several orders of magnitude.

Our work on  $\text{SPL}^{\text{LIFT}}$  was first published at PLDI 2013 [BTR<sup>+</sup>13].

## References

- [BTR<sup>+</sup>13] Eric Bodden, Társis Tolêdo, Márcio Ribeiro, Claus Brabrand, Paulo Borba, and Mira Mezini. SPLIFT: statically analyzing software product lines in minutes instead of years. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, PLDI ’13, pages 355–364, 2013.
- [FYD<sup>+</sup>08] Stephen J. Fink, Eran Yahav, Nurit Dor, G. Ramalingam, and Emmanuel Geay. Effective tpestate verification in the presence of aliasing. *ACM Trans. Softw. Eng. Methodol.*, 17(2):9:1–9:34, May 2008.
- [GPT<sup>+</sup>11] Salvatore Guarnieri, Marco Pistoia, Omer Tripp, Julian Dolby, Stephen Teilhet, and Ryan Berg. Saving the world wide web from vulnerable JavaScript. In *Proc. 2011 int. symp. on Software Testing and Analysis*, ISSTA ’11, pages 177–187, 2011.
- [RHS95] Thomas Reps, Susan Horwitz, and Mooly Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Proc. 22nd ACM SIGPLAN-SIGACT symp. on Principles of programming languages*, POPL ’95, pages 49–61, 1995.
- [SHR<sup>+</sup>00] Vijay Sundareshan, Laurie Hendren, Chrislain Razafimahefa, Raja Vallée-Rai, Patrick Lam, Etienne Gagnon, and Charles Godin. Practical virtual method call resolution for Java. In *OOPSLA*, pages 264–280, 2000.
- [SRH96] Mooly Sagiv, Thomas Reps, and Susan Horwitz. Precise interprocedural dataflow analysis with applications to constant propagation. In *TAPSOFT ’95*, pages 131–170, 1996.

# C nach Eiffel: Automatische Übersetzung und objektorientierte Umstrukturierung von Legacy Quelltext

Marco Trudel

Lehrstuhl für Softwareentwicklung  
ETH Zürich  
Schweiz  
marco.trudel@inf.ethz.ch

**Abstract:** Ist es möglich einen Teil der riesigen in C entwickelten Code-Basis wiederzuverwenden um die Vorteile moderner Programmiersprachen wie Typsicherheit, Objektorientierung und Verträge nutzen zu können? Dieser Beitrag präsentiert eine Quelltext-zu-Quelltext Übersetzung und objektorientierte Umstrukturierung von C Code nach Eiffel, eine moderne objektorientierte Programmiersprache, und das zugehörige Tool C2Eif. Die Migration ist komplett automatisch und unterstützt die gesamte C Sprache wie sie in der Praxis verwendet wird. Die erstellten Eiffel Programme verfügen über die Eigenschaften guter objektorientierter Programme, wie beispielsweise lose Kopplung und starke Bindung von Klassen sowie geeignete Datenkapselung. Zudem setzen die Programme auch fortschrittliche Merkmale wie Vererbung, Verträge und Ausnahmen ein, um ein klareres Design zu erreichen. Unsere Experimente zeigen dass C2Eif C Anwendungen und Bibliotheken von signifikanter Grösse (wie z.B. vim und libgsl), sowie anspruchsvolle Benchmarks wie die GCC Torture-Tests, bearbeiten kann. Der erzeugte Eiffel Quelltext ist funktional äquivalent zu dem ursprünglichen C Quelltext und nutzt einige Eigenschaften von Eiffel um sichere und einfach zu korrigierende Programme zu erstellen.

## 1 Automatische Übersetzung

Wann immer möglich übersetzt C2Eif C Sprachkonstrukte in äquivalente Eiffel Konstrukte. Das ist der Fall für Funktionen, Variablen, Anweisungen, Ausdrücke, Schleifen und Grundtypen mit ihren Rechenoperationen. In Fällen, in denen in Eiffel keine entsprechenden Konstrukte verfügbar sind, bietet das Tool Simulationen der C Konstrukte und Unterstützung durch Bibliotheken an. So werden beispielsweise Sprungbefehle (`break`, `continue`, `return` und `goto`) durch strukturierten Kontrollfluss und Hilfsvariablen simuliert. Für Pointer wird Bibliotheks-Unterstützung angeboten: sie werden zu Instanzen einer generischen Bibliotheks-Klasse `CE_POINTER [G]` übersetzt, welche die volle C Pointer Funktionalität unterstützt. C Structs werden in Eiffel Klassen übersetzt, die von einer Bibliotheks-Klasse `CE_CLASS` erben. Durch die Verwendung von Reflexion ermöglicht diese Klasse ihre Instanz in ein Objekt zu konvertieren, welches das genaue Speicherlayout des C Structs hat. Dies kann für Pointerarithmetik und Interoperabilität mit vorkompilierten C-Bibliotheken notwendig sein.

## 2 Objektorientierte Umstrukturierung

Die objektorientierte Umstrukturierung extrahiert Elemente mit gutem Design, die in hochwertigem C-Code vorhanden sind, und drückt sie durch objektorientierte Konstrukte und Konzepte aus. Die Umstrukturierung in C2Eif besteht aus vier Schritten, die sich solche implizite Designelemente zu Nutzen macht: (1) *Quelldatei*-Analyse erstellt Klassen und füllt sie basierend auf den C Quelldateien, (2) *Methodensignatur*-Analyse verschiebt Methoden in Klassen auf dessen Daten sie arbeiten, (3) *Aufrufs-Graph*-Analyse verschiebt Felder und Methoden in Klassen in denen sie exklusiv verwendet werden, (4) *Vererbungs*-Analyse erstellt Vererbungsbeziehungen zwischen Klassen anhand ihrer Felder. Neben diesen Kernelementen des objektorientierten Designs werden auch Verträge und Ausnahmen erstellt, basierend auf GNU C Compiler (GCC) Annotationen, Anforderungen an die Argumente von Funktionen sowie Verwendungen der `set jmp` Bibliothek.

## 3 Fazit

Eine Übersicht [Tru13] über verwandte Arbeiten im Bereich der objektorientierten Übersetzung zeigt, dass bisherige Ansätze Einschränkungen in Bezug auf Vollständigkeit, Automatisierung, Anwendbarkeit und Qualität der Übersetzung haben. Im Gegensatz präsentiert unser Beitrag einen Ansatz welcher sich durch folgende Merkmale auszeichnet:

- Die Übersetzung ist *komplett automatisch* und im frei verfügbaren Tool C2Eif implementiert. Benutzer müssen nur ein C Projekt angeben und C2Eif erstellt ein objektorientiertes Eiffel Programm, das kompiliert und ausgeführt werden kann.
- C2Eif unterstützt die gesamte C-Sprache wie sie in der Praxis verwendet wird, einschliesslich Pointerarithmetik, die Benutzung von nativen System-Bibliotheken (zB für I/O), Inline-Assembler-Code und uneingeschränkte Sprungbefehle.
- Eine umfangreiche Auswertung mit realer Software von beträchtlicher Grösse zeigt dass unsere Übersetzung objektorientierten Code mit hoher Kapselung erzeugt und Vererbung, Verträge, und Ausnahmen angemessen einsetzt.
- Die Übersetzung führt keine potentiell inkorrekten Transformationen durch und erzeugt somit Programme die funktional äquivalent mit den C Programmen sind.

Ausführliche Informationen können in den Veröffentlichungen [TFNM13, TFN<sup>+</sup>12], der Dissertation [Tru13] und auf der Projektwebseite [C2E] gefunden werden.

*Danksagung:* Die hier präsentierten Forschungsergebnisse stammen aus der Zusammenarbeit mit Carlo A. Furia und Martin Nordio.

## Literatur

- [C2E] C2Eif. Der C nach Eiffel Übersetzer. <http://se.inf.ethz.ch/research/c2eif/>.
- [TFN<sup>+</sup>12] Marco Trudel, Carlo A. Furia, Martin Nordio, Bertrand Meyer und Manuel Oriol. C to O-O Translation: Beyond the Easy Stuff. In *WCRE*, 2012.
- [TFNM13] Marco Trudel, Carlo A. Furia, Martin Nordio und Bertrand Meyer. Really Automatic Scalable Object-Oriented Reengineering. In *ECOOP*, 2013.
- [Tru13] Marco Trudel. *Automatic Translation and Object-Oriented Reengineering of Legacy Code*. Dissertation, ETH Zurich, 2013.

# Robustness against Relaxed Memory Models

Ahmed Bouajjani<sup>1</sup>, Egor Derevenetc<sup>2,3</sup>, and Roland Meyer<sup>2</sup>

<sup>1</sup>LIAFA, University Paris Diderot, and IUF

<sup>2</sup>University of Kaiserslautern      <sup>3</sup>Fraunhofer ITWM

abou@liafa.univ-paris-diderot.fr  
{derevenetc, meyer}@cs.uni-kl.de

**Abstract:** For performance reasons, modern multiprocessors implement relaxed memory consistency models that admit out-of-program-order and non-store atomic executions. While data race-free programs are not sensitive to these relaxations, they pose a serious problem to the development of the underlying concurrency libraries. Routines that work correctly under Sequential Consistency (SC) show undesirable effects when run under relaxed memory models. These programs are not robust against the relaxations that the processor supports. To enforce robustness, the programmer has to add safety net instructions to the code that control the hardware — a task that has proven to be difficult, even for experts.

We recently developed algorithms that check and, if necessary, enforce robustness against the Total Store Ordering (TSO) relaxed memory model [BDM13, BMM11]. Given a program, our procedures decide whether the TSO behavior coincides with the SC semantics. If this is not the case, they synthesize safety net instructions that enforce robustness. When built into a compiler, our algorithms thus hide the memory model from the programmer and provide the illusion of Sequential Consistency.

## Summary

Sequential Consistency (SC) is the memory consistency model that programmers typically expect. It reflects the idea of a global shared memory on which instructions take effect instantaneously. Consider the variant of Dekker’s mutex depicted below. Under SC, the locations `cs 1` and `cs 2` cannot be reached simultaneously.

While SC is intuitive, it forbids important hardware and compiler optimizations, and is therefore not implemented in existing processors. Instead, modern architectures realise

*relaxed* memory consistency models that weaken the program order and store atomicity guarantees of SC. x86 processors for instance implement the Total Store Ordering (TSO) relaxed memory model where stores may be delayed past later loads to different addresses. These delays capture the use of store buffers in the architecture: stores are gathered and later batch processed to reduce the latency of memory accesses. In Dekker’s mutex, a TSO execution may swap the instructions `st(x, 1)` and `ld(y, 0)`, and execute the four commands in the order 1 . 4 . indicated by the numbers.

```
Thrd 1:      Thrd 2:
4. st(x, 1);  2. st(y, 1);
1. ld(y, 0);  3. ld(x, 0);
// cs 1      // cs 2
```

Dekker’s protocol, `ld(y, 0)` is only executable if address `y` holds 0.

Programs that are correct under SC may show undesirable effects when run under relaxed memory models. As the example shows, already TSO (a relaxed memory model with strong guarantees) breaks Dekker’s mutex. The importance of SC stems from the *data*

*race freeness (DRF) guarantee*: on virtually all processors, DRF programs are guaranteed to have SC semantics. Performance-critical routines in operating systems and concurrency libraries, however, intentionally use data races and hence DRF does not apply. In these application domains, developers design their programs to be *robust* against the relaxations of the execution environment. Robustness requires that every relaxed execution is equivalent to some SC execution, where equivalent means they have the same control and data dependencies. These dependencies are captured by the classical *happens-before traces*, which means robustness amounts to  $Traces_{RMM}(Prog) \subseteq Traces_{SC}(Prog)$ . The inclusion guarantees that properties of the SC semantics (like mutual exclusion) carry over to the relaxed memory model, although the executions actually differ from SC.

To ensure robustness, architectures offer safety net instructions that synchronize the views of the threads on the shared memory. The problem is that this synchronization has a negative impact on performance. Therefore, programmers insert a small but for robustness yet sufficient number of safety net instructions into the code. As recent bugs in popular libraries have shown, this manual insertion is prone to errors.

We develop algorithms that check and, if necessary, enforce robustness of programs against the relaxed memory model of the targeted architecture. Given a program, our procedures decide whether the relaxed behavior coincides with the SC semantics. If this is not the case, they synthesize a minimal number of safety net instructions that enforce robustness. When built into a compiler, our algorithms thus hide the memory model from the programmer and provide the illusion of Sequential Consistency.

For TSO, our work [BMM11] proved robustness decidable and only PSPACE-complete. This was the first complete solution to the problem. We then extended the approach to handle general programs [BDM13], independently from the data domain they manipulate, without a bound on the shared memory size, nor on the size of store buffers in the TSO semantics, nor on the number of threads. Technically, the general solution is a reduction of robustness to SC reachability in a *linear-sized source-to-source translation* of the input program. With this reduction, robustness checking can harness all techniques and tools that are available for solving SC reachability (either exactly or approximately).

For relaxed memory models beyond TSO, robustness is addressed in the DFG project *R2M2: Robustness against Relaxed Memory Models*. On the theoretical side, our goal is to develop a *proof method* for computability and complexity results about robustness. The idea is to combine combinatorial reasoning about relaxed computations with language-theoretic methods [CDMM13]. On the practical side, we will address robustness against the popular and highly relaxed *POWER processors* and study robustness for *concurrency libraries* that act in an unknown environment.

## References

- [BDM13] A. Bouajjani, E. Derevenetc, and R. Meyer. Checking and Enforcing Robustness against TSO. In *ESOP*, volume 7792 of *LNCS*, pages 533–553. Springer, 2013.
- [BMM11] A. Bouajjani, R. Meyer, and E. Möhlmann. Deciding Robustness against Total Store Ordering. In *ICALP*, volume 6756 of *LNCS*, pages 428–440. Springer, 2011.
- [CDMM13] G. Călin, E. Derevenetc, R. Majumdar, and R. Meyer. A Theory of Partitioned Global Address Spaces. In *FSTTCS*, volume 24 of *LIPIcs*, pages 127–139. Dagstuhl, 2013.

# How Much Does Unused Code Matter for Maintenance?

Sebastian Eder, Maximilian Junker, Benedikt Hauptmann, Elmar Juergens,  
Rudolf Vaas, Karl-Heinz Prommer

Institut für Informatik  
Technische Universität München  
Boltzmannstr. 3  
85748 Garching b. München  
{eders, junkerm, hauptmab}@in.tum.de  
juergens@cqse.eu  
{rvaas,hprommer}@munichre.com

**Abstract:** Software systems contain unnecessary code. Its maintenance causes unnecessary costs. We present tool-support that employs dynamic analysis of deployed software to detect *unused* code as an approximation of *unnecessary* code, and static analysis to reveal its changes during maintenance. We present a case study on maintenance of unused code in an industrial software system over the course of two years. It quantifies the amount of code that is unused, the amount of maintenance activity that went into it and makes the potential benefit of tool support explicit, which informs maintainers that are about to modify unused code.

## 1 Introduction

Many software systems contain unnecessary functionality. In [Joh], Johnson reports that 45% of the features in the analyzed systems were never used. Our own study on the usage of an industrial business information system [JFH<sup>+</sup>11] showed that 28% of its features were never used.

Maintenance of unnecessary features is a waste of development effort. To avoid such waste, maintainers must know which code is still used and useful, and which is not. Unfortunately, such information is often not available to software maintainers.

**Problem:** Real-world software contains unnecessary code. Its maintenance is a waste of development resources. Unfortunately, we lack tool support to identify unnecessary code and empirical data on the magnitude of its impact on maintenance effort.

**Contribution:** We contribute a case study that analyzes the usage of an industrial business information system of the reinsurance company Munich Re over the period of over 2 years. The study quantifies maintenance effort in unused code and shows the potential benefits of the tool support we propose.

**Remarks:** A complete version of the paper, the study, and a presentation of our tool support can be found in [EJJ<sup>+</sup>12].

## 2 Study and Results

We conducted the study on the level of methods in the sense of object oriented programming. The systems contains 25,390 methods. Of these, 6,028 were modified with a total of 9,987 individual modifications. This means that considerable maintenance effort took place during the analysis period.

**RQ1: How much code is unused in industrial systems?** We found that 25% of all methods were never used during the complete period.

**RQ2: How much maintenance is done in unused code?** We first compared the degree of maintenance (i.e. percentage of maintained methods) between used and unused methods. We found that 40.7% of the used methods were maintained, but only 8.3% of the unused methods. That means, unused methods were maintained less intensively than used methods. The unused methods account for 7.6% of the total number of modifications.

**RQ3: How much maintenance in unused code is unnecessary?** We reviewed examples of unused maintenance with the developers. By inspecting the affected code and researching the reason why it is not used, we found that in 33% of the cases, the unused code was indeed unnecessary. In another 15% of the cases the code in question was no longer existent as it was either deleted or moved. That means that in nearly every second case unused methods were either unnecessary or potentially deleted from the system.

**RQ4: Do maintainers perceive knowledge of unused code useful for maintenance tasks?** We encountered great interest in the analysis results, especially in the cases in which unused methods were maintained. Often, the developers were surprised that the respective method was not used.

## 3 Summary

We believe that our analysis would show a greater amount of unnecessary maintenance for projects with a different structure of the maintaining team, since the maintaining team knew the system very well. Due to the results, we are optimistic that this analysis helps directing maintenance efforts more effectively.

## References

- [EJJ<sup>+</sup>12] S. Eder, M. Junker, E. Jurgens, B. Hauptmann, R. Vaas, and K. Prommer. How much does unused code matter for maintenance? In *ICSE '12*, 2012.
- [JFH<sup>+</sup>11] E. Juergens, M. Feilkas, M. Herrmannsdoerfer, F. Deissenboeck, R. Vaas, and K. Prommer. Feature Profiling for Evolving Systems. In *ICPC '11*, 2011.
- [Joh] Jim Johnson. ROI, It's Your Job. Keynote at XP '02.

# **The SecReq Approach: From Security Requirements to Secure Design while Managing Software Evolution**

J. Jürjens

K. Schneider

TU Dortmund & Fraunhofer ISST  
Dortmund

<http://www-secse.cs.tu-dortmund.de>

Leibniz Universität Hannover  
Hannover

<http://www.se.uni-hannover.de>

**Abstract:** We present the security requirements & design approach SecReq developed in joint work over the last few years. As a core feature, this approach supports reusing security engineering experience gained during the development of security-critical software and feeding it back into the development process through the HeRA Heuristic Requirements Assistant. Based on this information a model-based security analysis of the software design can be performed using the UMLsec approach and its associated tool-platform CARiSMA. In recent work within the project DFG project SecVolution (SPP 1593 “Design For Future – Managed Software Evolution”), we have been extending the approach with techniques, tools, and processes that support security requirements and design analysis techniques for evolving information systems in order to ensure “lifelong” compliance to security requirements. heuristic tools and techniques that support elicitation of relevant changes in the environment.

## **1 From Security Requirements to Secure Design**

Many software projects today are somehow security-related. Requirements engineers without expertise in security often overlook security requirements, leading to security vulnerabilities that can later be exploited. Identifying security-relevant requirements is labour-intensive and error-prone.

To facilitate the security requirements elicitation process, [SKHIJ12] presents an approach supporting organisational learning on security requirements by establishing company-wide experience resources, based on modelling the flow of requirements and related experiences. Based on those models, people can exchange experiences about security-relevant requirements while writing and discussing project requirements. Participating stakeholders can learn while writing requirements. This increases security awareness and facilitates learning on individual and organisational levels. As a tool basis, heuristic assistant tools [Sch08] like HeRA [SKHIJ12] support reuse of existing experiences that are relevant for security. They include Bayesian classifiers issuing a warning automatically when new requirements seem security-relevant. The approach is part of the SecReq approach introduced in [HIKJS10] and feeds into the UMLsec of which a recent application is reported in [LJ09].



Results indicate that this is feasible, in particular if the classifier is trained with domain-specific data and documents from previous projects. The paper shows how the ability to identify security-relevant requirements can be improved using this approach. It illustrates the approach with a step-by-step example of how it improved the security requirements engineering process at the European Telecommunications Standards Institute (ETSI) and reports on experiences made.

## 2 Maintaining Security Requirements During Software Evolution

Information systems are exposed to constantly changing environments which require constant updating. Software "ages" not by wearing out, but by failing to keep up-to-date with its environment. Security is an increasingly important quality aspect in modern information systems. At the same time, it is particularly affected by the above-mentioned risk of "software ageing". When an information system handles assets of a company or an organization, any security loophole can be exploited by attackers. Advances in knowledge and technology of attackers are part of the above-mentioned environment of a security-relevant information system. Outdated security precautions can, therefore, permit sudden and substantial losses. Security in long-living information systems, thus, requires an on-going and systematic evolution of knowledge and software.

In recent work within the project DFG project SecVolution (SPP 1593 "Design For Future – Managed Software Evolution"), we have been developing techniques, tools, and processes that support security requirements and design analysis techniques for evolving information systems in order to ensure "lifelong" compliance to security requirements, building on the SecReq approach. As a core feature, this approach supports reusing security engineering experience gained during the development of security-critical software and feeding it back into the development process. We develop a variety of heuristic tools and techniques that support elicitation of relevant changes in the environment. Findings are formalized for semi-automatic security updates. During the evolution of a long-living information system, changes in the environment are monitored and translated to adaptations that preserve or restore its security level.

## References

- [HIKJS10] S.H. Houmb, S. Islam, E. Knauss, J. Jürjens, K. Schneider: Eliciting security requirements and tracing them to design: an integration of Common Criteria, heuristics, and UMLsec. *Requir. Eng.* 15(1): 63-93 (2010).
- [Jür05] J. Jürjens: *Secure systems development with UML*. Springer 2005
- [LJ09] J. Lloyd, J. Jürjens: Security Analysis of a Biometric Authentication System Using UMLsec and JML. *MoDELS 2009*: 77-91.
- [MJ10] H. Mouratidis, J. Jürjens: From goal-driven security requirements engineering to secure design. *Int. J. Intell. Syst.* 25(8): 813-840 (2010)
- [Sch08] Schneider, K.: Improving Feedback on Requirements through Heuristics. *Proceedings of 4th World Congress for Software Quality (4WCSQ)*, Washington D.C., USA (2008)
- [SKHIJ12] K. Schneider, E. Knauss, S.H. Houmb, S. Islam, J. Jürjens: Enhancing Security Requirements Engineering by Organisational Learning in Requirements Engineering, *Requirements Engineering Journal*, Vol. 17, 2012, pp.35-56.

# Noninvasive regelbasierte Graphtransformation für Java\*

James J. Hunt  
aicas GmbH, Karlsruhe, Germany  
jjh@aicas.com

Arend Rensink and Maarten de Mol  
Department of Computer Science, University of Twente  
P.O.Box 217, 7500 AE, The Netherlands  
rensink@cs.utwente.nl

Der Standardansatz zur Verwendung von Graphtransformationen (GT) in einem Programm ist es, die Daten in eine bekannte Graphenstruktur umzuwandeln, diese zu bearbeiten und sie dann zurück zu wandeln in das Format des Programms. Diese Transformationen stellen sicher, dass die graphischen Operationen korrekt sind. Die Autoren haben eine alternative Methode zur Verwendung der Java-Programme entwickelt, die den Graphtransformator darüber informiert, wie Datenstrukturen eines Programms, die einem idealen Graph entsprechen, so dass in situ Graphtransformationen direkt auf der vorliegenden Datenstruktur durchgeführt werden können, ohne unter einem Korrektheitsverlust zu leiden. Der Vorteil des Ansatzes ist, dass er es ermöglicht, beliebige Java-Programme nicht-invasiv mit deklarativen Graphenregeln zu erweitern. Dieser verbessert die Klarheit, Prägnanz, Nachprüfbarkeit und Leistung.

Schwachstellen der GT sind ihre mangelnde Effizienz und die Notwendigkeit, Daten zwischen dem Anwendungsgebiet und der Graphdomäne zu transformieren. Obwohl mangelnde Effizienz zum Teil der Preis für allgemeine Anwendbarkeit sein kann, so scheint dies nicht der dominierende Faktor zu sein. Viel mehr ist es der Overhead der Transformation. Die beiden “Transformationen” sind eigentlich auch Modelltransformationen, und erhöhen die Komplexität der Technik so weit, dass es unpraktisch für große Graphen wird. Auch eine Anwendung zu erzwingen, die Graphenstruktur des Werkzeugs zu verwenden, hilft nicht viel, da die Graphenstruktur des Werkzeugs nicht ausdrucksfähig genug für die Anwendung sein kann, so dass bestehender Code neu geschrieben werden muss. Daher sind etablierte GT-Techniken invasiv.

Der hier vorgestellte Ansatz stellt diesen Prozess auf den Kopf. Da er Graphensemantik zu dem Problem und nicht das Problem zur Graphensemantik bringt, vermeidet es die invasive Natur der GT, erhält jedoch die Vorteile der GT, einschließlich der allgemeinen Anwendbarkeit und der deklarativen Natur. Es gibt drei Hauptteile dieses Ansatzes.

- Die Graphenstruktur wird spezifiziert durch Einfügen von JAVA-Annotationen in vorhandenen Klassencode. Als Konsequenz bilden die JAVA-Annotationen eine Spezifikationssprache für Graphen.

---

\*Diese Arbeit wurde von der Artemis Joint Undertaking durch das CHARTER-Projekt gefördert, Grant-Nr. 100039. Siehe <http://charterproject.ning.com/>.

- Graphenmanipulation, wie etwas Hinzufügen oder Löschen von Knoten und Kanten, werden durch applikationsdefinierte Operationen bereitgestellt, die auch Annotationen benötigen, um ihre Wirkung auf die Graphenstruktur zu beschreiben.
- Regeln werden in einer deklarativen Sprache namens CHART geschrieben und anschließend in JAVA-Code übersetzt. Dieser bearbeitet die oben genannten Benutzerklassen mit annotierten Methoden ohne die Übertragung von Datenstrukturen zu und von der Graphendomäne.

Dieser Ansatz ermöglicht es, Komponenten eines bestehenden JAVA-Programms durch deklarative Graphtransformationen zu ersetzen, nur mit der Anforderung, die Datenstrukturen des Programms mit Annotationen zur ergänzen. Der Ansatz wurde in dem CHARTER-Projekt entwickelt, bei dem er in drei verschiedenen Werkzeugen angewendet wurde.

Um Graphentransformationsregeln auf einem vorhandenen JAVA-Programm mit diesem Ansatz anzuwenden, müssen folgende Änderungen vorgenommen werden:

- eine `@Node` Annotation muss an jeder Klasse und Schnittstelle angefügt werden, die einen Knoten darstellt;
- für jeden Kantentyp muss eine dedizierte Schnittstelle mit einer `@Edge` Annotation definiert werden und
- für jede gewünschte Operation muss eine Methode zur Verfügung stehen, entweder durch das Annotieren einer bestehenden Methode oder das Hinzufügen einer neuen Methode mit der entsprechenden Annotation.

Diese Änderungen bereichern den vorhandenen Code lediglich durch Meta-Daten. Sie können daher an jedes JAVA-Programm angewendet werden. Daher wird dieser Ansatz *non-invasiv* genannt. Dies sollte nicht mit “nonmodifying” verwechselt werden, weil Annotationen und Schnittstellen zugefügt und zusätzliche Manipulationsmethoden implementiert werden müssen.

Eine CHART-Transformation besteht aus Transformationsregeln und kann durch Aufrufen von einer dieser Regeln gestartet werden. Jede Regel hat eine Signatur, die ihren Namen, ihre Parameter und Rückgabewerte definiert. Mehrere Ein- und Ausgabewerte sind erlaubt. Der Körper einer Regel besteht aus jeweils höchstens einem der folgenden Blöcke in der dieser Reihenfolge: ein Match-, ein Update-, ein Sequence- und ein Return-Block.

Ein großer Teil des Aufwands ist in den CHART Compiler geflossen, der den entsprechenden JAVA-Code generiert. Der Compiler wird RDT genannt, was für Regel Driven Transformer steht. Er unterstützt alle beschriebenen Funktionen. Er wurde bereits in vier verschiedenen Systemen erfolgreich verwendet. Die Komplexeste davon ist für die Optimierung in einem neuen Byte-Code-Compiler von aicas.

Obwohl CHART sich in der Praxis bewährt hat, gibt es noch viel zu tun, um die formalen Grundlagen stärken zu können und die Benutzbarkeit zu verbessern. Eine Theorie, die es einem CHART Programmierer ermöglicht, den Zusammenfluss und die Terminierung seiner Regelsystem abzuleiten, die semantische Erhaltung der Modelltransformationen zu beweisen und eine formale Überprüfung des RDT wären hilfreich. Auf der pragmatischen Seite benötigt der RDT weitere Arbeit an Effizienz, vielleicht mit einem besseren Matching-Algorithmus. Dies könnte die Leistung des erzeugten Codes verbessern. Schließlich könnte die CHART Sprache mit zusätzlicher Funktionalität erweitert werden.

# Second-Order Constraints in Dynamic Invariant Inference

Kaituo Li<sup>1</sup>, Christoph Reichenbach<sup>2</sup>, Yannis Smaragdakis<sup>3</sup>, Michal Young<sup>4</sup>

<sup>1</sup> Computer Science Department  
University of Massachusetts  
Amherst, MA 01003, USA  
kaituo@cs.umass.edu

<sup>2</sup> Institut für Informatik  
Goethe University Frankfurt  
60054 Frankfurt, Germany  
reichenbach@cs.uni-frankfurt.de

<sup>3</sup> Department of Informatics  
University of Athens  
Athens 15784, Greece  
smaragd@di.uoa.gr

<sup>4</sup> Computer Science Department  
University of Oregon  
Eugene, OR 97403, USA  
michal@cs.uoregon.edu

## Abstract:

Today’s dynamic invariant detectors often produce invariants that are inconsistent with program semantics or programmer knowledge. We improve the consistency of dynamically discovered invariants by considering second-order constraints. These constraints encode knowledge *about* invariants, even when the invariants themselves are unknown. For instance, even though the invariants describing the behavior of two functions  $f1$  and  $f2$  may be unknown, we may know that any valid input for  $f1$  is also valid for  $f2$ , i.e., the precondition of  $f1$  implies that of  $f2$ .

We explore an implementation of second-order constraints on top of the Daikon system. Our implementation provides a vocabulary of constraints that the programmer can use to enhance and constrain Daikon’s inference. We show that dynamic inference of second-order constraints together with minimal human effort can significantly influence the produced (first-order) invariants even in systems of substantial size, such as the Apache Commons Collections and the AspectJ compiler. We also find that 99% of the dynamically inferred second-order constraints we sampled are true.

## 1 Overview

In this work we enhance the dynamic invariant detection of tools such as Daikon [ECGN01] by including second-order constraints. The invariants inferred should be consistent with these constraints. More specifically, we identify a vocabulary of constraints on inferred invariants. We call these constraints “second-order” because they are constraints over constraints: they relate classes of invariants (first-order constraints). Such second-order constraints can be known even though the invariants are unknown.

We have developed two extensions to the Daikon inference process to utilize second-order constraints (Figure 1). Our system supports two scenarios: first, hand-written second-order constraints from users, and second, automatically inferred second-order constraints, obtained from known first-order invariants.

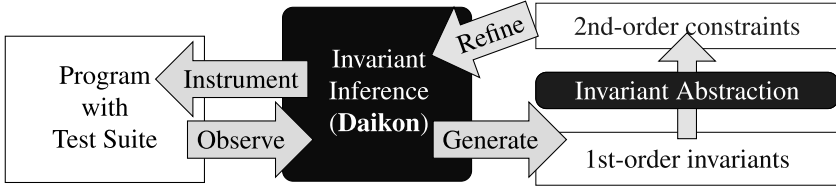


Figure 1: Inference architecture, extended by second-order constraints. We use second-order constraints (green) to *Refine* invariant inference. Conversely, we distill (first-order) invariants into (second-order) constraints through *Invariant Abstraction*.

Our five main forms of second-order constraints are over pairs of methods  $m_1$  and  $m_2$  and relate their pre- and postconditions. In this way, we have  $SubDomain(m_1, m_2)$  signifying that the pre-condition of  $m_1$  implies that of  $m_2$ . Similarly, we have the constraint  $SubRange(m_1, m_2)$  denoting that the post-condition of  $m_1$  implies that of  $m_2$ . The remaining constraints arise in the same manner:  $CanFollow(m_1, m_2)$  expresses that the post-condition of  $m_1$  implies the pre-condition of  $m_2$ , as in  $CanFollow(open, read)$  (“open enables read”) for file operations.  $Follows(m_2, m_1)$  expresses the converse implication. Finally,  $Concord(m_1, m_2)$  expresses that  $m_1$  guarantees at least as much as  $m_2$  does, for the set of preconditions that *both* methods agree on. This is useful to express e.g. that  $m_2$  is an optimized version of  $m_1$  for a restricted set of inputs.

## 2 Conclusions

Our experiments with hand-written second-order constraints in three test suites, including the Apache Commons Collections and AspectJ, show that our constraints can improve the quality of generated first-order invariants. Separately, our experiments with automatically inferring second-order constraints from previously inferred first-order invariants show a very high rate (99%) of correct constraints. Overall, we consider second-order constraints to be a particularly promising idea not just as a meaningful documentation concept but also for improving the consistency and quality of dynamically inferred invariants. Our full paper [LRSY13] discusses our results and insights in detail.

## References

- [ECGN01] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, February 2001.
- [LRSY13] Kaituo Li, Christoph Reichenbach, Yannis Smaragdakis, and Michal Young. Second-order Constraints in Dynamic Invariant Inference. *ESEC/FSE 2013*, pages 103–113, New York, NY, USA, 2013. ACM.

# Revisited: Testing Culture on a Social Coding Site

Raphael Pham<sup>\*</sup>, Leif Singer<sup>†</sup>, Olga Liskin<sup>\*</sup>, Fernando Figueira Filho<sup>‡</sup>, Kurt Schneider<sup>\*</sup>

<sup>\*</sup> Software Engineering Group  
Leibniz Universität Hannover, Germany  
{firstname.lastname}@inf.uni-hannover.de

<sup>†</sup> CHISEL Group  
University of Victoria, Canada  
lsinger@uvic.ca

<sup>‡</sup> Cooperative Interaction Lab  
Universidade Federal do Rio Grande do Norte, Brazil  
fernando@dimap.ufrn.br

**Abstract:** Testing is an important part of software development. However, creating a common understanding of a project’s testing culture is a demanding task. Without it, the project’s quality may degrade. We conducted a Grounded Theory study to understand how testing culture is communicated and disseminated in projects on GitHub. We investigated how the transparency of interactions on the site influences the testing behavior of developers. We found several strategies that software developers and managers can use to positively influence the testing behavior in their projects. We report on the challenges and risks caused by this and suggest guidelines for promoting a sustainable testing culture in software development projects.

## 1 Social Transparency and Testing Culture on GitHub

Social coding sites provide a high degree of social transparency ([SDK<sup>+</sup>12]). Members are able to easily find out who they are interacting with, whom everyone else is interacting with, and who has interacted with which artifacts. This transparency influences the behavior of software developers [DSTH12]). The social coding site GitHub.com acts as a version control repository with a Web interface, as well as a social network site. Users (*contributors*) browse projects, clone a public repository of interest and make changes to it. Then, they offer these changes back (*making a pull request*) to the *project owner*, who decides whether or not to accept them.

In our study, we explored the prevalent testing behavior on GitHub and the impact of social transparency on it (see [PSL<sup>+</sup>13]). GitHub’s **contribution process** is straightforward: a project owner receives a pull request, manually inspects it, runs a test suite and merges it. However, different factors influence this process. Contributions from unknown developers were checked more thoroughly (*trust*). Small changes (*size*) were accepted without tests while new features (*type*) triggered a demand for automated tests. In our study, several **challenges** for GitHub users became apparent. Project owners felt a need for automated tests simply for reasons of *scale* (too many contributions were flowing in). The *constant flux* of different contributors and the shortness of engagement made it difficult to effectively communicate requirements for automated tests. Different **coping strategies**

emerged: Project owners *lowered the barriers* for contributors to provide tests by using well-known testing frameworks, providing easy *access to learning resources* or *actively supporting* users in writing tests. Contributors reacted to obvious signals for automated testing. They were more inclined to provide tests in their contributions, if they saw automated tests already present in a project. Moreover, contributors heavily relied on existing tests as examples for their own test cases. The **impact of social transparency** on testing behavior was manifold: Some projects used their testing practices as advertisement for high quality development. Effective communication of testing guidelines removed uncertainties in contributors about how to participate correctly. Also, contributors to well-tested projects reported to feel more confident as problems would quickly become visible.

## 2 Conclusion and Outlook

Project owners on a social coding site interact with contributors with varying values regarding testing. Our study reports on the influences of GitHub's high degree of social transparency, low barriers, and high degrees of integration and centralization on testing practices. On GitHub, developers browse for projects of interest, contribute swiftly and gradually get more involved. Other users quickly contribute without further involvement. This creates large peripheries of contributors for popular projects. In an ongoing initiative [PSS13], we are exploring how to direct this peripheral potential towards automated testing by using crowdsourcing mechanisms. This way, projects could make their needs for automated tests more visible to peripheral users. Understanding the impact of social transparency on testing behavior is a key factor in designing a suitable crowdsourcing platform for software testing. Lastly, our findings can help developers to gain insights into issues that contributors may face and strategies for handling them.

## References

- [DSTH12] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proc. of the ACM 2012 Conf. on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.
- [PSL<sup>+</sup>13] Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Kurt Schneider. Creating a Shared Understanding of Testing Culture on a Social Coding Site. In *Proceedings of the 35th International Conference on Software Engineering (ICSE 2013)*, pages 112 - 121, San Francisco, USA, 2013.
- [PSS13] Raphael Pham, Leif Singer, and Kurt Schneider. Building Test Suites in Social Coding Sites by Leveraging Drive-By Commits. In *Proceedings of the 35th International Conference on Software Engineering (ICSE 2013, NIER Track)*, pages 1202 - 1212, San Francisco, USA, 2013.
- [SDK<sup>+</sup>12] H. Colleen Stuart, Laura Dabbish, Sara Kiesler, Peter Kinnaird, and Ruogu Kang. Social transparency in networked information exchange: a theoretical framework. In *Proc. of the ACM 2012 Conf. on Computer Supported Cooperative Work, CSCW '12*, pages 451–460, New York, NY, USA, 2012. ACM.

# Reusing Information in Multi-Goal Reachability Analyses \*

Dirk Beyer<sup>1</sup>, Andreas Holzer<sup>2</sup>, Michael Tautschnig<sup>3</sup>, Helmut Veith<sup>2</sup>

<sup>1</sup> University of Passau  
Software Systems  
Innstrasse 33  
D-94032 Passau, Germany

<sup>2</sup> Vienna University of Technology  
Institut für Informationssysteme 184/4  
Favoritenstraße 9-11  
A-1040 Vienna, Austria

<sup>3</sup> School of Electronic Engineering and Computer Science  
Queen Mary University of London  
Mile End Road  
London E1 4NS, UK

**Abstract:** It is known that model checkers can generate test inputs as witnesses for reachability specifications (or, equivalently, as counterexamples for safety properties). While this use of model checkers for testing yields a theoretically sound test-generation procedure, it scales poorly for computing complex test suites for large sets of test goals, because each test goal requires an expensive run of the model checker. We represent test goals as automata and exploit relations between automata in order to reuse existing reachability information for the analysis of subsequent test goals. Exploiting the sharing of sub-automata in a series of reachability queries, we achieve considerable performance improvements over the standard approach. We show the practical use of our multi-goal reachability analysis in a predicate-abstraction-based test-input generator for the test-specification language FQL.

## Overview

We consider the problem of performing many reachability queries on a program that is given as source code. Querying a model checker repeatedly for path-sensitive reachability information [BCH<sup>+</sup>04b] has many interesting applications, e.g., to decompose verification tasks, but most prominently to generate test cases from counterexample paths [BCH<sup>+</sup>04a, HSTV08]. If, for example, we want to achieve basic-block coverage, we will for each basic block  $b$  try to construct a path through the program that witnesses a program execution that reaches  $b$ . In our approach, we describe test-coverage criteria using the coverage-specification language FQL [HSTV08, HSTV09, HSTV10, HTSV10, Hol13], which provides a concise specification of complex coverage criteria. We translate an FQL coverage criterion into a (possibly huge) set of test goals. Each such test goal is represented as a finite automaton, called *test-goal automaton*, and specifies a reachability query. The model-checking engine then takes a test-goal automaton to restrict the state-space search to the specified paths (for which test cases are desired). Test-goal automata often have identical parts which let us reuse analysis results across several queries. We developed an

---

\*This is a summary of [BHTV13]. This work was supported by the Canadian NSERC grant RGPIN 341819-07, by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF), by the Vienna Science and Technology Fund (WWTF) grant PROSEED, and by the EPSRC project EP/H017585/1.



approach that exploits the automaton structure of reachability queries to efficiently reuse reachability results when solving multiple queries. Given two test-goal automata  $A$  and  $A'$ , we introduce the notion of *similarity of  $A$  and  $A'$  modulo a set  $X$  of transitions*, where  $X$  is a subset of the transitions of  $A'$ . We then identify potentially shared behavior between  $A$  and  $A'$  via a *simulation-modulo- $X$  relation  $H$*  between the states of  $A$  and  $A'$ . If two states  $s$  and  $s'$  are related via  $H$ , then each sequence of  $A'$ -transitions starting in  $s'$  and not including a transition from  $X$  corresponds to an equivalent sequence of  $A$ -transitions starting in  $s$ . This allows us to reason about feasibility of program executions that are covered by  $A'$  based on the reachability results for  $A$  as long as we investigate transition sequences shared by both automata [BHTV13].

Because it is generally undecidable whether a test goal is satisfiable on an arbitrary given program, we use an overapproximating reachability analyses, more specifically, a CEGAR-based predicate abstraction [BKW10], to approximate the set of executions of a program until we either (i) have found a partial program execution that is described by a word in the language of the test-goal or (ii) we have shown that there is no such execution. The test-goal automaton guides the reachability analysis, i.e., the analysis tracks program and automaton states simultaneously and stops exploring the state space if there is no possible transition in the program state space or no possible next automaton transition. Based on the excluded transitions  $X$ , we reuse parts of the already analyzed state space (those parts which do not involve these transitions) or continue state-space exploration along the transitions in  $X$ . We implemented our approach in the test-input generator CPA/TIGER<sup>1</sup>.

## References

- [BCH<sup>+</sup>04a] D. Beyer, A. J. Chlipala, T. A. Henzinger, R. Jhala, and R. Majumdar. Generating Tests from Counterexamples. In *Proc. ICSE*, pages 326–335. IEEE, 2004.
- [BCH<sup>+</sup>04b] D. Beyer, A. J. Chlipala, T. A. Henzinger, R. Jhala, and R. Majumdar. The BLAST Query Language for Software Verification. In *Proc. SAS*, LNCS 3148, pages 2–18. Springer, 2004.
- [BHTV13] D. Beyer, A. Holzer, M. Tautschnig, and H. Veith. Information Reuse for Multi-goal Reachability Analyses. In *Proc. ESOP*, pages 472–491, 2013.
- [BKW10] D. Beyer, M. E. Keremoglu, and P. Wendler. Predicate Abstraction with Adjustable-block Encoding. In *Proc. FMCAD*, pages 189–198. FMCAD Inc, 2010.
- [Hol13] A. Holzer. *Query-Based Test-Case Generation*. PhD thesis, TU Vienna, 2013.
- [HSTV08] A. Holzer, C. Schallhart, M. Tautschnig, and H. Veith. FSHELL: Systematic Test Case Generation for Dynamic Analysis and Measurement. In *Proc. CAV*, pages 209–213. Springer, 2008.
- [HSTV09] A. Holzer, C. Schallhart, M. Tautschnig, and H. Veith. Query-Driven Program Testing. In *Proc. VMCAI*, pages 151–166. Springer, 2009.
- [HSTV10] A. Holzer, C. Schallhart, M. Tautschnig, and H. Veith. How did You Specify Your Test Suite. In *Proc. ASE*, pages 407–416. ACM, 2010.
- [HTSV10] A. Holzer, M. Tautschnig, C. Schallhart, and H. Veith. An Introduction to Test Specification in FQL. In *Proc. HVC*, pages 9–22. Springer, 2010.

---

<sup>1</sup><http://forsyte.at/software/cpatiger/>

# Supporting Swift Reaction: Automatically Uncovering Performance Problems by Systematic Experiments

Alexander Wert, Jens Happe, Lucia Happe

Karlsruhe Institute of Technology  
Software Design and Quality  
Am Fasanengarten 5  
Karlsruhe, Germany  
alexander.wert@kit.edu  
jens.happe@kit.edu  
lucia.kapova@kit.edu

**Abstract:** Performance problems pose a significant risk to software vendors. If left undetected, they can lead to lost customers, increased operational costs, and damaged reputation. Despite all efforts, software engineers cannot fully prevent performance problems being introduced into an application. Detecting and resolving such problems as early as possible with minimal effort is still an open challenge in software performance engineering. In this paper, we present a novel approach for Performance Problem Diagnostics (PPD) that systematically searches for well-known performance problems (also called performance antipatterns) within an application. PPD automatically isolates the problems root cause, hence facilitating problem solving. We applied PPD to a well established transactional web e-Commerce benchmark (TPC-W) in two deployment scenarios. PPD automatically identified four performance problems in the benchmark implementation and its deployment environment. By fixing the problems, we increased the maximum throughput of the benchmark from 1800 requests per second to more than 3500.

## 1 Automated Performance Problem Diagnostics

In the paper [WHH13], we introduce a novel Performance Problem Diagnostics (PPD) that automatically identifies performance problems in an application and diagnoses their root causes. Once software engineers specified a usage profile for their application and setup a test system, PPD can automatically search for known performance problems. Since PPD encapsulates knowledge about typical performance problems, only little performance engineering expertise is required for its usage. PPD combines search techniques that narrow down the scope of the problem based on a decision tree with systematic experiments. The combination of both allows efficiently uncovering performance problems and their root causes that are otherwise hard to tackle. In its current state, PPD is tailored for the diagnosis of performance problems in Java-based three-tier enterprise applications. Overall, we make the following contributions:

- 1) We introduce a novel approach for performance problem detection and root cause anal-

ysis called Performance Problem Diagnostics. PPD systematically searches for known performance problems in three- tier enterprise applications. Once a problem has been found, PPD isolates its root causes as far as possible.

2) We structure a large set of known performance problems in a novel Performance Problem Hierarchy. To guide PPDs search, the hierarchy starts from very general problems (or symptoms). Each further level refines the problems down to root causes. The hierarchy allows systematically excluding classes of problems and focusing on the most relevant ones.

3) We define detection strategies for twelve performance problems in the hierarchy. The strategies are based on goal-oriented experiments tailored to trigger a specific problem. Based on the results, heuristics can decide if a problem is assumed to be present and refine the search. For each performance problem, we investigated and compared different heuristics for detecting the problems. We chose those heuristics that minimize false positives and false negatives.

4) We evaluated our approach in two steps. First, we determined the detection strategies that are most likely to find a performance problem. For this purpose, we evaluated the accuracy of each detection strategy based on ten reference scenarios. Each scenario contains different performance problems which have been injected into a test application. Second, we evaluated if PPD can detect performance problems in real enterprise applications.

## 2 Summary

We used the TPC-W Benchmark for evaluation of our approach. TPC-W is an official benchmark to measure the performance of web servers and databases. Thus, we expect it to be tailored for high performance. Finding performance problems there (if any) is especially challenging (and interesting). PPD identified four performance problems and isolated their root cause by systematically narrowing down the search space. In the initial setup, the size of the database connection pool, its default implementation, the network bandwidth, and the storage engine of the database limit the maximal throughput to 1800 req/s. Solving these problems increased TPC-Ws maximal throughput to more than 3500 req/s. Based on these promising results, we can state that our approach can diagnose performance problems in real applications and detect their root cause. PPD allows software engineers to automatically search for performance problems in an application with relatively low effort. Lowering the burden of performance validation enables more regular and more sophisticated analyses. Performance validation can be executed early and on a regular basis, for example, in combination with continuous integration tests.

## References

- [WHH13] Alexander Wert, Jens Happe, and Lucia Happe. Supporting swift reaction: automatically uncovering performance problems by systematic experiments. In *Proceedings of the 2013 International Conference on Software Engineering, ICSE '13*, pages 552–561, Piscataway, NJ, USA, 2013. IEEE Press.

# Concolic Testing of Concurrent Programs\*

Azadeh Farzan<sup>1</sup>, Andreas Holzer<sup>2</sup>, Niloofar Razavi<sup>1</sup>, Helmut Veith<sup>2</sup>

<sup>1</sup> Computer Science Department  
University of Toronto  
40 St. George Street  
Toronto, Ontario M5S 2E4, Canada

<sup>2</sup> Vienna University of Technology  
Institut für Informationssysteme 184/4  
Favoritenstraße 9-11  
A-1040 Vienna, Austria

**Abstract:** We describe (con)<sup>2</sup>colic testing — a systematic testing approach for concurrent software. Based on *concrete* and *symbolic* executions of a *concurrent* program, (con)<sup>2</sup>colic testing derives inputs and schedules such that the execution space of the program under investigation is systematically explored. We introduce *interference scenarios* as key concept in (con)<sup>2</sup>colic testing. Interference scenarios capture the flow of data among different threads and enable a unified representation of path and interference constraints.

## Overview

White-box testing concurrent programs has been a very active area of research in recent years. To alleviate the interleaving explosion problem that is inherent in the analysis of concurrent programs a wide range of *heuristic-based* techniques have been developed. Most of these techniques [WKG09, SFM10, SA06, RIKG12] do not provide meaningful coverage guarantees, i.e., a precise notion of what tests cover. Other such techniques [MQB07] provide coverage guarantees only over the space of interleavings by fixing the input values during the testing process. *Sequentialization* techniques [LR09] translate a concurrent program to a sequential program that has the same behavior (up to a certain context bound), and then perform a *complete static symbolic* exploration of both input and interleaving spaces of the sequential program for the property of interest. However, the sequential programs generated are not appropriate models for dynamic test generation due to the nondeterminism they involve. Recently, dynamic test generation was applied to sequentialized programs [RFH12]. Yet, this approach lacks completeness.

We propose (con)<sup>2</sup>colic testing, a new and systematic approach for the systematic exploration of both input and interleaving spaces of concurrent programs. (Con)<sup>2</sup>colic testing can provide meaningful coverage guarantees during and after the testing process. (Con)<sup>2</sup>colic testing can be viewed as a generalization of sequential concolic (*concrete* and *symbolic*) testing [GKS05] to concurrent programs that aims to achieve maximal code coverage for the programs. (Con)<sup>2</sup>colic testing exploits *interferences* among threads. An interference occurs when a thread reads a value that is generated by another thread. We introduce the new concept of *interference scenario* as a representation of a set of interferences among threads. Conceptually, interference scenarios describe the prefix of a concurrent program run such that all program runs with the same interference scenario

---

\*This is a summary of [FHRV13]. This work was supported by the Canadian NSERC Discovery Grant, the Vienna Science and Technology Fund (WWTF) grant PROSEED, and the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF).

follow the same control flow during execution of that prefix. By systematically enumerating interference scenarios,  $(\text{con})^2\text{colic}$  testing explores the input and scheduling space of a concurrent program to generate tests (i.e., input values and a schedule) that cover a previously uncovered part of the program.

Our  $(\text{con})^2\text{colic}$  testing framework has four main components: **(1)** A *concolic execution engine* executes the concurrent program according to a given input vector and schedule. The program is instrumented such that, during the execution, all important events are recorded. This information is used to generate further interference scenarios. **(2)** A *path exploration component* decides what *new* scenario to try next, aiming at covering previously uncovered parts of the program. **(3)** A *realizability checker* checks for the realizability of the interference scenario provided by the path exploration component. Based on this interference scenario it extracts two constraint systems (one for the input values and one for the schedule) and checks for the satisfiability of them. If both are satisfiable, then the generated input vector and the schedule are used in the next round of concolic execution. **(4)** An *interference exploration component* extends unrealizable interference scenarios by introducing new interferences.  $(\text{Con})^2\text{colic}$  testing can be instantiated with different search strategies to explore the interference scenario space.

To evaluate our approach we have implemented the tool CONCREST<sup>1</sup> [FHRV13]. It supports multi-threaded C programs and uses a search strategy that targets assertion violations and explores interference scenarios according to the number of interferences in an ascending order. This exploration strategy is complete modulo the explored interference bound and produces minimal error traces (wrt. the number of interferences).

## References

- [FHRV13] A. Farzan, A. Holzer, N. Razavi, and H. Veith. Con2colic testing. In *Proc. ESEC/SIGSOFT FSE*, pages 37–47, 2013.
- [GKS05] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed Automated Random Testing. In *Proc. PLDI*, pages 213–223. ACM, 2005.
- [LR09] A. Lal and T. Reps. Reducing Concurrent Analysis Under a Context Bound to Sequential Analysis. *Formal Methods in System Design*, 35:73–97, 2009.
- [MQB07] M. Musuvathi, S. Qadeer, and T. Ball. CHES: A Systematic Testing Tool for Concurrent Software, 2007.
- [RFH12] N. Razavi, A. Farzan, and A. Holzer. Bounded-Interference Sequentialization for Testing Concurrent Programs. In *ISoLA*, pages 372–387, 2012.
- [RIKG12] N. Razavi, F. Ivancic, V. Kahlon, and A. Gupta. Concurrent Test Generation Using Concolic Multi-trace Analysis. In *Proc. APLAS*, pages 239–255. Springer, 2012.
- [SA06] K. Sen and G. Agha. CUTE and jCUTE: concolic unit testing and explicit path model-checking tools. In *CAV*, pages 419–423, 2006.
- [SFM10] F. Sorrentino, A. Farzan, and P. Madhusudan. PENELOPE: Weaving Threads to Expose Atomicity Violations. In *Proc. FSE*, pages 37–46. ACM, 2010.
- [WKG09] C. Wang, S. Kundu, M. K. Ganai, and A. Gupta. Symbolic Predictive Analysis for Concurrent Programs. In *Proc. FM*, pages 256–272. Springer, 2009.

---

<sup>1</sup><http://forsyte.at/software/concrest/>

SE | 14  
SOFTWARE ENGINEERING

## **Software Engineering Ideen**



# **Vorwort Software Engineering Ideen Programm der SE 2014**

Das Ziel des Software Engineering Ideen-Programms war es, ein Forum für die Präsentation von Ideen im Bereich der Softwaretechnik bereitzustellen, die noch nicht den Reifegrad von wissenschaftlichen Konferenz-Beiträgen haben, aber eine vielversprechende Innovation oder Idee aufgreifen. Dabei haben wir folgende Bereiche im Auge gehabt:

- Agendas für neue Forschungsbereiche
- Neuartige Plattformen, Frameworks oder Werkzeuge
- Systeme, die eine neuartige Technologie verwenden
- Neue Projektmanagement-Techniken
- Verlaufsberichte über Innovationsprojekte
- Forschungsideen, die Zusammenarbeit und Synergie mit anderen Disziplinen erfordern

Bei den Beiträgen haben wir die Autoren ermutigt, Ideen einzureichen, die noch nicht vollständig implementiert oder noch nicht evaluiert sind. Insgesamt wurden 13 Beiträge eingereicht, von denen das Programm Komitee 6 akzeptiert hat. Die akzeptierten Beiträge sind in diesem Tagungsband der SE 2014 aufgenommen worden. Als Mitglieder des Programm Komitee wirkten mit:

- Bernd Brügge, TU München (Vorsitz)
- Stepanie Balzer, Carnegie Mellon
- Oliver Creighton, Siemens AG
- Michael Goedicke, Uni Duisburg-Essen
- Martin Glinz, Uni Zürich
- Volker Gruhn, Uni Duisburg-Essen
- Wilhelm Hasselbring, Uni Kiel
- Robert Hirschfeld, HPI Potsdam



- Florian Matthes, TU München
- Christoph Peylo, Trust2Core
- Dirk Riehle, Uni Erlangen-Nürnberg
- Kurt Schneider, Leibnitz Universität, Hannover
- Stephan Verclas, T-Systems
- Markus Voß, Accso Accelerated Solutions GmbH, Darmstadt

Wir danken den Mitgliedern fuer die lebhaften Diskussionen, die auf Grund der Vagheit des Aufrufes nicht unerwartet sehr lebhaft und kontrovers ausfielen. Herzlichen Dank an die zusätzlichen Reviewer Tim Felgentreff, Helmut Naughton, Michael Perscheid und Marcel Täumel. Wir hoffen, dass auch die Präsentationen der Beiträge zu lebhaften Diskussionen führen.

Bernd Brügge  
Technische Universität München  
SEI Tagungsleitung

# Sozial empathische Systeme coram publico

Anke Tallig

Fakultät für Informatik  
Technische Universität Chemnitz  
Straße der Nationen 62  
09111 Chemnitz  
anke.tallig@informatik.tu-chemnitz.de

**Abstract:** Dieses Paper beschreibt die Weiterentwicklung von öffentlichen interaktiven Systemen zu sozial empathischen Systemen. Als öffentliche interaktive Systeme werden in diesem Zusammenhang allgemein alle zugänglichen Informationssysteme coram publico verstanden, mit denen der Nutzer auf verschiedene Weise interagieren kann. Im Rahmen des interdisziplinären Graduiertenkollegs „*CrossWorlds – Connecting Virtual and Real Social Worlds*“ wird, durch die Erkennung der sozialen Signale der Nutzer, ein sozial empathisches System entwickelt. Dieses System ist ein interaktiver Public Screen mit einer *Edutainment*-Oberfläche. Das Paper zeigt Punkte auf, die aus der Mensch-Mensch-Interaktion für eine soziale Empathie softwaretechnisch umgesetzt werden können. Ebenso wird ein Vorgehensmodell der Softwareentwicklung vorgestellt, welches während der Zusammenarbeit von Ingenieurin und Sozialwissenschaftler entwickelt wurde und an die Bedingungen und Bedürfnisse der interdisziplinären Arbeit angepasst wurde. Ziel der Entwicklung sozial empathischer Systeme ist der leichtere Umgang mit diesen Systemen, damit sich nicht nur der Mensch auf das System einstellen muss, sondern das System sich auch auf seine(n) Menschen einstellen bzw. sein Angebot anpassen kann, was gleichzeitig die Wirtschaftlichkeit und die soziale Akzeptanz der Systeme erhöht.

## 1 Die Idee

Von der Idee Systeme zu entwickeln, die die Nutzer verstehen und nicht nur vom Nutzer verstanden werden müssen, träumen die Menschen seitdem sie die Entwicklung vorantreiben. Maschinen zu entwickeln, die nicht nur ihren Zweck erfüllen, wenn der Nutzer den richtigen Knopf drückt, sondern Maschinen, die angepasst an den jeweiligen Nutzer nur bestimmte Knöpfe zur Verfügung stellen, um die Nutzung einfacher und schneller zu gestalten. Interaktive Systeme, die ihre Arbeit aufnehmen, bevor der Nutzer den Prozess aktivieren muss. „Bitte berühren Sie den Bildschirm“ – warum? Damit das System aus dem Ruhemodus erweckt wird und zur Arbeit zur Verfügung steht? Warum sieht das System den Nutzer nicht kommen und bootet aus dem Ruhemodus, damit der Nutzer keine Wartezeiten hat oder womöglich denkt das System sei defekt. Warum lädt das System den Nutzer nicht ein, sich mit den Inhalten zu beschäftigen? Warum muss der Nutzer das System „bitten“ seine Informationen preiszugeben? Interaktive Informationssysteme sollen das Leben erleichtern. Aber Interaktion fordert nicht nur

vom Menschen, dem Nutzer, sich auf das System einzustellen und zu verstehen, was in diesem vorgeht, auch das System sollte sich auf seinen Interaktionspartner einstellen können, um einen reibungslosen Austausch von Informationen zu gewährleisten.

Für eine softwaretechnische Lösung von Nutzerproblemen im Umgang mit technischen Systemen muss versucht werden, eine, in den meisten Fällen, funktionierende *Mensch-Mensch-Interaktion* zu analysieren und die auf die *Mensch-Computer-Interaktion* übertragbaren Faktoren für das System zu implementieren.

Initiiert durch die *menschliche Interaktion*, die die sozialen Signale, die der Mensch aussendet, nutzt, um eine angepasste und soziale Kommunikation einzuleiten [ArM13], begann die Entwicklungsarbeit an einem sozial empathischen interaktiven System. Dieses prototypische System entsteht im Rahmen des Graduiertenkollegs „*CrossWorlds – Connecting Virtual and Real Social Worlds*“. Genau, wie sich Menschen in ihrer Kommunikation auf die *non-verbalen Signale* ihres Gegenübers verlassen, sollte dies ein intelligentes interaktives System auch können. Ziel des Projektes ist es, zu beginnen, diese Fähigkeiten zu implementieren.

Da die Erkennung von sozialen Signalen in erster Linie keine technische Fragestellung ist, muss die Entwicklung einer erfolgreichen Erkennungssoftware (*Punkt 2*) in interdisziplinärer Zusammenarbeit erfolgen. Die Zusammenarbeit von Ingenieurwissenschaft und Sozialwissenschaft, wie auch in [PeM94, StJ98] beschrieben, gestaltet sich aufgrund der unterschiedlichen Vorgehensweisen der Wissenschaften problematisch. Die Entwicklung eines interdisziplinären Vorgehensmodells (*Punkt 2.1*) für die spezielle Zusammenarbeit ist für das Erreichen der Ziele unumgänglich. Denn erst die Anpassung der Softwareentwicklung an den realen Nutzer lässt den Transfer der Erkenntnisse aus der Wissenschaft in die Wirtschaft zu.

## 2 Die Grundlagen

Über die Grundlagen des nonverbalen Ausdrucks [ArM13] lassen sich die Vorteile der Anwendung in interaktiven Systemen ableiten. Die menschliche Wahrnehmung des Gegenübers findet in drei Phasen statt: Die *erste* Wahrnehmungsphase erfolgt in der Entfernung. Der mögliche Interaktionspartner wird beobachtet und es wird eingeschätzt, was seine Intension an diesem Ort ist bzw. sein könnte. Die *zweite* Phase beschreibt das Näherkommen, in dem das Bild, das sich der Mensch von seinem möglichen Interaktionspartner macht, verfeinert wird. Die *dritte* Phase ist das Gespräch oder die Interaktion an sich.

Ingenieure und Informatiker entwickeln technische Systeme mit immer umfangreicheren Funktionen und nutzerfreundlicheren Interfaces, die den Nutzer unterstützen und ihm weiterhelfen sollen. Aber das System nimmt den Menschen nicht als soziales Wesen war. Dem Nutzer werden verschiedene interaktive Systeme, z. B. Großbildschirme [DMi09], im öffentlichen Raum angeboten. Diese Public Screens spiegeln den Nutzer, damit er sich auf dem Bildschirm und damit dessen Interaktivität erkennt. Einige Systeme geben ab und an Geräusche von sich, um auf sich aufmerksam zu machen.

Durch z. B. Blickerkennung und –verfolgung versuchen die Hersteller von Mobilgeräten [AI13], den Nutzern den Umgang zu erleichtern.

Aber all diese Implementierungen betreffen die *dritte* Wahrnehmungsphase in der menschlichen Interaktion. Diese softwaretechnischen Umsetzungen überspringen die ersten beiden Schritte der Interaktion und können damit nicht ausreichend auf ihre Nutzer eingehen. Sie verschenken die Möglichkeit, den Nutzer besser kennenzulernen bevor die eigentliche Interaktion stattfindet. Die (soziale) Einschätzung des Partners ist ein wichtiger Punkt für Interaktion von Menschen und kann im Umgang mit interaktiven Systemen eine herausragende Rolle spielen, was nicht nur die allgemeine Akzeptanz der Systeme, sondern durch umfangreichere und gezieltere Nutzung auch deren Wirtschaftlichkeit erhöht.

## 2.1 Nonverbale Signale

Nonverbale Signale sind unter anderem die Körperhaltung, das Raumverhalten, Körperbewegung und der Gesichtsausdruck [ArM13]. Werden diese Signale wahrgenommen, kann eine softwareseitige Verarbeitung vorgenommen werden, die der des Menschen gleicht. Die Verarbeitung dieser nonverbalen Signale ist beim Menschen weitaus komplexer als es zurzeit softwaretechnisch realisierbar ist. Die Kommunikation ist ein Kreislauf, der keinen Anfang und kein Ende kennt, jede Aktion ist Reaktion und Reaktion eine Aktion [WaP11]. Sich diesen Kreislauf zum Nutzen zu machen und den fortwährend kommunizierenden Menschen, den möglichen Nutzer, wahrzunehmen und ihm so zu helfen, ist ein erreichbares Ziel. Der erste Schritt ist es, die menschlichen Wahrnehmungsphasen in ihren Grundzügen zu analysieren. Im Weiteren müssen die vorerst ausgewählten Signale extrahiert und für diese eine oder mehrere Deutung/en hinterlegt werden, um diese dem System Stück für Stück zuzuführen.

Es gibt diverse Untersuchungen, die sich mit der Analyse von menschlichem Verhalten und zwischenmenschlichem Verhalten im sozialen Raum beschäftigen. Die F-Formations (facing formations) [MaP11] sind zum Beispiel Ergebnis solcher Untersuchungen. Hier wird dargestellt, in welcher Entfernung und in welchem Winkel zwei oder mehr Menschen zueinanderstehen. Dies lässt wiederum Rückschlüsse auf deren Beziehung zueinander und unter Umständen ihre Offenheit für andere mögliche Interaktionspartner zu. Zusätzlich wurden diese F-Formations für den Kontakt mit öffentlichen und mobilen Systemen erweitert [DiE13, MaN12]. Darüber hinaus wurden Schemen für die Annäherung von möglichen Nutzern an öffentliche Bildschirme entwickelt [MiD11]. Diese Annäherungsphasen sind für die Entwicklung einer Erkennungssoftware von großem Nutzen. In Verbindung mit den Erkenntnissen, die menschliche Körpersprache betreffend, ist die Erkennungssoftware theoretisch realisierbar und der praktischen Umsetzung stehen, momentan noch, zum einen die Rechenkapazität und zum anderen die präzise Deutung der menschlichen sozialen Signale im Weg.

Auf Basis der Forschungen und auf ersten Erfahrungen mit dem Prototyp ist es möglich eine Software zu entwickeln, die die User-Konstellationen und deren sozialen Kontext wahrnimmt. Das Problem, welches oft auch Fehlverhalten in der Mensch-Mensch-

Interaktion hervorruft, ist die Interpretation der menschlichen sozialen Signale. An dieser Stelle ist die Zusammenarbeit mit Sozialwissenschaftlern unerlässlich, die den sozialwissenschaftlich notwendigen Hintergrund beleuchten. Durch diverse Nutzerstudien, die die Motivation und die Wahrnehmung der Probanden betreffen, können Fehler in der Implementierung der Wahrnehmung und Interpretation vermieden werden. Oft wird von der informatischen Seite zu Testzwecken ein Testszenario entworfen, welches auf lückenhaftem Wissen in dieser Wissenschaftsrichtung beruht. Im Rahmen der Zusammenarbeit werden die Ergebnisse der sozialwissenschaftlichen Studien einer technischen Machbarkeitsprüfung durch die Ingenieure unterzogen und somit kann die Umsetzung der sozialen Wahrnehmung eingeschränkt oder angepasst werden und eine bessere Deutung der nonverbalen Kommunikation stattfinden.

Das Modell in Abbildung 1 stellt die Phasen der Zusammenarbeit von Ingenieurwissenschaft und Sozialwissenschaft dar und ist angelehnt an das V-Modell von Boehm [BoB79].

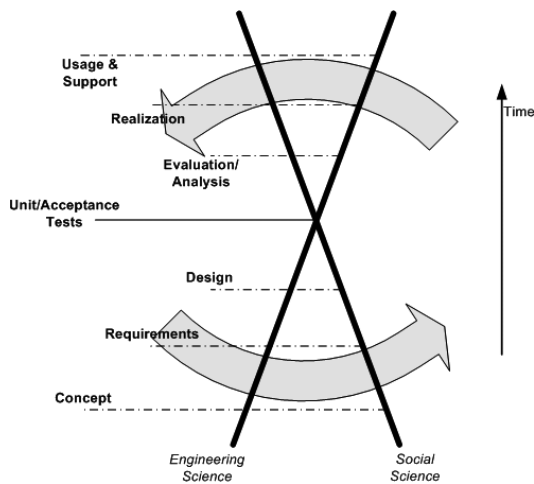


Abbildung 1: Interdisziplinäres CHI-Modell

Die zwei sich kreuzenden Stränge stellen die beiden Wissenschaften dar. Vertikal ist die Zeit und horizontal sind die sieben Phasen der *gemeinsamen* Arbeit dargestellt, die parallel von beiden Wissenschaften ausgeführt werden. Die sich kreuzenden Stränge zeigen, dass die Wissenschaften von unterschiedlichen Ausgangssituationen ausgehen und sich auch die Endpunkte der wissenschaftlichen Arbeit unterschiedlich gestalten. Eine Zusammenarbeit soll keine „Hybridwissenschaft“ erschaffen, beide Wissenschaften sollen ihre Entwicklung in Reinform betreiben. Dazu gehört, dass beide Wissenschaften ihre Basis behalten: Die Ingenieurwissenschaft sieht die Technik und diverse Möglichkeiten und Szenarien, die realisiert werden können, die Sozialwissenschaft sieht den Menschen im Mittelpunkt und die Möglichkeiten, ihm den Umgang mit der Technik zu erleichtern. Durch die Zusammenführung der Erkenntnisse können die nächsten Phasen geplant und abgestimmt werden und so zu einem optimalen Ergebnis führen. Ohne eine längerfristige gemeinsame interdisziplinäre Zusammenarbeit kann eine Wahrnehmung und Interpretation der sozialen Signale nicht eindeutig realisiert werden.

## 2.2 Softwaretechnische Umsetzung

Der Prototyp der Erkennungskomponente arbeitet mit einer HD-Kamera und Infrarotsensoren (IR-Sensoren). Um den Platz der Systeme nicht zu strapazieren, habe ich mich für die raumsparende Umsetzung mit einer Kamera, einem IR-Projektor und einer IR-Kamera entschieden. Wobei eine Variante welche wenigsten zwei Seiten überwacht mehr und zuverlässigere Daten geliefert hätte. Da ein solcher Aufbau aber selten in öffentlichen Räumen realisierbar ist, entschieden wir uns dagegen. Der erste Schritt ist die Festlegung des Aktionsbereiches: Der überwachte Interaktionsbereich von 1 m bis 3,5 m folgt den Beobachtungen HALLS [HE82], der, aufgrund seiner Studien, diesen Abstand zum möglichen Gesprächspartner als Interaktionsbereich festlegt. Im zweiten Schritt der Bilderkennung folgt die Wahrnehmung des Menschen an sich. Wird kein Mensch bzw. keine menschliche Statur erkannt, arbeitet die Bilderkennung nicht weiter, weil kein potenzieller Interaktionspartner in Reichweite ist. Erkennt das System einen Interaktionspartner, wird in der aktuellen Version des Prototypen, eine verbale Meldung an den möglichen User gerichtet. Diese Variante wurde gewählt, um zum einen die sozialwissenschaftlichen Studien zu realisieren und zum anderen die Aufmerksamkeit der Nutzer auf das System zu lenken. Daraus ergeben sich Vorteile, wie z. B.: Die Probanden können direkt befragt werden, wie „diese Art System“ auf sie wirkt und was sie denken, was es verfolgt (quantitative Sozialforschung), zusätzlich zur Beobachtung der Probanden (qualitative Sozialforschung). Der Vorteil der Aufmerksamkeitslenkung ist, der potenzielle Nutzer steht frontal zum System und bietet somit der Bilderkennung eine bessere Chance, den Nutzer weiter zu analysieren.

Für diese Analyse wird der Körper und seine Position im Raum bzw. die Position seiner Gliedmaßen zum Körper und der Gliedmaßen zueinander vermessen. Diese berechneten Abstände und Winkel werden mit hinterlegten Pattern verglichen. Die Interpretation, die das System vornimmt und die entsprechenden Ausgaben, erfolgen auf Grundlage der Ergebnisse der sozialwissenschaftlichen Studien. Wie weit die Erkennung mit diesem einfachen System und die Interpretation von sozialen Signalen gehen kann und wie hoch die Akzeptanz in den unterschiedlichen Erkennungsstadien ist, wird zurzeit noch getestet, ebenso wird an der Robustheit der Erkennung gearbeitet.

## 3 Die Zusammenfassung und Aussicht

Sozial empathische Systeme sollen Menschen aktiv ansprechen, die als mögliche Nutzer identifiziert werden. Damit soll das System eine höhere Nutzungsfrequenz erreichen. Außerdem wird durch die Erkennung der sozialen Signale ein angepasster und dadurch individueller Informationsaustausch ermöglicht. Durch das Einbeziehen der Informationen, die der Nutzer durch seine sozialen Signale vermittelt und ein dadurch erreichtes einfacheres Handling sowie eine bessere Datenbeschaffung kann der Nutzungsdurchsatz erhöht werden und damit die Wirtschaftlichkeit. Das aktive Einbeziehen der Nutzer verringert auch die beim Menschen vorhanden Hemmschwellen und öffnet somit auch technisch nicht versierten Menschen den Zugang zu interaktiven Systemen. Ziel der Entwicklung sind aufmerksame interaktive Systeme, die den potenziellen Nutzer aus der Ferne als diesen identifizieren und analysieren, um eine

individuelle Interaktion zu initiieren, den Nutzungsdurchsatz und die Wirtschaftlichkeit zu erhöhen.

„*Wissenschaft meets Wirtschaft*“ an dem Punkt, an dem die Analyse der sozialen Signale der Nutzer erfolgt ist und diese prototypisch implementiert sind, sowie umfangreiche Tests absolviert worden sind, deren Ergebnisse in die Weiterentwicklung eingegangen sind. Ist eine robuste Erkennung implementiert worden, kann der Schritt in die Wirtschaft erfolgen. Nach der gemeinsamen Anpassung an die Situation und die Bedürfnisse kann das System im Realraum eingesetzt werden. Die Inbetriebnahme und die ersten Erfahrungen mit dem System und der Erkennungssoftware bieten wiederum Ergebnisse, die in der Wissenschaft für die Weiterentwicklung und Neuentwicklung Verwendung finden können. Die Akzeptanz der Nutzer und die Wirtschaftlichkeit sind Ziele, die mit der Realisierung der Erkennung von sozialen Signalen erreicht werden können und dadurch für beide Seiten einen Vorteil bieten. Durch die sukzessive Einführung der neuen Eigenschaft ist es möglich, eine längerfristige Zusammenarbeit von Wissenschaft und Wirtschaft zu realisieren. Somit kann eine Interaktion entstehen, die durch einen ausgeglichenen „Wissensstand“ von Mensch und Maschine von beiden Seiten individuell gestaltet wird.

## Literaturverzeichnis

- [AI13] Champbell, M: AppleInsider; <http://appleinsider.com/articles/13/05/30/apple-shows-renewed-interest-in-gaze-detection-possibly-in-response-to-samsungs-smart-scroll> (last access: 2013-10-04).
- [ArM13] Argyle, M.: Körpersprache & Kommunikation, Nonverbaler Ausdruck und Soziale Interaktion. Paderborn: Junfermann Verlag 2013.
- [BoB79] Boehm, B. W.: Guidelines for verifying and validating software requirements and design specifications. Euro IFIP 1979. North Holland 1979.
- [DiE13] Dim, E. & Kuflik, T.: Social F-formation in Blended Reality. LAMBDa 2013, March 19-22, Santa Monica, CA, USA.
- [HE56] Hall, E. T.: The Hidden Dimension. Massachusetts Institute of Technology, 1956.
- [MaN12] Marquardt, N.: Cross-Device Interaction via Micro-mobility and F-Formations. UIST 2012, October 7-12, Cambridge, Massachusetts, USA.
- [MaP11] Marshall, P.: Using F-Formations to Analyse Spatial Patterns of Interaction in Physical Environments. Proceedings of the ACM 2011, New York.
- [MiD11] Michelis, D. & Müller, J.: The Audience Funnel: Observations of Gesture Based Interaction With Multiple Large Displays in a City Center. In: International Journal of Human-Computer Interaction. Vol. 27, Nr. 1, 2011.
- [MiD09] Michelis, D.: Interaktive Großbildschirme im öffentlichen Raum, Springer, 2009.
- [PeM94] Paetau, M.; Rohde, M. & Wulf, V.: Das Maschinenmodell wird zum Auslaufmodell. In: Wechselwirkung, Zeitschrift für Technik, Naturwissenschaft, Gesellschaft, Jg.16, Nr. 69.
- [StJ98] Strübing, J.: Vom Nutzen der Mavericks, Zur Zusammenarbeit von Informatik und Soziologie auf dem Gebiet der Verteilten KI. KI-Tagung Bremen 1998.
- [TA13] Tallig, A.: A Robot Companion as mobile Edutainer. In: International Summerworkshop Computer Science 2013 Proceedings of International Summerworkshop 17.7. – 19.7.2013. Chemnitzer Science-Report CSR-13-04, July 2013.
- [TA13a] Tallig, A.: Creating an interactive Mixed Reality. HCI International 2014 (in progress).
- [WaP11] Watzlawick, P.: Menschliche Kommunikation. Bern: Hans Huber Verlag 2011.

# The Usability Engineering Repository (UsER)

Marc Paul, Amelie Roenspieß, Tilo Mentler, Michael Herczeg

Institut für Multimediale und Interaktive Systeme (IMIS)

Universität zu Lübeck

Ratzeburger Allee 160

23562 Lübeck

paul@imis.uni-luebeck.de

roenspiess@imis.uni-luebeck.de

mentler@imis.uni-luebeck.de

herczeg@imis.uni-luebeck.de

**Abstract:** The Usability Engineering Repository (UsER) is a tool system designed and implemented in prototypical state to support the analysis, design and evaluation of interactive systems. For this purpose, UsER provides method modules covering different aspects of analysis, design, implementation and evaluation. While created in a linear, document-like structure, contents can be interlinked in order to point out semantic interrelationships, ease navigation and allow tracing requirements forward and backward in the process. The modules can be configured and used as needed and none of them is mandatory. We will introduce UsER using the Human-Centered Design (HCD) Process as example.

## 1 Introduction

Various development processes for software engineering exist. However, the user-centered analysis, design and evaluation of interactive systems are deficiently taken into account in many projects. Either the process model does not support considering users, tasks and overall context or integrated tooling is missing. We combined several approaches in the areas of Software Systems Engineering, Requirements Engineering and especially Usability Engineering to support the whole process of development. The current result is an analysis and modelling environment as development repository called UsER (Usability Engineering Repository). It allows to flexibly combine different tools and methods depending on project-specific needs as well as to trace and reuse all information raised during the development process. UsER supports the process of human-centered design (ISO 9241:210) according to the six key principles:

1. the design is based upon an explicit understanding and documentation of users, tasks and context;
2. stakeholders can be involved throughout design and development;
3. the design is driven and refined by user-centered formative evaluation;
4. the process is iterative;



5. the design addresses the whole user experience;
6. the design team provides multidisciplinary skills and perspectives.

Most software developing companies meanwhile recognize how important usable software is for their customers, but not how to reach this goal. Usability is not simply an additional requirement to be implemented at the end of the development. Usability Engineering provides a wide range of methods and systematic approaches for the support of development, for example the Goal-Directed-Design [CRC07], the Usability Engineering Lifecycle [Ma99] or the Human-Centered Design (HCD) Process [ISO11], to name but a few. All of these need to be applied throughout the whole development process. We decided to design and develop UsER as a modular tool environment so it can be adapted flexibly to any existing process.

## 2 UsER Modules in a Sample Process

To provide a systematic and predictable approach to software development, we support every step of the HCD process (as an example process) with certain modules (see Fig. 1). In order to benefit from this process, all of its results may be transferred to the implementation process. Requirements are particularly well suited for this task.

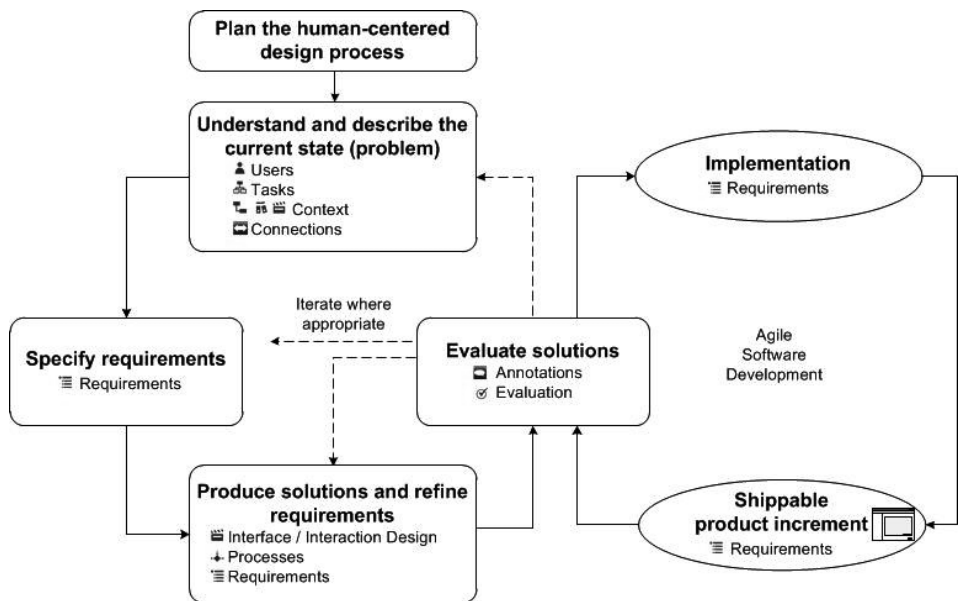


Figure 1: Extended Human-Centered Design Process

A UsER module supports certain aspects of a usability engineering method. Information gathered in one module can be interlinked with information in other modules. These references help to understand development decisions made during analysis, design or evaluation, as they can be retraced directly from the requirements concerned.

A development project in UsER is organized as a linear structure like a typical development document. Each chapter or section provides the functionality of a specific module (see Figure 2). These can be arranged via drag&drop to model contract and development documents. The resulting structure itself can be stored and later reused as a template for future projects. The modules realized so far support analysis and design as well as requirements management throughout the whole software development process.

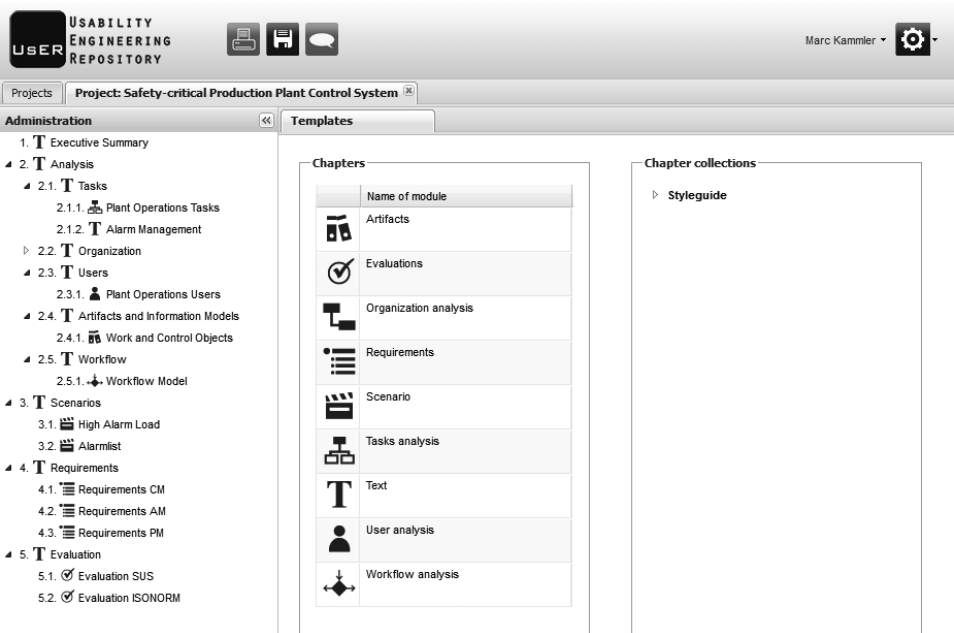


Figure 2: Selected combination of modules in UsER

## 2.1 Analysis of the Domain and the Users

Hereafter, we will have a look at the functionality of some modules following the approach of HCD.

### *User Analysis*

In order to understand the context in which application software is supposed to be used, it has to be analyzed together with the stakeholders. The first idea for this module was to create and manage personas [CRC07, PA06], which could then be reused in the other modules. During development we realized that it would be helpful and practical classifying user descriptions on different levels of detail [PA06]. User classes or less abstract stereotypes can now be created as organizational roles or target groups. The information collected in user classes can then be reused by inheritance and refined for more detailed user descriptions. Thus, user descriptions can be defined on an abstraction level according to the needs of the project [PA06]. Additionally, the module has a progress bar indicating roughly how much important information has already been

specified in the user description. Including field data can be important for user descriptions. UsER supports this by providing so called “behavioral variables” [CRC07]. These are user attitudes and aptitudes which are relevant for the project, e.g. whether the users prefer simple interfaces over customizable ones. The answers are mapped onto a bidirectional scale, visualizing trends and clusters which can be used to describe characteristics of fictional users based on field research data. The data might even be gathered semi-automated from an evaluation module. Specified for single or several projects, user descriptions also contain user goals, which can become requirements in order to be taken into account for the later specification. It is indispensable to determine and analyze the tasks future users are supposed to carry out with the system [Co99, He09]. The module for user analysis already allows task description on an abstract level, also called “external tasks” [He09] in operational contexts. These can be described in more detail in the module for task analysis, while the artifacts used can be specified in the module for artifact models.

### *Task Analysis*

Tasks may be decomposed into subtasks using the model of Hierarchical Task Analysis (HTA) [AD67]. The integrated list editor allows direct graphical manipulation of the tree representation as well as keyboard-only input. HTA is an effective tool for visualizing and understanding tasks [Be10] as it shows tasks in the context of their higher purpose. A task description contains some basic information like pre- and postconditions, duration, criticality etc. As mentioned before, artifacts (work objects) required for a task can be described in a module of their own. Additionally references to organizational roles may be established, which allow the description of team-based tasks and lead to the method called HTA-T (HTA for Teams). A UsER-wide component allows interlinking any entities of all modules.

### *Artifact Models*

From the user’s perspective, it often is not the application which is relevant but the work objects to be manipulated [PA06] like documents or physical work objects. UsER supports the modeling of artifacts with names, an optional description and arbitrarily configurable attributes. These can serve as a basis for information models or help to describe detailed context information by linking to other modules, e.g. for defining who in an organization works with which artifacts or for which task certain artifacts are required.

### *Organization Analysis*

Information about operational structures is often depicted in form of organization charts. The handling of this module is similar to existing applications like Microsoft Visio. Organization units, roles and staff positions can be described in this module. Every piece of information already collected – e.g. users or tasks – can be assigned to the elements of the organization chart to get a good overview and deeper understanding.

## *Scenarios*

Alternatively or additionally it is possible to describe the application in a more informal textual form. Inspired by the approach of Scenario-based Design by Rosson and Carroll [RC01], this module supports the description of problem scenarios. Furthermore, it allows to enrich every scenario by adding pre- and postconditions. All information collected so far (users, tasks, etc.) can be linked to these descriptions. This module is also suitable for the transition to the next step of the HCD process. As a rich text application the module allows to include graphics and other media.

### **2.2 Specifying Requirements**

For a comfortable transition to the next step of the HCD process there is an interface integrated in the scenario module for deriving requirements from each problem scenario. Here it is important to articulate these requirements in an abstract way like recommended by Goal-oriented Requirements Engineering [La00, La01, Po08]. All requirements gathered from the scenario module can be processed inside this module to create formal requirements. It will be possible to import or export requirements from or into other requirements engineering environments through the standardized ReqIF-format.

The requirements module collects all requirements within one project in a sortable table. It allows decomposing a requirement into sub-requirements. Furthermore, it is possible to add more information to the requirements, like source, reason, type (bug, new feature etc.) or a category to sort the requirements by system components.

### **2.3 Create Solutions and Refine Requirements**

The idea in this step of the HCD process is to refine the abstract requirements (goals) previously collected by describing the activities in a textual way. These textual descriptions (“design scenarios” [RC01]) work well for discussing solutions with the stakeholders. They are used to specify how a goal can be met [Ro98]. Because of the different ways imaginable for achieving a goal, it should be possible to describe whole use cases with a main case and alternative cases. The module we are using for this step is the scenario module already used for describing the problem situation at the beginning of the process. To attach several scenarios to one use case it is possible to include them in different chapters within a project. Scenarios can be interlinked using the hypertext function. For refining the scenarios continuously we integrated the external mockup tool Balsamiq® which supports the fast and flexible creation of low fidelity prototypes.

### **2.4 Evaluate the Solutions**

Two of the key principles of HCD are involving the user and evaluating the developed solutions together with them. To support this as well as an iterative refinement process we integrated an evaluation module which allows formative and summative evaluation of the software using standardized or specialized questionnaires together with an annotation

function which allows comments for every element of content. By this, solutions can be collaboratively refined and iterated. As the results of questionnaires are shown in UsER and the annotations are sorted chronological, decision processes can be retraced. When a solution finally meets the user requirements, the concepts can be transferred to the developers for implementation.

### 3 Summary and Future Work

The Usability Engineering Repository (UsER) is an integrated modular framework to support various user centered development methods and processes for interactive systems in a flexible way. Modules for user, task and organizational analysis can be combined with modules for scenarios, interaction design or evaluation. As an open architecture UsER can be extended by additional modules. For instance, we are currently working on tool support for the analysis, design and evaluation of safety-critical systems.

### References

- [AD67] Annett, J. and Duncan, K. Task analysis and Training Design. *Occupational Psychology*. 41, July, 1967, pp. 211–221.
- [Be10] Benyon, D. *Designing Interactive Systems: A Comprehensive Guide to HCI and Interaction Design*. Harlow, England: Pearson Education, 2010.
- [Co99] Constantine, L. *Software for use: A practical Guide to the Models and Methods of usage-centered Design*, 1999.
- [CRC07] Cooper, A., Reimann, R. and Cronin, D. *About Face 3: The Essentials of Interaction Design*, 2007.
- [He09] Herczeg, M. *Software-Ergonomie: Theorien, Modelle und Kriterien für gebrauchstaugliche interaktive Computersysteme*. München: Oldenbourg Wissenschaftsverlag, 2009.
- [ISO11] International Organization for Standardization. *ISO 9241-210:2010 Human-centred Design for Interactive Systems*, 2011.
- [La01] Lamsweerde, A. *Goal-Oriented Requirements Engineering : A Guided Tour*. *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, 2001.
- [La00] Lamsweerde, A. Handling Obstacles in goal-oriented Requirements Engineering. *IEEE Transactions on Software Engineering*, 10, 2000, pp. 978–1005.
- [Ma99] Mayhew, D.J. *The Usability Engineering Lifecycle : A Practitioner’s Handbook for User Interface Design*. San Francisco, CA: Morgan Kaufmann, 1999.
- [Po08] Pohl, K. *Requirements Engineering*. dpunkt.verlag, 2008.
- [PA06] Pruitt, J. and Adlin, T. *The Persona Lifecycle: Keeping People in Mind throughout Product Design*. San Francisco, CA: Morgan Kaufmann, 2006.
- [Ro98] Rolland, C. Guiding Goal Modeling using Scenarios. *IEEE Transactions on Software Engineering* 12, 1998, pp. 1055–1071.
- [RC01] Rosson, M. and Carroll, J.M. *Usability Engineering: Scenario-Based Development of Human-Computer Interaction*. San Francisco, CA: Morgan Kaufmann, 2001.

# Sicherstellung von Performanzeigenschaften durch kontinuierliche Performanztests mit dem KoPeMe Framework

David Georg Reichelt, Lars Braubach

davidgeorg\_reichelt@dagere.de, braubach@informatik.uni-hamburg.de

**Abstract:** Um garantieren zu können, dass Performanzanforderungen von einer Anwendung erfüllt werden, werden Modelle zur Performanzvorhersage am Anfang der Entwicklung und Lasttestwerkzeuge am Ende der Entwicklung eingesetzt. Verschiedene Schwächen dieser Ansätze machen eine neue Methode zur Sicherstellung von Performanzanforderungen notwendig. Die Idee dieses Beitrags ist es, Performanzeigenschaften durch kontinuierliche Tests sicherzustellen. In diesem Papier wird ein Werkzeug vorgestellt, das kontinuierliche Performanztests für Java-Programme ermöglicht. Es stellt eine Schnittstelle bereit, mit der Performanztests implementiert werden können und enthält Möglichkeiten zur Einbindung in Ant und Maven. Weiterhin existiert die Möglichkeit, die Performanzverläufe im Integrationsserver Jenkins zu visualisieren. Die Wirksamkeit des Ansatzes wird mit Hilfe eines Tests von Performanzeigenschaften der Revisionen des Tomcat-Servers durch das Performanztestwerkzeug überprüft.

## 1 Einleitung

Performanz ist eine zentrale Eigenschaft von Software. Ist sie unzureichend, kann dies zu wirtschaftlichen Einbußen führen. Dennoch ist Performanz im Vergleich zur funktionalen Korrektheit von Programmen ein wenig beachtetes Problem [Mol09, Seite 10].

Ein Ansatz, um Performanz sicherzustellen, ist der „Beheb - Es - Später“ (engl. „Fix-It-Later“) Ansatz: Zuerst wird funktionale Korrektheit sichergestellt und am Schluss der Entwicklung wird die Performanz betrachtet. Dies ist ineffizient, da zur Performanzverbesserung oft Architekturveränderungen notwendig sind, die am Ende der Entwicklung aufwändiger sind als am Anfang [BMIS03]. Daneben ist zu beobachten, dass Software immer öfter langlebig ist, d.h. Software wird lange weiterentwickelt und während dieser Entwicklung eingesetzt [Par94]. Aktuelle Vorgehensmodelle legen es ebenfalls nahe, schnell in der Praxis einsetzbare Programmteile zu schaffen und diese einzusetzen [BBvB<sup>+</sup>01]. Wenn Software parallel zur Entwicklung eingesetzt wird, ist es notwendig, kontinuierlich die Erfüllung von Anforderungen, insbesondere den Performanzanforderungen, zu prüfen.

Grundidee dieser Arbeit ist es deshalb, ein Werkzeug zur Unterstützung der kontinuierlichen Performanzbetrachtung zu schaffen. In Abschnitt 2 werden Anforderungen an dieses herausgearbeitet. Anschließend wird in Abschnitt 3 auf verwandte Arbeiten eingegangen. Danach wird in Abschnitt 4 die Umsetzung des Werkzeugs dargestellt. Ein Evaluationsansatz wird in Abschnitt 5 dargestellt. Abschließend wird eine Zusammenfassung gegeben.

## 2 Anforderungen

Die Basisanforderung an ein Performanztestwerkzeug ist es, die Ausführungszeit eines Anwendungsfalls zu messen. Analog zum funktionalen Testen sollte es möglich sein, zu überprüfen, ob Grenzwerte überschritten werden, und ggf. den aktuellen Build als fehlgeschlagen zu markieren. Diese Anforderung wird *Grenzwertüberprüfung* genannt.

Lange Antwortzeiten können u.a. durch Engpässe beim Arbeitsspeicher oder bei den Festplattenzugriffen entstehen. Daneben können andere Messwerte wie die CPU-Auslastung Auskunft über die Gründe von Performanzengpässen geben. Andere Performanzkriterien sollten, um derartige Gründe für schlechte Performanz schneller finden zu können, deshalb ebenfalls erfasst werden. Diese Anforderung wird *Messdatendiversität* genannt.

Durch die Veränderung von funktionalen und Performanzanforderungen kann es dazu kommen, dass Performanzwerte nicht mehr ausreichend sind. In diesem Fall ist es hilfreich, zu wissen, welche Entwicklungsschritte zu welcher Performanzveränderung geführt haben, um sie ggf. rückgängig zu machen. Zur Bereitstellung dieser Information ist die Speicherung und Anzeige des *Performanzverlaufs über Revisionen* notwendig.

Neben diesen zentralen Anforderungen gibt es weitere Anforderungen, die für kontinuierliches Performanztesten sinnvoll sind. Zu nennen sind hier eine Unterstützung von *paralleler Ausführung* von Teilprozessen eines Performanztests sowie eine Lösung, die bei Ausführung von Performanztests auf verschieden leistungsfähiger Hardware Vergleichbarkeit der Testergebnisse gewährleistet. Weiterhin sollte es möglich sein, durch *selbstdefinierte Datenmessung* Messwerte im Quelltext und nicht durch externe Messung zu bestimmen. Diese Probleme werden im Rahmen dieses Papiers nicht genauer behandelt.

## 3 Verwandte Arbeiten

Etablierte Methoden, um Performanz in der Softwareentwicklung zu verbessern, sind Lasttests, Performanzmodelle sowie Performanztests. Diese werden im Folgenden dargestellt.

Die Methode, Lasttests am Schluss der Entwicklung auszuführen, wird durch diverse Werkzeuge<sup>1</sup> unterstützt und durch Anleitungen wie [Mol09] beschrieben. Zur Überprüfung von Performanzanforderungen sind Lasttests vor dem Einsatz eines neuen Releases eines Programmes sinnvoll. Refactoring am Ende der Entwicklung ist jedoch oft aufwändiger als früheres Refactoring. Deshalb ist es sinnvoll, Performanz eher zu betrachten.

Hierfür existieren analytische Performanzmodelle. Bei diesen wird, ausgehend von Performanzannotationen an Architekturmodellen bspw. mit UML STP [Gro02] eine Performanzschätzung entwickelt. Als Modelle werden u.a. Warteschlangenmodelle, Petrinetze und Prozessalgebren verwendet [Bar09, Seiten 22-29]. Ähnlich arbeiten Simulationsmodelle [BGM03]: Hier wird statt des analytischen Modells eine Simulation zur Performanzvorhersage eingesetzt. Ein Performanzvorhersagewerkzeug ist Palladio.<sup>2</sup>

---

<sup>1</sup> Lasttestwerkzeuge sind u.a. JGrinder (<http://jgrinder.sourceforge.net/>) und JMeter (<http://jmeter.apache.org/>)

<sup>2</sup> Offizielle Webseite von Palladio: <http://www.palladio-simulator.com>

Performanzmodelle können Engpässe frühzeitig aufdecken und Wege zur Vermeidung aufzeigen. Problematisch ist, dass weder die Architektur noch die Performanz einzelner Methoden vor dem Ende der Entwicklung exakt bestimmt werden kann. Deshalb können Performanzmodelle nur einen Teil der Performanzprobleme aufdecken. Es ist also eine Methode notwendig, die Performanz vor dem Ende der Entwicklung exakt misst, um Refactorings zu vermeiden. [Pod08] argumentiert dafür, neben den herkömmlichen Methoden Einzelnutzer-Performanztests als Unit-Tests durchzuführen, denn eine gute Performanz für einen einzelnen Nutzer ist Voraussetzung für eine gute Performanz bei großer Last.

Für Performanztests existieren bereits einige Ansätze. Das verbreitete Testframework JUnit ermöglicht es, über die *@Timeout*-Notation *Grenzwertüberprüfung* durchzuführen. Andere der genannten Anforderungen unterstützt JUnit nicht. JUnitBench,<sup>3</sup> eine Erweiterung von JUnit, ermöglicht zusätzlich das Speichern der Aufrufzeiten über verschiedene Revisionen und das anschließende Visualisieren des *Messwertverlauf über Revisionen*. Eine andere JUnit-Erweiterung, JUnitPerf<sup>4</sup>, ermöglicht es, die Ausführungszeit nebenläufiger Tests zu prüfen. Beide JUnit-Erweiterungen erfüllen außer den genannten keine zusätzlichen Anforderungen. Ein weiteres Werkzeug, das die Performanzbetrachtung ermöglicht, ist das Performance Plugin für Jenkins.<sup>5</sup> Es ermöglicht das Speichern von Performanztestergebnissen von JUnit und JMeter. Damit ermöglicht es über JUnit die *Grenzwertüberprüfung* sowie die Erstellung des *Performanzverlaufs über Revisionen*. Eine Messung anderer Performanzkriterien ist nicht möglich.

Insgesamt erfüllt also kein Werkzeug die Mindestanforderungen für kontinuierliche Performanztests. Deshalb wurde KoPeMe<sup>6</sup> zur **K**ontinuierlichen **P**erformanz**m**essung erstellt.

## 4 Konzeption und Umsetzung

Um die in Kapitel 2 dargestellten Anforderungen umzusetzen, ist es notwendig, an drei Stellen des üblichen Buildprozesses Erweiterungen zu schaffen: Es muss die Möglichkeit geschaffen werden, Performanztests zu definieren und auszuführen, diese im Buildprozess aufzurufen und die Ergebnisse der Performanztests zu visualisieren. Das KoPeMe-Framework ermöglicht die Definition der Performanztests in Java, das Aufrufen der Performanztests im Buildprozess mit Maven und Ant sowie das Visualisieren der Ergebnisse in Jenkins. Im Folgenden werden die einzelnen Komponenten erläutert.

Die Tests in Java könnten als Erweiterung eines der verbreiteten Testframeworks JUnit oder TestNG oder als eigenständiges Werkzeug entwickelt werden. Die Tests wurden eigenständig und als JUnit-Tests implementiert. Durch die Umsetzung der JUnit-Performanz-Tests ist die Umwandlung bestehender Tests in Performanztests unproblematisch. Die JUnit-Tests umfassen sämtliche Funktionalitäten der eigenständigen Tests und müssen durch JUnit-spezifische Annotationen gekennzeichnet werden. Ein eigenständiger

---

<sup>3</sup><https://code.google.com/p/junitbench/>

<sup>4</sup><http://www.clarkware.com/software/JUnitPerf.html>

<sup>5</sup><https://wiki.jenkins-ci.org/display/JENKINS/Performance+Plugin>

<sup>6</sup>Webseite des Frameworks: [www.dagere.de/KoPeMe](http://www.dagere.de/KoPeMe). Quelltext unter <https://github.com/DaGeRe/KoPeMe>.



Test kann folgendermaßen aussehen:

Listing 1: KoPeMe-Test

```
@PerformanceTest(executionTimes=5, warmupExecutions=2 )
public void testMöbelkauf(final TestResult tr) {
    tr.startCollection();
    //Hier werden die Berechnungen ausgeführt
    tr.stopCollection();
    tr.setChecker(new Checker() {
        @Override
        public void checkValues(TestResult tr) {
            MatcherAssert.assertThat(
                tr.getValue(CPUUsageCollector.class.getName()),
                Matchers.greaterThan(10L));
        }
    });
}
```

Messungen müssen mehrmals ausgeführt werden, um Verfälschungen, bspw. durch das initiale Laden einer Klasse, zu vermeiden. In der Annotation wird deshalb angegeben, wie oft die Methode ohne Messung zum Aufwärmen (*warmupExecutions*) und mit Messung (*executions*) aufgerufen werden soll. Diese Parameter sind optional, die Standardwerte sind 2 und 5. Die Messung verschiedener Daten zur Umsetzung der *Messdatendiversität* erfolgt über beliebig erweiterbare Datenkollektor-Objekte. Derzeitig sind Standardkollektoren für Zeit, Arbeitsspeicherauslastung und CPU-Auslastung vorhanden. Es ist bspw. denkbar, einen Kollektor zu schreiben, der nur die Auslastung eines Prozessors in einer Mehrkernmaschine misst. Mit *setCollectors* kann festgelegt werden, welche Kollektoren genutzt werden sollen. Da es unerwünscht ist, die Performanz der Sicherstellungen in die Performanz des Anwendungsfalls hineinzurechnen, wurde die Möglichkeit geschaffen, mit *startCollection* und *stopCollection* zu markieren, an welchem Punkt die Datensammlung beginnen bzw. enden soll. Um selbst im Quelltext Messwerte festzulegen, kann nach *stopCollection* über *addValue(String key, long value)* ein Messwert hinzugefügt werden.

Standardmäßig wird über die Ausführungen ein Durchschnitt gebildet und dieser gespeichert. Es ist möglich, über das Setzen eines *MeasureSummarizer* andere Wertzusammenfassungen wie Median, Maximum und Minimum zu wählen oder eigene Verfahren zu implementieren. Die Zusicherungen werden als Objekt, das die *Checker*-Schnittstelle implementiert, übergeben. Diese wird nach dem mehrmaligen Aufruf der Methode ausgeführt. Neben der Performanzüberprüfung durch ein *Checker*-Objekt können Grenzwerte über Annotationen darzustellen. Auf diesem Weg wird *Grenzwertüberprüfung* ermöglicht. Mit einer ähnlichen Syntax können parallele Tests beschrieben werden.

JUnit-KoPeMe-Tests werden über die jeweiligen JUnit-Werkzeuge in den Buildprozess eingebunden. So ist auch eine Testausführung in einer Entwicklungsumgebung möglich. Die Ausführung der eigenständigen Tests durch Ant und Maven wurde basierend auf einer gemeinsamen Konsolenaufrufklasse, die die Tests in der Konsole ausführt, umgesetzt. Dadurch lassen sich Erweiterungen in beiden mit einer Implementierung umsetzen.

Die Ausgabedaten der Performanzmessung sind YAML-Dateien.<sup>7</sup> In diesen Daten wird für jedes Performanzkriterium die Abbildung des Ausführungszeitpunktes auf den Messwert

<sup>7</sup>Das YAML-Format (<http://yaml.org/>) zeichnet sich durch besonders gute Lesbarkeit und geringen Speicher-

gespeichert. Diese können im KoPeMe-Jenkins-Plugin, basierend auf JFreeChart, visualisiert werden. So wird der *Performanzverlauf über Revisionen* erzeugt.

Insgesamt ermöglicht KoPeMe das Spezifizieren von Tests, das Ausführen sowie das Visualisieren der Ergebnisse. Damit erfüllt KoPeMe alle Anforderungen für ein Werkzeug für kontinuierliche Performanztests.

## 5 Evaluation

Es wurde eine quantitative Evaluation an Tomcat 6 durchgeführt. Da Tomcat 6 ein umfangreiches, frei verfügbares Projekt mit vielen Beteiligten und performanzkritischen Bestandteilen ist, wurde es für die Evaluation ausgewählt. Die Tests wurden auf einem PC mit i5-Prozessor und Ubuntu 12.04 ausgeführt.

Bei den Anwendungsfällen Laden einer Servlet- bzw. JSF-Seite, die ihre Darstellungstexte jeweils durch Java-Methoden erhielten, zeigten sich gravierende Performanzunterschiede zwischen verschiedenen Revisionen. Die Abbildungen 1 und 2 zeigen auf der X-Achse die Zeit der Ausführung und auf der Y-Achse die Antwortzeit des jeweiligen Testfalls. Die Zeit der Ausführung lässt sich auf eine Revision abbilden.

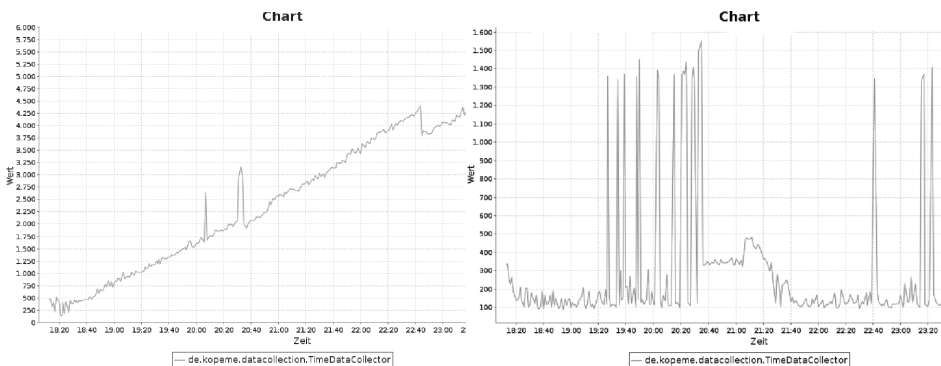


Abbildung 1: Verlauf der JSF-Downloadzeit    Abbildung 2: Verlauf der Servlet-Downloadzeit

Bei der JSF-Seite (Abbildung 1) steigt die Antwortzeit stetig an. Dies deutet darauf hin, dass in dem JSF-Renderingquelltext immer neue Veränderungen hinzukamen, die den Quelltext langsamer machen. Es ist davon auszugehen, dass bei kontinuierlicher Performanzbetrachtung performanzverbessernde Änderungen früher durchgeführt worden wären, wie bspw. in der Revision, die 22:40 getestet wurde. Beim Laden einer Servlet-Seite (Abbildung 2) zeigt sich noch deutlicher, dass ein KoPeMe-Einsatz die Entwicklung effizienter gemacht hätte: Die hohen, unregelmäßig auftretenden Ausschläge wären bei dem Einsatz von Performanztests schon während des Einreichens bzw. kurz danach entdeckt wurden. Insgesamt deutet die Evaluierung darauf hin, dass der Einsatz von kontinuierlichen Performanztests eine effizientere Softwareentwicklung unterstützen kann.

verbrauch aus.

## 6 Zusammenfassung und Ausblick

In diesem Beitrag wurde ein Ansatz zur Softwareentwicklung vorgestellt, der eine Entwicklung mit kontinuierlicher Performanzmessung und -überprüfung ermöglicht. Hierzu wird vorgeschlagen, ein Test-basiertes Vorgehen auch für nicht-funktionale Anforderungen der Software zu etablieren und diese bei jedem Build zu messen und sowohl gegenüber den initialen Anforderungen als auch im historischen Revisionsvergleich zu betrachten und bewerten. Im Gegensatz zu bestehenden praktischen Ansätzen wird dabei Performanz als Spektrum an Kriterien wie Ausführungszeit, CPU-Auslastung und Arbeitsspeicherverbrauch betrachtet. Für die Erstellung von Performanztests wird eine an JUnit angelehnte Spezifikation verwendet, die wenige zusätzliche Annotationen einführt. Die Ausführung im Build-Prozess wird sowohl für Ant- als auch für Maven-basierte Projekte unterstützt und eine Visualisierung der Messungen wird als Zusatz für den Jenkins Integrationsserver bereitgestellt.

Die Wirksamkeit des Ansatzes und des Werkzeuges wurden anhand einer quantitativen Evaluierung des Tomcat-Servers untersucht. Hierbei konnten Performanzsprünge zwischen Revisionen nachgewiesen werden, die bei Nutzung des Ansatzes u.U. hätten vermieden werden können. Als wichtiger Teil zukünftiger Arbeiten soll untersucht werden, ob und wie die Ergebnisse der Performanztests auch bei Ausführung auf unterschiedlicher Hardware weiter genutzt werden können.

## Literatur

- [Bar09] M. Barth. *Entwicklung und Bewertung zeitkritischer Softwaremodelle: Simulationsbasierter Ansatz und Methodik*. Dissertation, 2009.
- [BBvB<sup>+</sup>01] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland und D. Thomas. *Manifesto for Agile Software Development*, 2001.
- [BGM03] S. Balsamo, M. Grosso und M. Marzolla. *Towards Simulation-Based Performance Modeling of UML Specifications*. Bericht, 2003.
- [BMIS03] S. Balsamo, A. Di Marco, P. Inverardi und M. Simeoni. *Software Performance: state of the art and perspectives*. 2003.
- [Gro02] Object Management Group. *UML Profile for Schedulability, Performance, and Time Specification*. OMG Adopted Specification ptc/02-03-02, Juli 2002.
- [Mol09] I. Molyneaux. *The Art of Application Performance Testing - Help for Programmers and Quality Assurance*. O'Reilly, 2009.
- [Par94] D. Parnas. *Software aging*. In *Proceedings of the 16th international conference on Software engineering, ICSE '94*, Seiten 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
- [Pod08] A. Podelko. *Agile Performance Testing*. In *Int. CMG Conference*, Seiten 267–278. Computer Measurement Group, 2008.

# Visualizing cross-tool ALM projects as graphs with the Open Service for Lifecycle Collaboration

Oliver Siebenmarck, B.A.

Rational CoC  
IBM Deutschland GmbH  
Lise-Meitner-Str. 25-29  
24223 Schwentinental  
olisie@de.ibm.com

**Abstract:** Environments for Application Lifecycle Management (ALM) projects become increasingly more heterogeneous, with different integrated tools containing a lot of related pieces of information. It is hard enough to keep track of relations within a tool, but when relations exist across tools, this becomes next to impossible. Do all requirements have corresponding tasks and tests? Are there any circular dependencies? These questions can only be answered when taking the whole project into account, i.e. multiple tools are usually involved.

Using the Open Service for Lifecycle Collaboration (OSLC) as a standard for ALM tool integration, artifact dependencies within a project are analyzed across tools and a graph is generated. This graph contains the artifacts as nodes and their relationships as edges. It is displayed in a standard web browser, where it can be inspected and analyzed. When viewing the generated graph in the browser, especially on large displays, it is easy to grasp the structure of the entire ALM project. Patterns can be observed, giving an indication on whether the project is well structured or whether something is amiss (e.g. spaghetti or circular dependencies).

Thinking of the artifacts that make up an ALM project as nodes in graph and their relationships as edges allows for new ways of inspecting a project. Patterns can be recognized easily, even when they stretch across different tools, giving engineers a new way to reason about their project.

## 1 Introduction

The market for Application Lifecycle Management tools has seen profound changes in the last years, not the least of which is the move away from huge monolithic tools towards smaller, but highly integrated tools that are geared towards specific domains, as documented by Grant in [Gr12].

This change brings with it not only high demands on the way tools are integrated, but also raises the question of how a project that spans multiple tools can be sensibly managed.

This paper sets out to show how the Linked Data approach taken by the integration standard Open Services for Lifecycle Integration (OSLC) allows project managers and architects to think of their ALM projects as graphs spanning multiple tools as well as how these projects can be visualized.

## **2 Linked Data and the Open Services for Lifecycle Collaboration**

In contrast to more traditional integrations where data is just synched (i.e. copied on a regular basis) from one tool to another, OSLC takes a more modern approach. Based on Tim Berners-Lee's Linked Data [Be06], OSLC considers creating links from one tool to another as the primary means of integration. This stems from the belief that each tool, as it is already geared towards a specific use-case, is best at handling its data. Other tools hence do not need to copy its data, but rather need a way to point to it. A short example to illustrate this point:

When thinking about an ALM environment with three tools, one each for Requirements Management (Tool A), Test Management (Tool B) and Configuration Management (Tool C), a requirement in Tool A could be implemented by a user story described in Tool C and its implementation could be validated by a Test Case in Tool B. OSLC as an integration standard provides the means for how these relationships could be implemented as links. To this end, OSLC demands that tools expose their data in a standardized and restful way, so that other tools can access and link to them.

Additionally the standard mandates so-called UI previews, allowing parts of tool's user interface to be displayed within the context of another tool, e.g. to search for a specific artifact or create a new one.

The OSLC standard can be classified using Anthony Wasserman's model [Wa90] for tool integration. When seen in Wasserman's dimension, the data dimension of OSLC is REST-based while the presentation integration is based on HTML and CSS. OSLC does not provide any control integration, leaving the last dimension at 0. OSLC-based tool integration can thus be expressed as: ("REST-based", "HTML+CSS", 0).

## **3 Visualization of ALM projects as graphs**

In order to get a better understanding of a project's structure, it is not uncommon to use some kind of visualization. However, ALM tools hardly ever provide any means to analyze data that is located in another tool, but are confined to displaying only their own data. To get an understanding of the complete project, including the relationships between the different domains (requirements management, quality management, configuration management), a different approach is needed.

OSLC's Linked Data approach to tool integration offers just that: A way to interpret the project with its individual artifacts (such as defects, requirements, tasks etc) as nodes of a graph that represents the complete project. The relationships between these artifacts then form the edges of said graph.

Using a small tool to traverse the complete graph (or just some parts of it) a HTML/JavaScript visualization can be created and rendered in a browser. Figure 1 shows an extract of a sample ALM project.

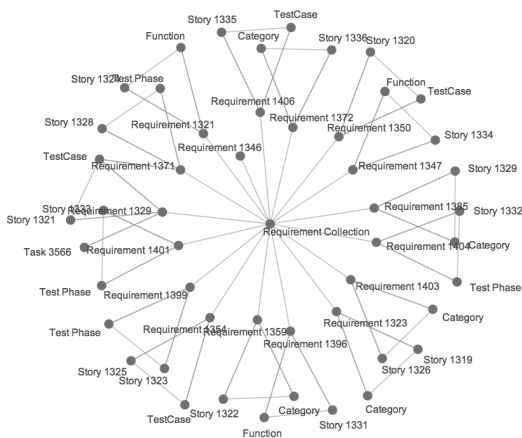


Figure 1: An ALM project as a graph

The extract shows a rather well structured project. A collection of requirements is centered in the middle, with requirements forming a circle around it. Most of the requirements already have an associated task by which they are implemented and a test case by which they can be validated. It should be noted that the nodes of the graph represent data in three different tools that can be seamlessly inspected in the graph.

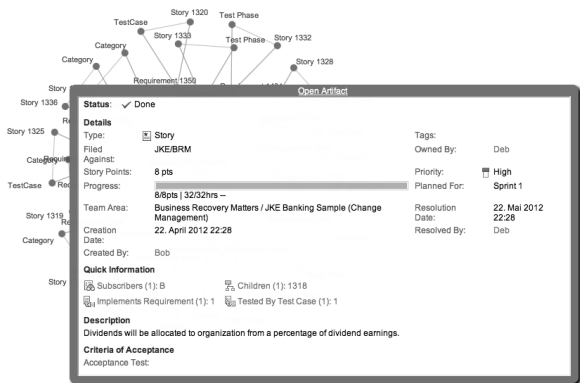


Figure 2: Using a UI preview to inspect individual nodes

Using the so-called OSLC previews allows for interactive inspection of the project's graph; by clicking on a node a preview of the artifact can be displayed. This preview is generated at runtime by the tool holding the data; it will thus always reflect the most current information.

One pattern that can be seen over and over again in the example is the triple of a requirement, an implementing task and a validating test case. This is as would expected and an indicator of a healthy project structure where every requirement has a corresponding implementation and validation. In contrast, a rather unhealthy pattern could be a 'spaghetti dependency', referring to a long chain of single relationships between artifacts as depicted in figure 3.

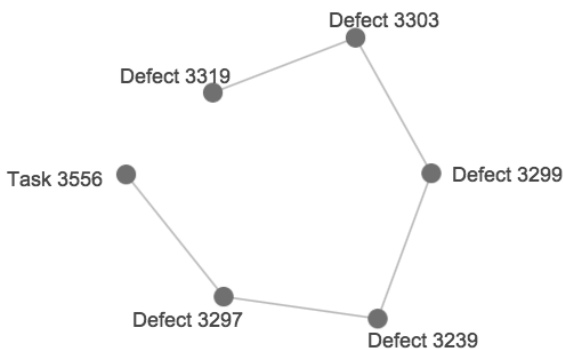


Figure 3: A spaghetti dependency

### 3 Future Work

This paper used a small tool to create and display graphs of ALM projects in a web browser. Future versions of this could incorporate more data, add query capabilities to the graph and provide project statistics. The prospect of an empirical study of ALM projects seems very promising, as it would allow not only for statistical analysis, but also for the compilation of a visual pattern library that might be used to evaluate ALM projects.

### Bibliography

- [Gr12] Grant, T. The Forrester Wave™: Application Life-Cycle Management, Q4 2012. [report] Cambridge: Forrester Research, Inc. 2012 p. 3
- [Be06] Berners-Lee, T. *Linked Data - Design Issues*. [online] Available at: <http://www.w3.org/DesignIssues/LinkedData.html>, 2006

[Wa90] Wasserman, A. Tool Integration in Software Engineering Environments. Lecture Notes in Computer Science, 467 p.137-149, 1990





# ACCD - Access Control Class Diagram

Martin Otto Werner Wagner

Department of Computer Science  
Technische Universität München  
Bolzmannstr. 3, 85748 Garching, Germany  
martin.wagner@tum.de

**Abstract:** The access control logics of an information system may become large and complex. To develop a system that fulfils the customer's requirements regarding access control three conditions are to be satisfied. These requirements are to be complete, unambiguous and defined by the customer. These three objectives are conflicting. Freely defined requirements, formulated in the customer's native language, may be incomplete and ambiguous. More formalised requirements, for example in an XML format, can be complete and unambiguous but may not be formulated or even understood by the customer. This work shows a diagram type based on the UML class diagram that helps to define complete and unambiguous access control logics. It aims to be used by non-IT experts like UML itself.

## 1 Introduction

Defining the access control of an IT system is basically just answering the question "who can do what?". The answer to this question is not as simple as it might appear at first. For instance it is not obvious that parts of the access control logic are realised in the system itself (static) others are to be configured in this system at runtime (dynamic).

Sometimes there is a choice whether an aspect of the access control logic is dynamic or static. It is conceivable that based on the same access control requirements roles could be implemented statically and dynamically. Roles are here understood as the concept of Role Based Access Control (RBAC) [FK92] describes. Of course the usage and usability of these two different implementations might differ. In this example the dynamic definition of roles would lead to more (non-functional) features but most likely also to more costs. Consequently a reasonable distinction between static and dynamic access control as well as a refined definition of the static access control is important. It is crucial for the correct implementation of the functional requirements, the usability and costs of the system.

The person who defines the requirements of a system (hereinafter referred to as customer) is often no IT expert but an expert of the problem domain. Nevertheless the customer has to be aware of the consequences of his decisions. In contrast to costs the consequences for security and usability cannot be quantified. Thus the customer has to understand the differences in detail, even though he is no IT expert. Two combinable approaches can help the customer during that process. Firstly IT experts can assist the customer, secondly a

comprehensible notation that covers the aspects access control may be applied. There is no comprehensible way to describe the access control logic of IT systems. For the general description of a system UML is relatively comprehensive [RSP07]. However, UML has no dedicated concept for modelling access control. A combination of class and use case diagrams can be utilised to define simple access control logic. This approach is not sufficient because the models become enormous and certain requirements can't be modelled. In other approaches UML-Profiles define a set of UML-extensions for class diagrams. None of the approaches support the description of the entire access control logic in a comprehensible way. Their shortcomings are for example: class diagram elements are multiplied by a factor of more than two, cryptic description of logics, class elements and their access control logic is described separately, access control logic and program logic is mixed together, and the modelling of important access control aspects is not possible. This idea paper is structured as follows: In chapter 2 important concepts of access control, that should be expressible in modelling languages for access control, are defined. Chapter 3 gives an overview of the existing relevant approaches and explains their capabilities, strengths and weaknesses. In chapter 4 an approach, based on UML class diagrams, will be introduced and explained. Some ideas for future work are described in chapter 5.

## 2 Important concepts for access control logic

Firstly a class diagram of an application for managing calendars is shown (figure 1) and some exemplary requirements are stated. On the basis of this class diagram important concepts for access control logic are stated. A modelling language for access control should offer to express these concepts.

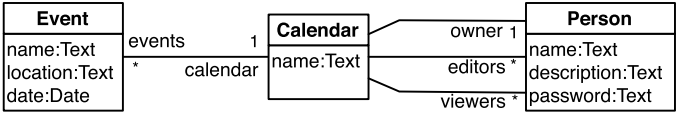


Figure 1: Example: class diagram of a simple calendar application

**Exemplary (access control) requirements:** The application manages calendars that can have several Events and Persons. Everybody may register to the application as a Person, each registered Person may create a new calendar. Different Persons can access a calendar with different rights which are described in roles (Owner, Editor, Viewer). A Person who has a role on a calendar has automatically the same role on all associated Events and Persons. A Viewer of a calendar may read name of the calendar, read all attributes of the calendars events, and read name and description of all viewers and editors and the owner of the calendar. A Editor of a calendar has all Viewer rights and can add Events and change attributes of Events and may add and remove Persons as Viewers. An Owner of a calendar has all Editor rights and can remove Events and the calendar itself, may add and remove Persons as Editors and change the Owner to another Person. Each Person has the role Self on itself that allows to read and change all attributes of it.

**Important concepts:** A role based assignment of rights is important as the single rights

for a calendar with its associated events and persons should not be defined for each editor separately (**roles**). It has to be defined who owns which role (**role assignment**). A hierarchical approach like described in RBAC is desirable (**role hierarchy**) as it simplifies the definition of the roles editor and owner. Rights can differ between the instances of the same class. This is required as different calendars can have different owners (**instance distinction**). Rights can differ between the properties of an object as a third party may see the name but not the password of a Person (**property distinction**). Some rights have to be granted without an existing object. For example a calendar should be creatable without its previous existence (**static rights**). Other rights have to be granted without an existing subject. For example someone should be able to register to the system without his previous existence in the system (**anonymous rights**). Rights on an object can include rights on other objects. For example a calendars owner should be automatically the owner of the calendars events (**transitive inclusion**). Subjects are accessible objects as the owner of a calendar should be seen (**subject access**). The bold written requirements names will be used in chapter 3 for classifying the existing approaches.

### 3 Related work and an overview of access control model types

The Unified Modeling Language (UML)[BRJ96] is a common [DP06] visual modelling language. As an extension to UML the Object Constraint Language (OCL) [Rat97, RG98] has been defined for formal definitions on models. It allows a modeller to specify the characteristic of an instance of a model. Role-Based Access Control (RBAC)[FK92, SCFY96] is a common[OL10] standard[FSG<sup>+</sup>01] to manage access rights in IT systems. In the first column the names of the different approaches are stated. In the following columns the concepts defined in chapter 2 are evaluated, ✓ tells that this concept is expressible. In the last column statement for the comprehensibility is given (1 = bad, 2 = medium, 3 = good) which reflects the personal opinion of the articles author.

approach	roles	role assignment	role hierarchy	instance distinction	property distinction	static rights	anonymous rights	transitive inclusion	subject access	comprehensibility
Epstein/Sandhu	✓	✓	✓		✓	✓	✓		✓	2
UMLsec					✓	✓	✓		✓	2
Shin/Gail-Joon	✓				✓	✓	✓		✓	2
SecureUML	✓	✓	✓	✓	✓	✓	✓		✓	1
Kuhlmann/Sohr/Gogolla	✓	✓	✓	✓	✓	✓	✓		✓	1
ACCD	✓	✓	✓	✓	✓	✓	✓	✓	✓	

Figure 2: Overview of the expressiveness of access control models

In the following the existing graphical approaches are shown for modelling access control.

Some of them have been named by their creators, the others are named by their authors.

**Epstein/Sandhu** This graphical modelling approach has a focus of expressing the concept of RBAC [ES99] for IT systems. It is designed for a specific software development tool, the Role Based Access Control Framework for Network [TOB98], thus its range of application is very reduced. **UMLsec** UMLsec[Jür02, BJN07] is an UML based approach for modelling security aspects that is much broader than the aim of defining access control. It focuses more the communication and encryption and access control rights can't be defined very specifically. **Shin/Gail-Joon** An OCL based approach [SGJ00, AS01] defines access control logics via constraints (e. g. separation of duty constraints). While it is very powerful in defining permissions it doesn't focus on roles and role assignment. **SecureUML** is a modelling language on the basis of UML and OCL is SecureUML [LBD02]. SecureUML provides the broad spectrum of constraints as it is based on OCL. **Kuhlmann/-Sohr/Gogolla** [SMBA08, KSG13] uses UML and OCL to define access control logic that bases concepts of SecureUML. It separates the access control logic from the application logic. This allows subsequent development of access control functionalities without the necessity of change the application itself.

#### 4 ACCD: The Access Control Class Diagrams

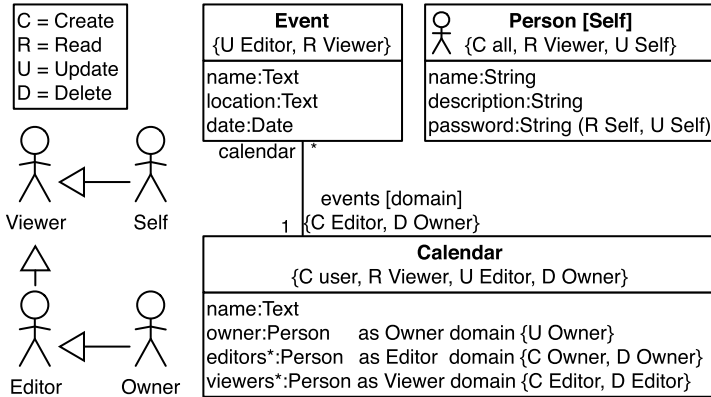


Figure 3: The Access Control Class Diagram for the calendar application

In this chapter the calendar application is modelled in an Access Control Class Diagram. On the basis of this model Access Control Class Diagrams are explained. Access Control Class Diagrams are based on class diagrams. The class diagrams are enriched by the information that is necessary to define access control logic. All important concepts (defined in chapter 2) can be expressed by this approach. **Roles** can be expressed with the actor symbol of use case diagrams. A **role hierarchy** can be defined by using arrows between the defined roles. A role can include another roles rights by pointing with an arrow on it. **Role assignment** can be defined with the keyword **as**. The **as** can only be used in references to subjects. The referenced subjects will gain the Role which is stated after the **as**. **Instance**

**distinction** is achieved as the referenced subjects are defined in the concrete instance individually. **Property distinction** is achieved as the rights can be defined in the granularity of single properties. If not rights are defined for a property the rights defined for the class will be inherited. On a class the rights create, read, update and delete can be defined while create allows to create a instance of the class and delete allows to delete this concrete instance of the class. The rights read and update are only for the inheritance to attributes of the class. Attributes are the properties of a class on which the defined type behind the colon are a primitive data type of the set (*Int, Double, Float, String, Boolean, Date*). For an attribute the right read allows to see its value and update allows to change its value. References are properties of a class on which the defined type behind the colon is the Name of an existing class or subject class. Subjects are classes which are tagged with a stick-figure. On references the right create allows to add a referenced instance and the right delete allows to remove a referenced instance. Read and update rights on referenced instances can be defined via transitive inclusion. **Static rights** can be defined with the keyword **user**. Any registered user of the system can execute the rights assigned to **user**. **Anonymous rights** can be granted with the keyword **any**, these rights can be performed without being a subject in the system. **Transitive inclusion** is defined with the keyword **domain**. The roles assigned to an instance are automatically applied to the instances which are referenced with keyword **domain**. **Subject access** is allowed as subject classes can be referenced like normal classes.

## 5 Future Work

There are many open topics for future work. The evaluation of existing approaches should be extended. The concepts which are important for access control should be evaluated. Either through inquiring software developers or the usage of the ACCD in real projects. For the usage of ACCD in real projects two aspects are interesting: Is all required access control logic definable in ACCD and do customers and software developers understand the concept of ACCD. The observation of customers and software developers during the definition of access control could show the problems they usually face. A transformation logic can be defined which allows the usage of ACCDs as the model of model driven development, code generation directly from an ACCD is desirable.

## References

- [AS01] Gail-Joon Ahn and Michael E Shin. Role-based authorization constraints specification using object constraint language. *Proceedings Tenth IEEE International Workshop on Enabling Technologies*, pages 157–162, 2001.
- [BJN07] Bastian Best, Jan Jurjens, and Bashar Nuseibeh. Model-based security engineering of distributed information systems using UMLsec. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 581–590. IEEE, 2007.

- [BRJ96] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The unified modeling language*. University Video Communications and the Association for Computing Machinery, 1996.
- [DP06] Brian Dobing and Jeffrey Parsons. How UML is used. *Communications of the ACM*, 49(5):109–113, 2006.
- [ES99] Pete Epstein and Ravi Sandhu. Towards a UML based approach to role engineering. In *Proceedings of the fourth ACM workshop on Role-based access control, RBAC '99*, pages 135–143, New York, NY, USA, 1999. ACM.
- [FK92] David F Ferraiolo and D Richard Kuhn. Role-Based Access Controls. *National Computer Security Conference*, pages 554–563, January 1992.
- [FSG<sup>+</sup>01] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.
- [Jür02] Jan Jürjens. UMLsec: Extending UML for Secure Systems Development. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*. Springer-Verlag, September 2002.
- [KSG13] Mirco Kuhlmann, Karsten Sohr, and Martin Gogolla. Employing UML and OCL for designing and analysing role-based access control. *Mathematical Structures in Computer Science*, pages 796–833, 8 2013.
- [LBD02] Torsten Lodderstedt, David A. Basin, and Jürgen Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proceedings of the 5th International Conference on the UML, UML '02*, pages 426–441. Springer-Verlag, 2002.
- [OL10] Alan C. O'Connor and Ross J. Loomis. 2010 Economic Analysis of Role-Based Access Control. *RTI International report for NIST*, December 2010.
- [Rat97] Rational Software Corporation. Object Constraint Language Specification, 1997.
- [RG98] Mark Richters and MARTIN GOGOLLA. On formalizing the UML object constraint language OCL. *Conceptual Modeling–ER'98*, pages 449–464, 1998.
- [RSP07] Rozilawati Razali, Colin F Snook, and Michael R Poppleton. Comprehensibility of UML-based formal model. In *WEASEL Tech '07*. ACM Request Permissions, November 2007.
- [SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *Computer*, 29(2):38–47, February 1996.
- [SGJ00] M E Shin and Gail-Joon. UML-based representation of role-based access control. *IEEE 9th International Workshops on Enabling Technologies Infrastructure for Collaborative Enterprises*, pages 195–200, 2000.
- [SMB08] Karsten Sohr, Tanveer Mustafa, Xinyu Bao, and Gail-Joon Ahn. Enforcing role-based access control policies in web services with UML and OCL. In *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, pages 257–266. IEEE, 2008.
- [TOB98] Dan Thomsen, Dick O'Brien, and Jessica Bogle. Role Based Access Control Framework for Network Enterprises. *Proceedings of 14\* Annual Computer Security Application Conference*, pages 50–58, dec 1998.

# Formal software verification for the migration of embedded code from single- to multicore systems

Thorsten Ehlers<sup>a</sup>, Dirk Nowotka<sup>a,\*</sup>, Philipp Sieweck<sup>a</sup> and Johannes Traub<sup>b</sup>

<sup>a</sup>Dependable Systems Group  
Department of Computer Science  
Kiel University  
{the, dn, psi}@informatik.uni-kiel.de  
<sup>b</sup>Powertrain Electronics, Daimler AG  
johannes.traub@daimler.com

**Abstract:** The introduction of multicore hardware to the field of embedded and safety-critical systems implies great challenges. An important issue in this context is the migration of legacy code to multicore systems. Starting from the field of formal verification, we aim to improve the usability of our methods for software engineers. In this paper, we present approaches to support the migration process, mainly in the domain of safety-critical, embedded systems. The main contribution is a verification process which is inspired by the waterfall model. Especially, backtracking is considered carefully.

This work is part of the ARAMiS<sup>1</sup> project.

## 1 Introduction

Multicore systems are becoming more and more common, also in the domain of safety-critical embedded systems. Multiple challenges arise from this, ranging from the design of appropriate hardware to the development of software engineering techniques. One of these challenges is the avoidance of race conditions. Race conditions are accesses of at least two tasks to the same memory location, with at least one task writing. This kind of bugs can cause inconsistent data states, and thus unpredictable system behaviour. Although race conditions are not restricted to multicore systems, the probability of their occurrence is much higher on systems using several CPUs.

As the software used in the automotive domain has become more and more complex - nowadays, it has reached a magnitude of more than 10,000,000 lines of code[BKPS07] in one car - tool support is urgently needed. Our focus lies on the formal verification of software, especially in the automotive domain. In [NT12, NT13] we presented MEMICS. This tool supports the verification of OSEK/AUTOSAR-code [ose, Con], as it is used in

---

\*This work has been supported by the BMBF grant 01IS110355.

<sup>1</sup>Automotive, Railway and Avionics Multicore Systems <http://www.projekt-aramis.de/>



automotive systems. The main focus lies on the detection of race conditions. Additionally, other runtime errors like e.g. “DivisionByZero” or “NullDereference” can be detected.

Nevertheless, there are more requirements than “only” formal correctness. In order to provide remarkable impacts in practical issues, they need to be incorporated into the software engineering processes. In this paper, we show how to involve MEMICS in the process of migrating legacy software from single- to multicore systems. Therefore, we suggest to organize the migration process according to a waterfall model. This model contains the possibility of backtracking in well-defined cases. Furthermore, we suggest to combine testing and model checking. This approach can be used to check whether the behaviour of a systems remains deterministic with respect to scheduling decisions after migrating to a multicore environment.

## 2 Technical Background

Operating systems compatible to OSEK/AUTOSAR implement tasks and interrupt service routines, which have a fixed priority, and are statically assigned to a core. In a single-core setting, these strict priorities allow only a small number of interleavings. When migrating to a multicore system, there are basically two possibilities. The software engineer can chose one core for every task, or split the functionality of a task into two or more new tasks, and distribute them to different cores. Obviously, the first alternative is the easier one, whereas the second possibility can provide more benefits. Nevertheless, both approaches significantly increase the number of possible interleavings between tasks. Hence, safety properties of the software have to be checked carefully after the migration. If these checks show possible race conditions, they can be avoided by changing the distribution of tasks to cores, or otherwise by adding synchronization via semaphors.

## 3 Related Work

In the last decade, formal software verification became usable for practical applications. Microsoft reports several cases [BCLR04]. One of them is SLAM [BLR], which is used to check whether device drivers use the Windows Driver Model (WDM) according to its specifications. Only drivers that pass this check become digitally signed, indicating that their usage will not compromise the stability of the system. Furthermore, Microsoft uses VCC [CDH<sup>+</sup>] in order to formally prove the absence of errors in their hypervisor. As this is a more complex task, it requires the programmers to annotate their code with formal pre- and postconditions. In the avionic domain, formal methods are seen as a supplement of testing [MLD<sup>+</sup>13]. The reason is not only the completeness of their results, but also the possibility to decrease the costs for testing [CKM12].

Tools like Bauhaus [RVP06] and Polyspace [pol] are able to find runtime-errors in OSEK/-AUTOSAR code. Yet, they work with a large overapproximation of the program behaviour. This yields a lot of false-positive results, making it difficult to use their results

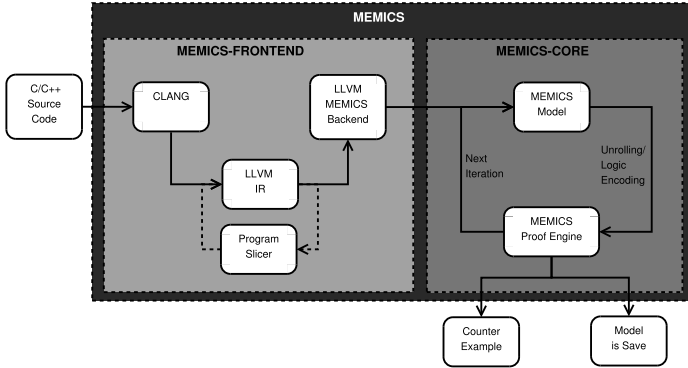


Figure 1: Using the LLVM-IR to generate a MEMICS model

in an iterative migration approach. The high number of false-positives forces the software engineers to perform a manual code review for each of them, taking lots of time. As part of the ARAMiS project, the model checker MEMICS was developed to deal with this problem [NT12, NT13]. It can be used to find software errors itself, or to check error candidates given by tools like Bauhaus or Polyspace.

ESBMC [RBCN12] or Threader [GPR11] are able to handle multithreaded code. They have two major drawbacks: They do not support scheduling according to OSEK and AUTOSAR, and their memory model is not precise enough to significantly reduce the number of false-positive errors reported.

## 4 MEMICS

Mainly designed to find race conditions, MEMICS is also able to find other runtime errors like DivByZero, NullDereference, and others. It supports priority-based scheduling as used in OSEK/AUTOSAR-based operation systems. The input is either C/C++-code or LLVM IR code [LA04, Lat], which makes it compatible to all languages supported by Clang [Fan10]. This input is transformed to an internal model which is based on the MIPS assembly language, see figure 1. There are basically two modi operandi. Either MEMICS is used to verify an error candidate, given e.g. by Polyspace, or it is used to find and verify error candidates itself. In the first case, MEMICS tries to find a program trace to the error location. The actions on such traces are translated into a first order logic formula, which is solved with regard to a flat memory model that allows tracking arbitrary data structures including pointers and function pointers. Therefore, the question whether a trace is feasible in the original program or not can be answered with high confidence, which is a big advantage in comparison to other tools like Threader [GPR11] or ESBMC [RBCN12]. If a feasible trace is found, it is reported as an error. Otherwise, other traces are searched until either a feasible one has been found, or safety is proven w.r.t. this error candidate. Without given error candidates, MEMICS can unroll the program itself,

following only feasible traces and looking for runtime errors. Although this works for benchmarks, this approach suffers from state space explosion, thus it should be used only if the preprocessing fails, or to compute pre/postconditions for small parts of a program, as suggested in [CKM12].

## 5 Supporting the Migration Process

We show how different tools can be used to combine their strengths, and find some of the hardest bugs faced when using parallel hardware: race conditions.

### 5.1 Model Checking

As implied before, there exist tools like Polyspace and Bauhaus which are able to deal with OSEK/AUTOSAR code. Though, they perform an abstract interpretation, yielding an overapproximation of the set of errors. Checking each of these error candidates manually is far too time consuming, and hence too expensive. We use MEMICS in order to find a trace to an error candidate. If there exists a feasible trace, this indicates the existence of a real error. Otherwise, we have found a false positive which can be removed. On the one hand, this massively shrinks the search space and hence accelerates the model checking. On the other hand, we remove false positive results, which makes the results usable for software engineers.

### 5.2 Combining Testing and Model Checking

In this section, we describe how to combine model checking with testing. We may assume that there exist test cases for the software under consideration, which run in the single-core setting without faults. Furthermore, we assume that model checking as described in section 5.1 has proven the absence of race conditions like Lost Updates. Such race conditions are only a subclass of problems that may occur when dealing with parallel programs [NM92]. Another type of errors which are hard to detect are data races, i.e. situations in which the behaviour of the program depends on the scheduling. Please note that data races may occur even if every access to global memory is synchronized. Although such errors can be found by model checking, they are practically intractable for large programs. Hence, we propose to use test cases from the single-core setting. For each of the test cases we perform model checking, restricting the behaviour of the program according to the test inputs. If we can prove that the results are unique, this implies that they do not depend on the scheduling. Otherwise, we can provide traces which lead to different results.

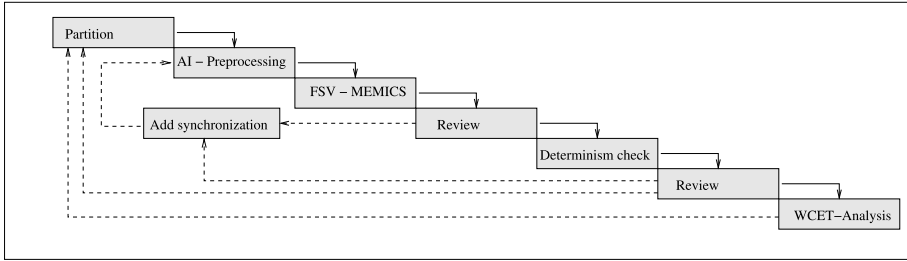


Figure 2: Iterative workflow

### 5.3 Combined Workflow

Putting this together, we derive an iterative workflow for the migration process, as shown in figure 2. Firstly, a software engineer decides how to distribute the tasks onto the cores, or splits tasks into subtasks, as depicted in section 2. Tools based on abstract interpretation like Polyspace or Bauhaus are used to find error candidates, which are checked using more precise tools, e.g. MEMICS. As we do not require annotations in the code, it is possible to retain false-positive results. This may happen e.g. if interfaces have restrictions that are not visible in the code itself. Thus, the results must be checked by a software engineer. If there are possible race conditions, either another partition can be chosen, or they can be fixed by introducing semaphors.

In the next step, test cases from the single threaded setting can be used to check whether the program is deterministic or not. Again, the importance of nondeterministic behaviour must be evaluated. If no non-determinism occurs, or if it is irrelevant, the process continues with the analysis of the worst case execution time or worst case reaction time. Otherwise, the overall process is iterated, beginning with a new partition.

## 6 Conclusion & Future Work

We proposed an iterative process to migrate OSEK/AUTOSAR-code from single- to multicore. Furthermore, we suggested how to support this by using testing, abstract interpretation and model checking. Given an appropriate exchange format, parts of this process can be run automatically. Nevertheless, this process cannot be fully automated. Hence, we hope to offer helpful support. This requires sufficiently precise results, as too many false positive error reports slow down the process.

Further research will be necessary to improve the efficiency of this process. This does not only concern better running times and increased precision of the verification processes, but also a deeper understanding of the information required by the software engineer, and his interaction with the tool box.

## References

- [BCLR04] T. Ball, B. Cook, V. Levin, and S. Rajamani. SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft. In EerkeA. Boiten, John Derrick, and Graeme Smith, editors, *Integrated Formal Methods*, volume 2999 of *LNCS*, pages 1–20. Springer Berlin Heidelberg, 2004.
- [BKPS07] M. Broy, I.H. Kruger, A. Pretschner, and C. Salzmänn. Engineering Automotive Software. *Proceedings of the IEEE*, 95(2):356–373, 2007.
- [BLR] T. Ball, V. Levin, and S. K. Rajamani. A Decade of Software Model Checking with SLAM.
- [CDH<sup>+</sup>] E. Cohen, M. Dahlweid, M. Hillebrand, D. Leinenbach, M. Moskal, T. Santen, W. Schulte, and S. Tobies. VCC: A practical system for verifying concurrent C. In *IN CONF. THEOREM PROVING IN HIGHER ORDER LOGICS (TPHOLS), VOLUME 5674 OF LNCS*.
- [CKM12] C. Comar, J. Kanig, and Y. Moy. Integration von Formaler Verifikation und Test. In *Automotive - Safety & Security*, pages 133–148, 2012.
- [Con] Autosar Consortium. *AUTOSAR - Specification of Operating System*. <http://autosar.org>.
- [Fan10] D. Fandrey. Clang/LLVM Maturity Report. June 2010. See <http://www.iwi.hs-karlsruhe.de>.
- [GPR11] A. Gupta, C. Popeea, and A. Rybalchenko. Threader: A Constraint-Based Verifier for Multi-threaded Programs. In *CAV*, pages 412–417, 2011.
- [LA04] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.
- [Lat] C. Lattner. *LLVM Language Reference Manual*. <http://llvm.org/docs/LangRef.html>.
- [MLD<sup>+</sup>13] Y. Moy, E. Ledinot, H. Delseny, V. Wiels, and B. Monate. Testing or Formal Verification: DO-178C Alternatives and Industrial Experience. *IEEE Software*, 30(3):50–57, 2013.
- [NM92] R. H. B. Netzer and B. P. Miller. What are race conditions?: Some issues and formalizations. *ACM Lett. Program. Lang. Syst.*, 1(1):74–88, March 1992.
- [NT12] D. Nowotka and J. Traub. MEMICS - Memory Interval Constraint Solving of (concurrent) Machine Code. In *Automotive - Safety & Security 2012*, Lecture Notes in Informatics, pages 69–84, Bonn, 2012. Gesellschaft für Informatik.
- [NT13] D. Nowotka and J. Traub. Formal Verification of Concurrent Embedded Software. In *IJSS*, pages 218–227, 2013.
- [ose] OSEK. <http://portal.osek-vdx.org>.
- [pol] Polyspace. <http://www.mathworks.com/products/polyspace>.
- [RBCN12] H. Rocha, R. Barreto, L. Cordeiro, and A. Neto. Understanding Programming Bugs in ANSI-C Software Using Bounded Model Checking Counter-Examples. In John Derrick, Stefania Gnesi, Diego Latella, and Helen Treharne, editors, *Integrated Formal Methods*, volume 7321 of *LNCS*, pages 128–142. Springer Berlin Heidelberg, 2012.
- [RVP06] A. Raza, G. Vogel, and E. Plödereder. Bauhaus - A Tool Suite for Program Analysis and Reverse Engineering. In *Reliable Software Technologies - Ada-Europe 2006*, volume 4006 of *LNCS*, pages 71–82. Springer Berlin Heidelberg, 2006.

# Architecture-driven Incremental Code Generation for Increased Developer Efficiency

Malte Brunnlieb

Capgemini Deutschland GmbH  
malte.brunnlieb@capgemini.com

Arnd Poetzsch-Heffter  
AG Softech

TU Kaiserslautern  
poetzsch@cs.uni-kl.de

**Abstract:** Modern information systems are often developed according to reference architectures that capture not only component structures, but also guidelines for framework usage, programming conventions, crosscutting mechanisms e.g., logging, etc. The main focus of reference architectures is to allow for reusing design knowledge. However, following good maintainable and large reference architectures often creates an undesirable programming overhead.

This paper describes a fine-grained pattern-based generative approach that reduces such implementation overhead and improves the overall developer efficiency by transferring the advantages of reference architectures to code level. Our generative approach extracts specific information from the program, generates implementation and configuration fragments from it, and structurally merges them into the code base. Integrated into the Eclipse IDE, the generation can be incrementally triggered by the developer within the developer's individual workflow. Furthermore, the experiences made within three real world projects will be presented.

## 1 Motivation

As part of today's software engineering, especially in model driven development, generators have become omnipresent. Until today very different use cases have been managed by different generator applications, e.g., generation of code from different models or Create/Read/Update/Delete (CRUD) user interface generation like done by SpringFuse/Celerio [RR13] or the NetBeans IDE [Com13]. Code generators are often domain specific and focus on small to medium scale projects. Nevertheless, to assure architectural conformance and thus good maintainability also of generated code, the generated code has to be structured in an architectural consistent and easily readable way especially in large scale projects. Hence, our generator approach focuses rather on small architecture-driven implementation patterns. Using these it becomes possible to guide and support the developer during the development of architecture compliant software. This paper explains a text-template-based generator approach—the APPS-Generator<sup>1</sup>—, which is capable of generating very fine-grained increments of code. It has been implemented as an internal developer tool<sup>2</sup> focusing on a smooth integration into the developer's workflow and thus resulting in a guided and architecture-compliant development.

---

<sup>1</sup>APPS stands for Application Services—a Capgemini-internal business unit

<sup>2</sup>with all rights belonging to Capgemini Deutschland GmbH

## 2 Related Work of Pattern-based Code Generation

By definition of Arnoldus et al. [AvdBSB12] template-based code generators can be divided into homogeneous and heterogeneous generators. The presented approach can be classified as heterogeneous template-based generator as the template (meta) language is different from the target (object) language. In the domain of homogeneous template-based generators, there are approaches like [BST<sup>+</sup>94] to enable reusable patterns right within object language. There are also heterogeneous template-based generators, e.g., for CRUD applications like [RR13][Com13]. Also use case independent template-based generators like the Hibernate JPA 2 Metamodel Generator [Inc10] are available to generate any contents from a database input source. As a generation approach focusing on patterns Budinsky et al.[BFVY96] realized a generator for design patterns defined by the Gang of Four (GoF). In contrast to such a generic approach, Heister et al.[HRS<sup>+</sup>98] indicate, that code generation is more effective for domain specific environments as patterns can be much more specific as in a general context. Our presented APPS-Generator will also implement an approach with basic assumptions on architectural guidelines, such that it can be considered as domain specific, too. The major difference is the ability of incrementally generating code right into the currently developed solution by using structural merge techniques. Furthermore, due to incrementally generating code, the APPS-Generator focuses on fine-grained code templates with the advantage to be able to generate small increments directly fitting into the developers individual workflow.

## 3 APPS-Generator

For the development of the APPS-Generator, we identified five challenges which have to be managed in order to implement a generic, integrated, and fine-grained incremental generation approach:

1. Structural merge
2. Extension of the template language for enabling Java type checks
3. Context-aware / parametrized generation
4. Generation of semantically and technically closed increments
5. Usable integration into a development environment

With a text-template-based generation approach, we can generate any text-based content from a given object model. But for an integrated and fine-grained incremental generation approach we also have to cope with very fine-grained contents which have to be merged into existing files. So we do not refer to the original use case of text-templates to generate one file per template, but we rather use text-templates to generate patches, which can be applied to existing code. To implement this use case, we have divided the APPS-Generator into mainly three core components, one integrated open source template engine, and the user interface as described in Figure 1. The first input for the APPS-Generator is an input file containing information for processing the target templates. Therefore the input transformer extracts all needed information from the input file and provides these in a simple object model for template processing. After that the model processor injects further context information into the model, which are defined in the context configuration. Currently

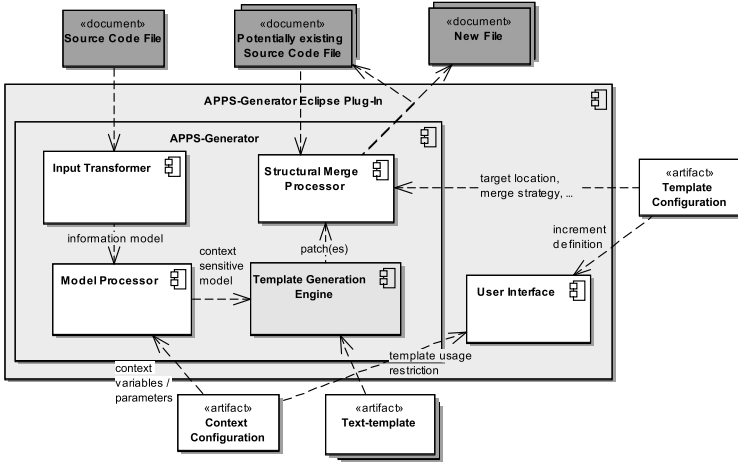


Figure 1: High-level architecture of the APPS-Generator

this can be injected constants or extracted values from the input file location. This enabled us to meet Challenge 3. Furthermore, the model will be extended by a toolkit implementation, which enables simple Java type checks like 'isSubtypeOf' in the text-templates later on such that we can meet Challenge 2. The next step of the generation process is to create a patch, which has to be applied to a potentially existing file afterwards. The patch generation is done by a template generation engine like FreeMarker[pro13] or Velcity[Fou13], which basically generates text from a given object model and a text-template. This generated patch as well as a newly introduced template configuration are the inputs for the structural merge processor addressing Challenge 1. The template configuration mainly specifies the target file location and some more meta-values for every template. If the target file already exists and the target language is supported for a structural merge, the patch will be merged into the existing file. Otherwise a new file will be generated.

On top of the APPS-Generator we have developed a user interface as an eclipse plug-in. With this user interface we are able to meet the last two challenges. On the one hand it consumes the context configuration as this configuration also provides a mapping of inputs to possibly usable templates. This mapping is used to restrict the user to meaningful generation products, which lead to good information hiding as basis for a usable user interface meeting Challenge 5. On the other hand the user interface reads the template configuration as this configuration also defines groups of templates. These groups specify small software increments, which are defined on the basis of the developers needs and which always lead to valid compiler results. This grouping of templates and the ability to integrate small patches into existing files enables us to meet also Challenge 4 of the challenges.

To get a deeper understanding about the different input artifacts and how the Challenges 1-3 have been meet by the APPS-Generator implementation, let's assume our architecture prescribes us to implement a copy constructor for every entity in the core layer. Furthermore, the copy constructor should only perform deep copies for fields, which type are



```

package ${pojo.package};
class ${pojo.name} {
    ${pojo.name}(${pojo.name} o) {
        <#list pojo.fields as field>
            this.${field.name} = o.${field.name};
        </#list>
    }
}

```

Listing 1: FreeMarker template for a simple copy constructor

subtypes of any specific architectural defined type. Therefore we first need to specify the location of the copy constructor dependent on the target language structure in the template. This can be achieved by specifying the same type within the same package as existent in the input file. As the current implementation of the APPS-Generator uses the FreeMarker engine to generate patches, the example of such a template—shown in Listing 1—is specified using FreeMarker syntax: `${...}` states a read action on the object model and the `<#list ... as ...>...</#list>` construct iterates over the list of Java fields. So we generate a patch, which does not only contain the copy constructor itself, but also the package and class declaration. This is necessary to cause a structural merge conflict of the type declaration later on. For now the copy constructor is simply implemented by a reference copy of each field. In addition, the target file location will be defined in the template configuration as shown in Listing 2. The `destinationPath` is a relative path starting at a configurable root folder. We define it equally to the chosen input file's path and select the `javamerge_override` merge strategy. This strategy tells the APPS-Generator to merge the patch into an existing file if necessary and whenever a unresolvable conflict occurs—e.g., the copy constructor already exists—the patch will override the conflicting contents, whereas conflicts of classes will be resolved recursively. So far we gain a fine-grained template which (re)generates a copy constructor for any Java class as input. In addition to the Java merge algorithm there are also implementations for structural merging XML and the Java notation.

```

<template id="copy_constructor"
    destinationPath="src/${pojo.package}/${pojo.name}.java"
    templateFile="copy_constructor.ftl"
    mergeStrategy="javamerge_override" />

```

Listing 2: Template configuration for the copy constructor template

For copy constructors it might also be interesting to consider specific types to be deep copied, such that we need simple Java type checks in the template language. As FreeMarker itself does not provide such checks, we have implemented a utility class for 'isSubtypeOf' and 'isAbstract' type checks. The utility implementation holds a Java class loader to load the types given as parameters and returns a boolean value. Given that we are now able to adapt the copy constructor template by a simple case distinction such that all fields of a special type will be deep copied.

So far we have implemented the (re)generation of a possibly architectural guided copy constructor. But let's assume there is another architectural constraint, which forces us to adapt the generated implementation depending on the component the input file is associated

with. Thus, we need context variables, which we assume to be extractable from the path respectively package of the input. To the advantage of our approach more and more reference architectures arise, which often prescribe strict naming conventions. So architectural information such as layer or component names are often encoded into file paths respectively package names and thus match our assumptions. How to retrieve context information from a Java input file is shown exemplarily in Listing 3. The context configuration can contain multiple `trigger` definitions, which specify a mapping between input class types—matching the `typeRegex`—and a set of associated meaningful templates—contained in the `tempalteFolder`. For parametrization purposes `variableAssignments` can be specified, which can be assigned to a value of a regular expression group given by the `typeRegex` or to a string value. The variables will be added to the object model for template generation such that our additional architectural constraint dependent on the input’s component can be considered in the templates as well.

```
<trigger id="coreLayerEntity"
        typeRegex=".+\.core\.([^\.]+)\.entity.+"
        templateFolder="entity_templates" >
  <variableAssignment variable="component" regexGroup="1" />
  <variableAssignment variable="var1" value="value" />
</trigger>
```

Listing 3: Trigger example for a context configuration

## 4 Industrial Experiences

The APPS-Generator has been used in three Capgemini projects in the context of very different use cases. In the first project the APPS-Generator has been used to generate an architecture-conform CRUD application with a JSF user interface. The main challenges were the incremental integration of Spring configuration entries within the architectural predefined files, simultaneously merging xHTML files to integrate new navigation elements, and merging newly generated Java fields into existing Java source files.

The second project uses the APPS-Generator in a more fine-grained generation scenario. The hash method, the equals method, and different copy constructors had to be regenerated dependent on the current input’s fields and their types—e.g., for copy constructors as shown in the paper’s example. In addition this scenario required the extension of the FreeMarker template engine to cope with Java type checks.

The third project used the APPS-Generator as a generic development tool. The basis was a huge collection of over 400 Java types—in the following called Hibernate entities—generated with Hibernate from the customer’s existing database. In order to implement the functional requirements of the new system, the Hibernate entities had to be structured within a type hierarchy of commonly usable interfaces. Dependent on the type name and the field names of each hibernate entity, the subtype relation and all inherited methods could be generated right in place. In this use case the APPS-Generator was not even used driven by a standardized architecture, but it was driven by the developer’s needs of automation. Thus the APPS-Generator also provides an interesting framework for defining and using macros and assist the developer in reoccurring tasks.

## 5 Conclusions

As a generic, integrated and fine-grained incremental generation approach the APPS-Generator could be established in the development process of the developer in very different use cases. It has been indicated, that with the approach of fine-grained incremental code generation the efficiency of the developer himself can be improved. This encompasses the traditional generation of separated code as well as the generation of code within existing code currently under development.

Nevertheless, we have observed that especially the structural XML merge mechanisms require further development and research. Among existing generic structural XML merge implementations, an approach has to be developed which is able to take a XML dialect's semantics into account and thus be able to adequately merge two XML documents of the same XML dialect. Furthermore, some efforts should be allocated for extending the given XML and Java parsers to also take comments into account as these are ignored for efficiency reasons in parsers and therefore unfortunately are lost after a structural merge.

## References

- [AvdBSB12] Jeroen Arnoldus, Mark G. J. van den Brand, Alexander Serebrenik, and Jacob Brunekreef. *Code Generation with Templates.*, volume 1 of *Atlantis Studies in Computing*. Atlantis Press, 2012.
- [BFVY96] F. J. Budinsky, M. A. Finnie, J. M. Vlissides, and P. S. Yu. Automatic code generation from design patterns. *IBM Syst. J.*, 35(2):151–171, May 1996.
- [BST<sup>+</sup>94] D. Batory, V. Singhal, J. Thomas, S. Dasari, B. Geraci, and M. Sirkin. The GenVoca model of software-system generators. *Software, IEEE*, 11(5):89–94, 1994.
- [Com13] NetBeans Community. Generating a JavaServer Faces 2.x CRUD Application from a Database - NetBeans IDE Tutorial, October 2013. <https://netbeans.org/kb/docs/web/jsf20-crud.html>.
- [Fou13] The Apache Software Foundation. Apache Velocity Site - The Apache Velocity Project, November 2013. <http://velocity.apache.org/>.
- [HRS<sup>+</sup>98] Frank Heister, Jan Peter Riegel, Martin Schuetze, Stefan Schulz, and Gerhard Zimmermann. Pattern-Based Code Generation for Well-Defined Application Domains. Technical report, In Frank Buschmann, Dirk Riehle (Eds.): Proceedings of the 1997 European Pattern Languages of Programming Conference, Irsee, 1998.
- [Inc10] Red Hat Inc. Hibernate JPA 2 Metamodel Generator, March 2010. [http://docs.jboss.org/hibernate/jpamodelgen/1.0/reference/en-US/html\\_single/](http://docs.jboss.org/hibernate/jpamodelgen/1.0/reference/en-US/html_single/).
- [pro13] FreeMarker project. FreeMarker Java Template Engine - Overview, June 2013. <http://freemarker.org/>.
- [RR13] Nicolas Romanetti and Florent Ramière. SpringFuse - Online Java Code Generator, October 2013. <http://www.springfuse.com/>.



## **Technologietransferprogramm**



## **Vorwort Technologietransfer-Programm der SE 2014**

Eine Besonderheit der universitären Informatik-Forschung in Deutschland ist der rege Austausch mit der Industrie. Beispielsweise kooperieren viele Lehrstühle der Software-Technik an Universitäten mit Firmen; Fraunhofer-Institute sowie Institute verschiedener Bundesländer widmen sich dem Technologie-Transfer. Dies geschieht zum wechselseitigen Nutzen: dem Transfer von neuestem Wissen steht im Austausch ein besserer Verständnis von Randbedingungen des industriellen Einsatzes neuer Methoden, Möglichkeiten zur Real-Welt-Validierung von neuen Ansätzen sowie die Kenntnis aktueller wirklicher Herausforderungen gegenüber, die für die weitere Grundlagenforschung sinnvoll eingebracht werden können. Daher möchte sich das Technologietransferprogramm der SE 2014 dediziert dem Austausch der Erfahrungen im Technologie-Transfer von Software-Technik-Forschung widmen.

Die angenommenen Einreichungen widmen sich verschiedenen Facetten dieses Themas: es reicht von der Diskussion neuer Instrumente des Technologietransfers (wie LivingLabs) bis hin zum Transfer von Werkzeugen für spezifische Themen wie Architekturentwurf; von Erfahrungsberichten aus konkreten Projekten und Lessons Learned bis hin zu Verallgemeinerungen und Einsichten aus mehreren Projekten.

Dies zeigt die Bandbreite dieses Gebietes mit seiner Dynamik. Aus wissenschaftlicher Sicht zeigt sich (wieder einmal), dass der Technologietransfer keine Einbahnstraße ist, sondern auch starke Impulse liefern kann (ja muss) für unsere Disziplin der Software-Technik. Dies gilt meiner Meinung nach für die Forschung in Form neuer zu untersuchenden Hypothesen wie auch für die Lehre mit ihren ständig sich weiter zu entwickelnden Inhalten und dem aus der Praxis gewonnenem Verständnis über best practices.

In diesem Sinne wünsche ich Ihnen eine anregende Lektüre und eine anregende Teilnahme an der SE 2014.

Bedanken möchte ich mich bei allen Mitgliedern des Programmkomitees, der zusätzlichen Gutachter sowie natürlich auch bei den Autoren für Ihre Beiträge!

Ralf Reussner

Vorsitzender des Programmkomitees des Technologietransferprogramms der SE 2014



# Agilität braucht Architektur!

Balthasar Weitzel<sup>1</sup>, Matthias Naab<sup>1</sup>, Mathias Scheffé<sup>2</sup>

<sup>1</sup> Fraunhofer IESE, Fraunhofer-Platz 1, 67663 Kaiserslautern  
{balthasar.weitzel, matthias.naab}@iese.fraunhofer.de

<sup>2</sup> Insiders Technologies, Brüsseler Straße 1, 67657 Kaiserslautern  
m.scheffé@insiders-technologies.de

**Abstract:** Es hat sich gezeigt, dass ein agiler Ansatz, der nur durch Refactorings neue Features in ein Produkt integriert, nicht dazu geeignet ist langfristig Agilität im Projekt zu bewahren. Dabei gerät das Team in einen ständigen Zyklus, bei dem viele Teile des Produktes in jedem Sprint immer wieder angepasst werden. Mit einem überlegten Einsatz von Software-Architektur in der agilen Entwicklung lassen sich wertvolle Vorteile erzielen, die sich sowohl lang- als auch kurzfristig auswirken. Unsere Erfahrungen, die in industrieller Produktentwicklung gesammelt wurden, bestätigen dies eindrucksvoll. Innerhalb einer Kooperation von erfahrenen Praktikern aus der Industrie und Fraunhofer-Forschern in einem R&D-Lab wurde gezeigt, wie solch ein Einsatz von Architektur gestaltet sein kann und welche Effekte damit zu erzielen sind. Hierbei wurde durch vorrausschauende Architektur-Arbeiten sowohl die Produktivität des Teams als auch der Entscheidungsspielraum des Product Owners erhöht. In diesem Beitrag beschreiben wir den konkreten Kontext des R&D-Labs von Insiders Technologies und Fraunhofer IESE und berichten von Erfahrungen des Transfers und der Anpassung von Architekturmethoden in ein agiles Entwicklungsprojekt.

## 1 Erosion von Agilität als wiederkehrendes Muster

In vielen agilen industriellen Softwareentwicklungsprojekten, die über einen längeren Zeitraum laufen, kann eine Erosion von Agilität festgestellt werden. Die Zeichen dafür sind unterschiedlich, häufig jedoch dauert die Realisierung neuer Features im Verlaufe des Projektes immer länger, obwohl deren fachliche Komplexität nicht zunimmt. Eine weitere Beobachtung ist die Zunahme der Kosten für Änderungen an der bereits realisierten Software. In direktem Zusammenhang damit steht eine Vergrößerung der Anzahl der bei einem Feature betroffenen System-Artefakte (siehe Abbildung 1). Es hat sich gezeigt, dass der Verlust von Agilität durch die Erosion der Architektur verursacht wird. Um den Auslöser für diesen Verlust einer klaren Systemarchitektur zu verstehen, lohnt es sich den gesamten Lebenszyklus der Software zu betrachten.

Dieser beginnt meistens mit einem grob umrissenen Kernproblem, das die Software lösen soll. Dafür wird ein initiales Konzept skizziert und zügig umgesetzt. Basierend auf den Erfahrungen, die dabei gemacht werden, wird das Kernproblem erweitert oder sogar

---

Diese Arbeit wurde unterstützt durch das Ministerium für Bildung und Forschung (BMBF) im Programm „Software Campus“, Förderkennzeichen 01IS12053



geändert. Gleichzeitig findet eine Erweiterung der Software statt, um auf diese geänderten Anforderungen zu reagieren. Während agile Entwicklungsprojekte die schrittweise Erweiterung der Anforderungen ausdrücklich akzeptieren und als Stärke ansehen, unterbleiben jedoch meist weitere Überlegungen zur Gesamtarchitektur. Einerseits werden agile Praktiken teils so interpretiert, dass nichts vorausgeplant wird, was nicht Bestandteil des aktuellen Sprints ist, andererseits verhindert Projektdruck die notwendige Investition. Die Folge ist die Entstehung vieler Einzelkonzepte, die als Lösungen für Teilaspekte des geänderten Kernproblems der Software „hinzugefügt“ werden.



Abbildung 1: Erosion der Agilität

Insgesamt steigt dadurch die Komplexität des Systems, da es kein greifbares Gesamtkonzept mehr gibt, sondern viele Teillösungen, die miteinander in Wechselwirkung stehen. Bei jeder Anpassung, die an der Software vorgenommen wird, müssen mehrere Konzepte angefasst und deren Verbindung untereinander verstanden und wiederhergestellt werden. Während agile Entwicklung Refactorings als Mittel mitbringt, um konzeptionelle Inkonsistenzen zu vermeiden, ist es ab einem gewissen Punkt nicht mehr wirtschaftlich vertretbar jedes Refactoring durchzuführen. Durch die Betrachtung von Architektur während der agilen Entwicklung kann die Agilität langfristig bewahrt werden. Im Weiteren soll dies durch die Erfahrungen aus einem aktuellen Projekt illustriert werden.

## 2 Kooperation von Industrie und Forschung

Das Ziel dieses Projektes ist die Neuentwicklung eines Business-Intelligence-Tools für den sogenannten „Document-Entry-Point“. Es existieren keine Bestandskunden für das neue Produkt, jedoch ein bereits durch verwandte Produkte erschlossener Markt, der damit bedient werden soll. Um eine möglichst gute Ausrichtung auf die potenziellen Kunden zu haben, werden diese sehr früh in die Entwicklung einbezogen. In regelmäßigen Workshops werden deren Anforderungen erfasst und der aktuelle Stand des Produktes diskutiert. Daraus resultieren neue User-Stories, die in das Backlog einfließen. Diese kurze Charakterisierung des Projekts kann als typisch für andere vergleichbare Projekte gesehen werden, die nach agilen Methoden durchgeführt werden.

Die Besonderheit dieses Projektes bestand in dem Umfeld, in dem es ausgeführt wurde. Das Entwicklungsteam, das für das Produkt verantwortlich war, entstand durch die Kooperation von Industrie und angewandter Forschung: In einem gemeinsamen Research & Development Lab überprüfen Fraunhofer IESE-Mitarbeiter zusammen mit Industriepartnern Praktiken aus dem Bereich Software Engineering auf ihren praktischen

Nutzen. Das eher heterogene Team bestand zum einen aus langjährigen Entwicklern, die sehr viel Programmiererfahrung ins Team mitbrachten, für die jedoch Software Engineering-Praktiken eher ein Randthema darstellte. Auf der anderen Seite steuerten erfahrene Experten des Software Engineering gezielt Wissen zu eben diesen Praktiken bei, besaßen aber selbst keine langjährige Entwicklungserfahrung in der Domäne der Dokumentverarbeitung. Durch Pairing und intensiven Austausch innerhalb des Teams wurde jedoch eine produktive Gruppe geschaffen, die auch flexibel genug war, um neue Wege zu gehen. Das Research & Development Lab wurde in dieser Konstellation über ein Jahr geführt, so dass auch langfristige Effekte untersucht werden konnten. Ziel ist es dabei, direkt anwendbare Lösungen für solche Probleme zu finden, die in der industriellen Entwicklung häufig auftreten.

### **3 Zeitraubende Refactoring-Zyklen**

Zu Beginn des Projektes wurde nach einem agilen Entwicklungsprozess nach Scrum [SS09] und TDD [B02] gearbeitet, der sich auf Wunsch der industriellen Entwickler relativ dogmatisch an agile Prinzipien hielt, wie zum Beispiel „jede Iteration muss direkten Kundennutzen bieten“. Trotz der guten Zusammenarbeit im Team wurden bald anfänglich beschriebene Probleme sichtbar: Die Geschwindigkeit der Feature-Entwicklung nahm ab, jede neue Story brauchte mehr Aufwand. Es wurde ein regelrechter Refactoring-Zyklus festgestellt, bei dem Teile des Systems mit jedem Sprint wieder umgebaut wurden. Die innere Komplexität der Software stieg immer weiter an, was am besten dadurch sichtbar wurde, dass ein immer größerer Teil des Aufwands, der für jede Story benötigt wurde, in das Erarbeiten eines Realisierungskonzeptes floss. Das liegt in der Anzahl an Einzelschritten begründet, die notwendig waren, um zu einem tragfähigen Ergebnis zu kommen. Ein Großteil der Zeit nahm dabei das Verstehen des momentanen Systems in Anspruch, was durch die Vielzahl der Einzelkonzepte, die dort verwendet wurden, erschwert wurde. Ein weiteres Konzept zu erarbeiten, das die neue Story unter Beachtung von vielen Mechanismen realisieren kann, gestaltete sich immer schwieriger. Die Stories wurden immer mehr zu kleinen Migrationsprojekten, bei denen durch viele Anpassungen die Realisierung eines neuen Features erfolgte. Eine Nebenerscheinung dieses Vorgehens waren Probleme bei der Integration, Teile des Systems verhielten sich zusammen anders als erwartet. Insgesamt wurde intern eine hohe Komplexität wahrgenommen, obwohl die extern sichtbare Fachlichkeit doch eher überschaubar war. Dem Team fiel es immer schwerer den Aufwand für neue Stories zu vertreten, da der Aufwand für die „Integration“ der neuen Features in das bestehende System von außen nicht gesehen wurde und entsprechend schwer darzustellen war. Zielsetzung des R&D Labs ist es, genau solche Engineering Probleme gemeinsam zu analysieren, Gegenmaßnahmen zu erarbeiten und nachhaltig zu implementieren. Deshalb wurden neue Ansätze getestet, um das Problem der Agilitäts-Erosion zu beseitigen. Im Weiteren soll auf den erfolgreichsten dieser Ansätze eingegangen werden.

## 4 Verschiebung des Planungshorizontes

Am einfachsten lässt sich die gewählte Lösung als Verschiebung des Planungshorizontes beschreiben, der im bisherigen Ansatz auf eine Story begrenzt war. Stattdessen werden nun alle Stories betrachtet, deren baldige Realisierung wahrscheinlich ist, was typischerweise alle Stories eines Epics sind.

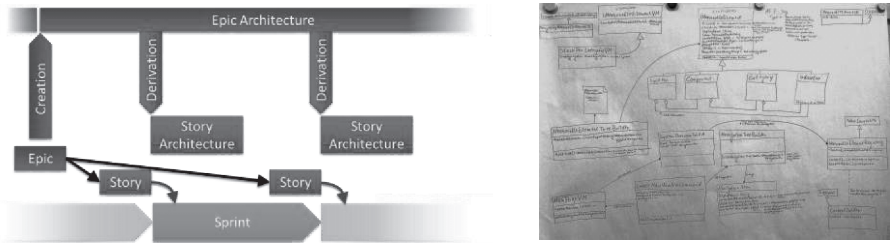


Abbildung 2: Bezug zwischen Epic- und Story-Architektur, Beispiel Epic-Architektur

Eine Art der Planung, die sich dabei als nützlich erwiesen hat, ist die Konzeption einer Systemarchitektur für diese Menge an User-Stories eines Epics. Diese Architektur bezeichnen wir als Epic-Architektur. Dabei ist der notwendige erste Schritt der Entwurf einer geeigneten fachlichen Architektur, die in der Lage ist, eine einheitliche Strukturierung des Problemraumes bereitzustellen. Auf deren Basis kann die fachliche Lösung der User-Story und Wechselwirkungen zwischen einzelnen Stories beschrieben werden, ohne in technische Komplexität abzutauchen. Erst im nächsten Schritt wird die technische Umsetzung auf Basis des momentanen Systems geplant. Dabei findet eine gezielte Abwägung zwischen Umbau und Erweiterung von bestehenden Realisierungen statt. Nur so sind ganzheitliche Architekturentscheidungen möglich, die in ein einheitliches Konzept zur Umsetzung aller User-Stories eines Epics führen.

Um den Aufwand für diese Art der Planung gering zu halten, wird dabei mit einem Team von 2 bis 3 Personen auf einem hohen Abstraktionsniveau gearbeitet, das noch Entscheidungsfreiräume in der Implementierung lässt. Trotzdem enthält die Epic-Architektur alle grundlegenden Entscheidungen, sodass es möglich ist, mittels Design-Walkthroughs potenzielle Fehler in der Konzeption frühzeitig zu entdecken. Als Art der Dokumentation wurde bewusst eine sehr einfache gewählt, die auch ein schnelles Ändern zulässt. Dabei wird bevorzugt am Whiteboard gearbeitet und UML-Diagramme mit zusätzlichen Notationen und Kommentaren erweitert. Der wichtige Aspekt dabei ist die Angemessenheit der Dokumentation für den Zweck, der in dem Fall das Schaffen eines einheitlichen Verständnisses bei allen Teammitgliedern ist. Ein Beispiel für solch eine Dokumentation ist in Abbildung 2 zu sehen. Der entscheidende Unterschied zwischen Systemarchitekturen, die so vorab geplant werden und solchen, die iterativ während der Umsetzung mehrerer Stories entstehen, ist die Art, wie mit Gemeinsamkeiten und Wechselwirkungen zwischen Features umgegangen wird. Bei dem erweiterten Planungshorizont wird explizit Infrastruktur konzipiert, die die effiziente Umsetzung von mehreren Features ermöglicht. Sobald solche gemeinsam genutzten Konzepte erkannt werden, kann darauf auch in der Reihenfolge der Stories im Backlog eingegangen werden. Hierbei können unter fachlich ähnliche priorisierten

Stories solche zuerst realisiert werden, die zum Aufbau einer gemeinsamen Infrastruktur beitragen. Die konkrete Planung der Umsetzung der Stories eines Sprints bleibt nach wie vor erforderlich, gestaltet sich jedoch viel einfacher, da eine Orientierung an der Epic-Architektur erfolgen kann (siehe Abbildung 2). Die Ableitung eines konkreten Umsetzungsplanes einer Story erfolgt dann als Vorbereitung des Teams auf den nächsten Sprint, was in diesem Kontext als Story-Architektur bezeichnet wird. Dabei werden Details, die nicht in der Epic-Architektur vorgegeben sind, ergänzt. Hierbei handelt es sich um technische Realisierungsaspekte, die beispielsweise durch Prototypen erkannt wurden. Weiterhin findet eine Ableitung von konkreten Tasks statt, die dann im Team umgesetzt werden können.

Die Schwierigkeit besteht beim Ableiten von Story-Architekturen darin, eine geeignete Zerlegung des Gesamtkonzepts in realisierbare Einzelschritte zu finden. Auf der einen Seite müssen alle relevanten Aspekte für die aktuellen Stories realisiert werden können, auf der anderen auch ein vertretbares Maß an zusätzlicher Infrastruktur, die eine Vorarbeit für zukünftige Stories darstellt. Dabei muss eine Abwägung zwischen Zusatzaufwand durch „Bauvorleistung“ und späterem gesparten Aufwand durch Entfallen von Refactorings getroffen werden. Nur durch diese Art der Planung ist es möglich, solch eine Entscheidung bewusst zu fällen und belastbare Aufwandsabschätzungen zu Rate zu ziehen. Ein wichtiger Schritt, der nach jedem erfolgreichen Umsetzen einer Story erfolgen sollte, ist das Zurückspielen von neuen Erkenntnissen in die Planung. Damit ist gewährleistet, dass etwaige Abweichungen so gering wie möglich gehalten werden und dass die Realisierung zukünftiger und bereits umgesetzter Stories konsistent bleibt. Im bisherigen Verlauf des Projektes bestanden solche Feedback-Zyklen aus Ergänzungen zu den Konzepten, die in der Epic-Architektur vorgesehen sind. Potenziell wären auch größere Änderungen an der geplanten Architektur möglich, wobei solche Umbrüche genau abgewogen werden müssten, um nicht die Vorteile des einheitlichen Gesamtkonzeptes zu verlieren.

## **5 Günstig agieren statt teuer reagieren**

Die Erfahrungen, die durch das Erweitern des Planungshorizontes hin zu Epic-Architekturen gemacht wurden, sind durchweg positiv – sowohl innerhalb des Entwicklungsteams als auch die extern wahrgenommenen Effekte. Innerhalb des Entwicklungsteams war vor allem die Reduktion des Refactoring-Aufwandes auffallend, der durch das Ausnutzen von Gemeinsamkeiten der Stories gewonnen werden konnte. Infrastrukturkomponenten werden ergänzt, anstatt sie immer wieder zu verändern oder sogar zu duplizieren. Die Verbesserungen in exakte Zahlen zu fassen ist eher schwierig, da innerhalb des Projektes keine Kontrollgruppe aufgesetzt werden konnte, die ohne unseren Ansatz gearbeitet hat. Basierend auf einem Vergleich des Aufwandes für Stories mit einer vergleichbaren Komplexität vor und nach der Einführung des Konzeptes der Epic-Architektur kann eine Reduktion von 25% des Aufwandes festgestellt werden. Das führt zu einer Ersparnis von ungefähr 0,75 Tagen für das gesamte Team. Im Gegensatz dazu steht der Aufwand zur Erstellung einer Epic-Architektur, die zwei Personen für 1,5 Tage beschäftigt hat. Bei der Größe der im Projekt verwendeten Epics von 15 Stories ergeben sich daraus 0,1 Tage für zwei Personen in einer Story, die für die Erstellung der

Epic-Architektur verwendet wurden. Im Vergleich ergibt sich daraus eine deutliche Ersparnis. Als ein Resultat daraus stieg die Zufriedenheit innerhalb des Teams an, was auch der nun verbesserten Möglichkeit zur Orientierung im Epic geschuldet ist. Jedem Teammitglied ist dabei klar, wie mit der momentanen Arbeit zur Erreichung des Epics beigetragen wird und wie offene Design-Entscheidungen getroffen werden müssen, um diesem Ziel näher zu kommen. Extern wurde vor allem eine Steigerung der Umsetzungsgeschwindigkeit wahrgenommen. Die Zeit, die das Team braucht, um selbst eine bisher unbekannte Story des Epics zu planen und zu realisieren, ist wieder ihrer fachlichen Komplexität angemessen. Das frühzeitige Erfassen von Kundenfeedback wurde dadurch in kürzeren Zyklen möglich und die Entwicklung insgesamt als agil wahrgenommen.

## 6 Fazit

Inwieweit die positiven Effekte des Einführens von Epic-Architekturen nur den speziellen Projekteigenheiten geschuldet sind wurde noch nicht abschließend evaluiert. Erste Erfahrungen aus anderen Projekten zeigen jedoch, dass diese durchaus auch in geänderten Kontext anwendbar sind. Als weiterer interessanter Punkt wurde eine verbesserte Einbindung in unterschiedliche Entwicklungsprozesse identifiziert. Eine Integration in Scrum konnte in diesem Projekt bereits erfolgreich durchgeführt werden. Weiterhin besteht noch Potenzial in der Unterstützung der Planung selbst, also dem Erstellen der Epic-Architektur. Die Auswahl der notwendigen Detailtiefe konnte zwar innerhalb dieses Projektes gut gewählt werden, jedoch fehlen noch allgemeine Regeln, die eine Abwägung zwischen benötigter Vorab-Investition und späterer Ersparnis beim Erstellen der Sprint-Planungen beachten. Innerhalb des hier beschriebenen Projektes wurde jegliche Dokumentation der Architektur als UML-Diagramme in Papierform vorgenommen. Dazu ist bereits geplant zu einem modellbasierten Ansatz überzugehen, der vor allem in größeren Projekten bessere Übersichtlichkeit ermöglicht. Die wichtigste Erkenntnis, die in diesem agilen Entwicklungsprojekt gemacht wurde, ist der positive Effekt eines erweiterten Planungshorizontes auf die Agilität des Teams. Als angemessene Größe der Planung hat sich ein Epic bewährt, sodass eine signifikante Menge an User-Stories auf einmal betrachtet werden kann. Wir hoffen, dass solche Erfahrungen auch andere Teams dazu motivieren, aktiv die Entwicklung ihres Produktes zu steuern und damit langfristig ihre Agilität zu sichern.

## 7 Literaturverzeichnis

- [BNO10] Brown, N., Nord, R., Ozkaya, I. (2010) Enabling Agility through Architecture, CrossTalk Nov/Dec 2010.
- [BNO12] Bachmann, F., Nord, R., Ozkaya, I. (2012). Architectural Tactics to support rapid and agile stability, CrossTalk May/June 2012.
- [SS09] Schwaber K., Sutherland J. (2009) Scrum Guide, Scrum Alliance, vol. 19, no. 6, p. 21.
- [B02] Beck K. (2002) Test Driven Development: By Example, Addison-Wesley Professional, p. 192.

# Tool-Driven Technology Transfer to Support Software Architecture Decisions

Heiko Koziolk, Thomas Goldschmidt

Industrial Software Systems  
ABB Corporate Research  
Wallstadter Str. 59  
68526 Ladenburg, Germany  
heiko.koziolk@de.abb.com  
thomas.goldschmidt@de.abb.com

**Abstract:** Software architecture design decisions are key drivers for the success of software systems. Despite awareness for their criticality, software architects often rationalize and document their decisions poorly. On this behalf, ABB Corporate Research initiated a technology transfer project to integrate an architecture decision framework from the University of Groningen into ABB software development processes. The project involved close communication between university researchers, industry researchers, and ABB software architects and resulted in the implementation of a plug-in for the UML tool Enterprise Architect. This paper summarizes success factors for the technology transfer, such as strong buy-in from the stakeholders, short feedback cycles, and seamless integration into existing tool-chains.

## 1 Introduction

ABB Corporate Research faces the issue of transferring software engineering technologies into a large number of diverse ABB business units, which develop software for power systems and industrial automation. In recent years, we have adopted a tool-driven strategy for several software engineering technology transfer projects joint with University partners. This involves for example the creation of Visual Studio plug-ins or Team Foundation Server extensions for ABB's often Microsoft-centric development environments [SDRF12]. Packaging research results into seamlessly integrated development tools can help to make advanced research concepts available to regular developers. This results in a bottom-up, developer-driven transfer and improves the acceptance of the results in the development units.

This paper reports from a tool-driven technology transfer project in the area of software architecture. Architecture design decision research has gained substantial academic interest in the last ten years [BDLV09] (cf. Section 2). Our tooling originates from a published conceptual framework [vHAH12] and was developed in close collaboration with ABB software architects (cf. Section 3). The article summarizes lessons learned while conducting the technology transfer (cf. Section 4).

## 2 Research on Architecture Decision Modeling

Despite the growing complexity of modern software systems and the higher awareness for software architecture concerns, there is still limited, systematic architecture decision documentation in practice [TBGH06]. Decision documentation is seen as time-consuming and not immediately rewarding, as its true value may be realized only in the maintenance phase of a system when decisions need to be changed or augmented. If software architects use UML diagrams to document structural or behavioral aspects of their systems, it is usually cumbersome to annotate these diagrams with rationale for the decisions made. Common UML tools only allow informal textual annotations, which are difficult to manage and maintain if the documentation gets more complex.

A recent stream of research advocates treating architectural decisions as first class entities of the architectural documentation on the same level as components and interfaces [JB05]. Tang et al. [TBGH06] surveyed how architecture design rationale is treated in practice. Kruchten et al. [KLvV06] argued for a repository of architecture knowledge explicitly documenting decisions and their rationale. A couple of knowledge management tools spanning from Wikis, UML profiles, and programming language extensions have been proposed [TAJ<sup>+</sup>10]. Zimmermann et al. created reusable decision models for SOA systems [ZGK<sup>+</sup>07]. The international standard for architecture description (ISO/IEC/IEEE 42010) added architecture decision documentation to its framework in its 2011 revision.

Our technology transfer project originated from one of the recent approaches to architecture decision modeling. van Heesch et al. [vHAH12] proposed a documentation framework for architectural decisions spanning five viewpoints using the conventions of ISO/IEC/IEEE 42010. The relationship viewpoint shows dependencies between decisions, the chronology viewpoint shows different states of decision over time, the stakeholder viewpoint connects stakeholders and decisions, and the detailed viewpoint contains a detailed description and rationalization for a single decision. Finally, the forces viewpoint published in a separate paper shows the decision forces affecting architecture decisions.

As an open problem for the technology transfer, tool support for architecture decision is still limited. Existing tools [TAJ<sup>+</sup>10] have been developed in academic, non-commercial contexts, and are often not well embedded into architecture design processes or existing tool chains. Although there is the ISO/IEC/IEEE 42010 standard, it provides only coarse-grained guidance what attributes of a decision could be documented, but gives no concrete guidelines. For architecture documentation, usually informal diagrams or sometimes UML models are used in practice. The lack of specific tool support for documenting architecture design decisions makes some software architects reluctant to document their decision rationale at all. For the conceptual decision framework from van Heesch et al. [vHAH12], no tool support existed at all.

## 3 From Research to Practice

The goal of our technology transfer project for architecture decision modeling was twofold. First, we wanted to better understand the current practices of software architecture documentation and decision documentation. This included not only finding out what

information was important for software architects to document, but also under what organizational and technical constraints such documentation was created. Second, we wanted to improve the current practice of explicit decision documentation. Therefore, the goal was to create a documentation tool that seamlessly integrated into existing ABB software development processes in order to lower the barrier to document decisions in the future. Besides the concrete tool implementation, involving ABB software architects into the requirements engineering process for the tooling was intended to disseminate the idea of explicit decision modeling from research into practice.

The overall project team consisted of one PhD student and two Master students from the partnering University as well as two researchers from ABB Corporate Research. The project duration was one year. We involved five practicing ABB software architects in our study, which had 10 to 24 years of experience in software development and were all working in projects where they made architecture design decisions. The architects worked in the domain of industrial process automation systems and each designed different products.

To gather requirements for the tooling and better understand the constraints under which the software architects worked, we first conducted a series of semi-structured phone interviews. We asked each architect for the current practice of architecture decision documentation, gave a brief preview of the planned tooling, and asked for specific functional and non-functional requirements. Additionally, we analyzed existing architecture decision documentation from two projects. This for example involved spreadsheet templates, slide sets, and architecture design documents. There was also a document giving architecture modeling guidelines, which we analyzed to find a good way of integrating the decision documentation.

We found that all architects were familiar with the concepts of architecture decision documentation, but that there was no standard way of doing this. Some architects documented decisions in presentation slides or spreadsheets. Others provided decision rationale in informal texts accompanying UML diagrams. Decision rationale was also sometimes only kept in e-mails between different stakeholders or meeting minutes without being integrated into the standard architecture documentation at all.

Most architects used the tool Enterprise Architect from Sparx Systems to model in UML. Some of them had created tool chains (e.g., to generate documents), trainings, and modeling guidelines. Introducing a Wiki- or web-based tooling for documenting architectural decisions was not desirable as it would have added another tool, which required administration as well as synchronizing with the UML models. Therefore, we decided to implement the architecture decision tooling as an add-in to Enterprise Architect. Besides the software architects' familiarity with Enterprise Architect, one reason was to be able to seamlessly connect existing UML models with the decision documentation, which was seen as a great benefit by the software architects.

The software architects added a number of requirements for the tooling, which for example included generation of presentation slides from the model, automatic creating of the chronological decision viewpoint, or the ability to create trace-links between decision alternatives and complete diagrams. The architects also agreed that the tooling should



be flexible and not force the user to document each decision with the same detail. As a major constraint, the architects work under time pressure and are usually not immediately rewarded for complete and high quality documentation. Therefore, complex decision documentation templates were seen critical.

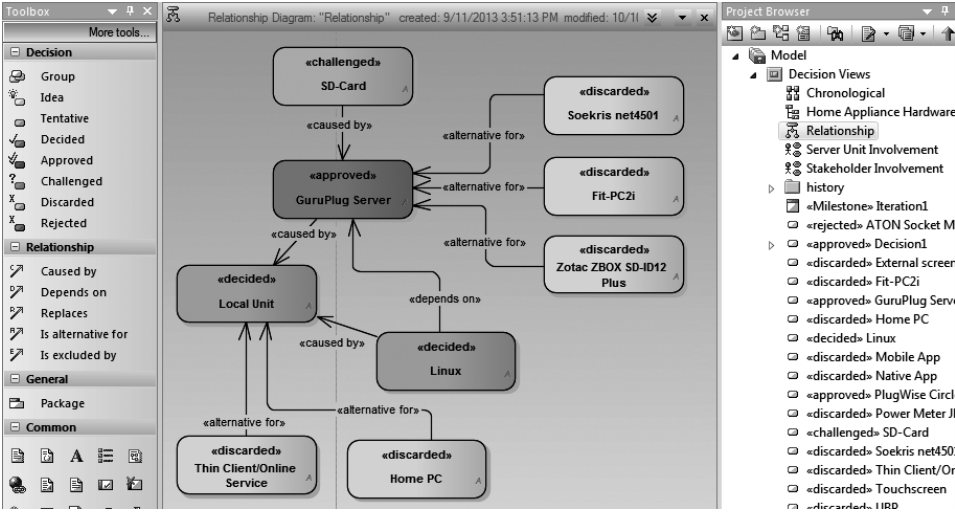


Figure 1: Decision Viewpoint Add-in for Enterprise Architect

Once we had implemented a prototype of the Enterprise Architect add-in, we sent it to the architects and asked them to test it. We then arranged five interview and tool demonstration sessions with each architect each lasting half a working day. We presented the current version of the tooling and asked the architects to document a few decisions from a current project. We observed the participant in using the tool to uncover usability issues. Afterwards, we interviewed the architects for their feedback.

Each architect was generally positive about the tooling, but had different emphasis regarding the different viewpoints. Some favored the forces viewpoint, which allowed a table-like comparison of multiple decision alternatives. Others saw the stakeholder viewpoint as interesting, as it allowed them to trace decisions to particular stakeholders. After using the tooling, the architects requested a change of the decision meta-model, so that the issue, alternatives, and outcome of a decision were captured more explicitly. Besides, they came up with a number of usability improvements.

The final version of the tooling incorporating the software architects' requirements is still under development. After its release, the software architects will use the tooling in their project to retroactively document a number of important decisions. We plan a third interview session with the architects in order to improve the tooling further. Once the Enterprise Architect add-in is a mature state, it is planned to be released as open source software, so that it can be used by other Enterprise Architect users and be extended for additional features by an interested community.

## 4 Lessons Learned

From our technology transfer project, we learned a number of generic lessons that are helpful for similar situations.

**Beneficial to center technology transfer around tooling:** In our case it was very useful to drive the technology transfer through the implementation of a software tool. It forces researchers to make former conceptual work applicable for a larger audience. The tooling pre-packages knowledge on how to structure architecture decision documentation and can provide immediate value to a software architect in a given project. It makes the concepts more accessible than through a research paper. The tooling is still generic and can be used for many projects inside and outside of ABB. ABB is following this tool-driven approach to technology transfer for other concepts as well (e.g., for code search [SDRF12]).

**Different emphasis in academic vs. technology transfer tool development:** The emphasis of tool development in academic contexts is often on creating a proof-of-concept solution that suffices to carry out an empirical validation in a well-protected setting. Instead, the emphasis of tool development in technology transfer projects is rather on process integration and robustness. Existing artifacts (e.g., models), workflows, and tool-chains need to be respected in order to get a buy-in for the tooling. Many of the implemented features provide no academic value (e.g., document generators), but are of high interest for the practitioners. The reliability and usability of the tooling is more important than a comprehensive list of features.

**Short feedback cycles important to get buy-in from users:** It proved very valuable to closely collaborate and communicate with the eventual users of the approach, i.e., the software architects. We presented some of the most promising concept to the software architects early to get their buy-in. The architects valued that we considered their documentation guidelines and templates. We included the architects already in early requirement gathering phases and made sure that their inputs were addressed. The architects were motivated when they saw that the tooling respected their particular issues. We not only presented the tooling to the architects, but also had them experimenting with early prototypes so that they became more familiar with the overall idea.

**Technology transfer as a source for research problems:** While technology transfer per se does not provide novel publishable results besides potential empirical validations, its role in finding new research problems may be underestimated. Through the process of collaboration between researchers and software architects, pointers for future research can be identified. For example, we found that architecture decision documentation is still focused mainly to single products and that a knowledge transfer about decision rationale seldom happens between products. Therefore, it is desirable to enable cross-product decision documentation in the future. Understanding better the issues in practice and the constraints (e.g., time, budget, skills) under which a software engineering approach is applied creates a bi-directional knowledge transfer between research and practice.

**Tool support requires long-term commitment:** If a technology transfer project creates tooling, it must ensure its proper maintenance and evolution after the project has ended. This is often a problem in academic settings, where research projects typically last only 1-3 years and there is little interest in maintenance after the project has finished. The

university collaborations at ABB are usually funded only for a single year, so that the same problem applies. We are thus aiming at making the developed software open source and creating a developer community around it to ensure its evolution. The tooling can also serve as a platform for future tool development for technology as it allows extensions with new concepts. Nevertheless, we deem long-term tool support after technology transfer projects have ended still a hard problem that needs attention from funding committees.

## 5 Conclusions

We have presented the tool-driven technology transfer process ABB Corporate Research applies in selected software engineering University collaborations. As an example, we have created an add-in to a popular UML tool and developed the tooling in close interaction with the target users. Centering the technology transfer around tool implementations brings many benefits such as the need to make conceptual contributions applicable and the ability to quickly benefit from the new concepts. A challenge to this form of technology transfer is the long-term commitment to the maintenance of the tooling, which we try to address by creating an open developer community. In the future we will carry out more such tool-driven technology transfer projects, which have proven to be valuable instrument of bringing advanced software engineering technologies into our organization.

## References

- [BDLV09] Muhammad Ali Babar, Torgeir Dingsyr, Patricia Lago, and Hans Vliet, editors. *Software Architecture Knowledge Management: Theory and Practice*. Springer, 2009.
- [JB05] Anton Jansen and Jan Bosch. Software Architecture as a Set of Architectural Design Decisions. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, WICSA '05, pages 109–120, Washington, DC, USA, 2005. IEEE Computer Society.
- [KLvV06] Philippe Kruchten, Patricia Lago, and Hans van Vliet. Building up and reasoning about architectural knowledge. In *Proc. 2nd Int. Conf. on the Quality of Software Architectures (QoSA'06)*, QoSA'06, pages 43–58, Berlin, Heidelberg, 2006. Springer-Verlag.
- [SDRF12] David Shepherd, Kostadin Damevski, Bartosz Ropski, and Thomas Fritz. Sando: an extensible local code search framework. In *Proc. 20th Int. Symp. on the Foundations of Softw. Eng.*, FSE '12, pages 15:1–15:2, New York, NY, USA, 2012. ACM.
- [TAJ<sup>+</sup>10] Antony Tang, Paris Avgeriou, Anton Jansen, Rafael Capilla, and Muhammad Ali Babar. A comparative study of architecture knowledge management tools. *J. Syst. Softw.*, 83(3):352–370, March 2010.
- [TBGH06] Antony Tang, Muhammad Ali Babar, Ian Gorton, and Jun Han. A survey of architecture design rationale. *J. Syst. Softw.*, 79(12):1792–1804, December 2006.
- [vHAH12] Uwe van Heesch, Paris. Avgeriou, and Rich. Hilliard. A documentation framework for architecture decisions. *J. Syst. Softw.*, 85(4):795–820, April 2012.
- [ZGK<sup>+</sup>07] Olaf Zimmermann, Thomas Gschwind, Jochen Küster, Frank Leymann, and Nelly Schuster. Reusable architectural decision models for enterprise application development. In *Proc. 3rd Int. Conf. on the Quality of Softw. Architectures (QoSA'07)*, QoSA'07, pages 15–32, Berlin, Heidelberg, 2007. Springer-Verlag.

# Individual Code Analyses in Practice

Benjamin Klatt, Klaus Krogmann, Michael Langhammer

FZI Forschungszentrum Informatik, Software Engineering  
Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany

**Abstract:** Custom-made static code analyses and derived metrics are a method of choice when dealing with customer-specific requirements for software quality assurance and problem detection. State-of-the-art development environments (IDE) provide standard analyses for code complexity, coding conventions, or potential bug warnings out-of-the-box. Beyond that, complementary projects have developed common reverse engineering infrastructures over the last years. Based on such infrastructures, individual analyses can be developed for project- and company-specific requirements.

In this paper, we oppose MoDisco to JaMoPP as prevailing infrastructure projects for source code analyses, as well as an approach to gain added value from their combination. We further provide insight into two individual analyses we developed for industrial partners. Those example scenarios and the ongoing development of reverse-engineering infrastructure underline the potential of project-specific analyses, which by now is not exhausted by built-in standard analyses from IDEs.

## 1 Introduction

Ongoing changes in business and usage contexts require continuous adaptation and maintenance of the software systems. According to Seng [SSM06], code quality (e.g. understandability and complexity) and dependency management (e.g. encapsulation and reuse) are success factors for flexible and efficient adaptations. While software grows in size and complexity, taking care of code quality and dependencies becomes expensive and time consuming. Especially with an entirely manual approach, analysis efforts tend to abolish the benefits of the improved adaptability and maintainability.

Over the last decade, several automated static code analyses have been developed to reduce the manual effort for identifying anomalies within a software implementation. Those anomalies range from violated coding conventions up to potential bug detection. Modern software development and management environments provide such analyses out-of-the-box or as extensions. Checkstyle [Che13] and FindBugs [Fin13] are typical tool examples for the Java platform prepared to be generally used by any developer or architect.

However, those widely-used tools are focused on common challenges and problems. Typically, projects and products have also specific and individual challenges to cope with. For example, a project has to reduce its dependency to an unmaintained third party library. Another example is to estimate the change impact of enabling parallel execution (i.e. which source code regions have to run thread safe). Automating such individual analyses does not require a fixed, fully integrated tool, but an infrastructure and know-how to build custom analyses on top of it.

In this paper, we introduce MoDisco and JaMoPP as infrastructures to implement and automate individual analyses. We present their key differences and an approach for their integration to gain added value.

Enabling teams to successfully use such infrastructures is not as easy as applying one of the common code analysis tools. It requires a well-structured approach to start with an initial problem identification and to end-up with a result presentation appropriate for the target audience. We present two example projects from industrial contexts during which we have introduced individual code analyses.

The rest of the paper is structured as follows: In Section 2 we introduce and oppose MoDisco and JaMoPP, followed by some best practices and the presentation of the exemplary industrial projects in Section 3. Finally in Section 4, we conclude our lessons learned and give an outlook how the industrial findings impact our future research.

## 2 Prevailing Technologies in the Eclipse Context

Today, Eclipse is used as an application platform because of its extendability and its support for model-driven software development. On top of this, the MoDisco [BCJM10] and JaMoPP [HJSW09] projects have developed infrastructures for static code analysis. They are both able to extract Abstract Syntax Tree (AST) models from Java resulting in models based on the Eclipse Modeling Framework (EMF) [Ec113b]’s Ecore infrastructure. In the following subsections, we distinguish the projects and present an approach to combine the best of both of them.

### 2.1 MoDisco

MoDisco provides a framework for software model extraction, querying and presentation. The extraction part is strongly related to OMG’s *Knowledge Discovery Metamodel* (KDM) specification. OMG’s Model Driven Software Modernization task force has developed this specification to provide a standard for software models. It covers an AST model concept for different types of programming languages, an architectural knowledge model (e.g. components), and an inventory model for physical artifacts (e.g. files).

MoDisco provides Ecore implementations of these meta models – except the AST one – and thus is promoted as a reference implementation by the OMG. The project’s extendable extraction concept is based on so-called discoverers, each producing software models covering supported types of artifacts. The discoverer provided for Java source code is based on Eclipse’s Java Development Tools (JDT) [Ec113a]. For any further model processing, the models can be persisted by the discoverers. In addition to the discoverers, MoDisco’s query infrastructure allows to implement queries on arbitrary Ecore models using Java, OCL, or XPath. Those queries can be executed either programmatically or through MoDisco’s user interface. The latter provides a flexible model browser, comparable to the EMF tree editor but with additional browsing and categorization capabilities. Browser and result views are provided for those queries. On top of this query infrastructure, MoDisco provides a facet infrastructure to non-intrusively decorate existing meta models with additional classes and attributes. Summarized, facets allow for comfortable presentations of the model analysis queries. Facets are capable to present additional model elements or attributes if a query evaluates to true. Furthermore, attribute values can be set to the result of a query, even for non-boolean ones. Finally, UI customizations can be used to style the model presentation in the model browser (e.g. coloring or icons), depending on a query result.

**Core Concept** From a conceptual point of view, a MoDisco discoverer extracts a model by parsing source code. The resulting model is a representation of logical software ele-

ments of the code, decoupled from the original artifacts. Some discoverers also extract an additional KDM inventory as a decorating model linking logical software elements to their physical artifacts (e.g. file).

**Limitations** The inventory model also contains source code formatting information. This could be used to re-generate the original source code. However, an according reliable model-to-text transformation or generator is not publicly available yet.

## 2.2 JaMoPP

The JaMoPP (acronym for Java Model Parser and Printer) completely describes its intention to parse Java code into a model representation and to print it back into Java code. JaMoPP is based on the textual modeling framework EMF Text [Dev13] and provides an EMF Text syntax specification for the Java language. An Ecore meta model is derived from this specification and combined with an EMF Text specification respectively a derived Ecore meta model for formatting information. As a result, each Java model element is able to carry formatting information to print source code in a specified format.

JaMoPP provides an according printer as an integrative part. If a software model element has no formatting attached, it is printed in a predefined format.

According to the EMF Text infrastructure, the JaMoPP parser and printer infrastructure are tightly coupled with the EMF resource concept. They are registered as EMF resource reader and writer for `.java` file extension.

**Core Concept** The JaMoPP core concept is to treat Java source code as a textual representation of a Java model instance. It is tightly coupled with the EMF resource infrastructure to provide a rich and reliable API. The later is ensured by an extensive test suite [HJSW09]. The availability of parser and writer enable round-trip cycles.

**Limitations** The core JaMoPP tooling does not provide any further processing of the extracted software model. Queries or transformations must be implemented individually but can make use of EMF-compatible infrastructure.

## 2.3 Comparison

In this section, we summarize the key characteristics distinguishing JaMoPP and MoDisco for individual code analyses. We do not claim for completeness nor for other use cases.

The query and facet infrastructure of MoDisco provides an easy-to-use UI which is also an advantage to promote analyses to development teams. JaMoPP provides no support in this direction. MoDisco publishes performance benchmarks for their discoverers, which is not available for JaMoPP. Also a performance comparison of the two is not available yet. While possible in theory, no reliable code generation is available for MoDisco yet. JaMoPP provides a printer out of the box. From our experience, JaMoPP preserves individual code formatting without any issues. The MoDisco discoverers do not provide the opportunity to influence persistence options of XMI resources. For larger software models, this can lead to non-optimized memory and storage usage. JaMoPP is more lightweight and leaves the EMF resource configuration up to the developer.

MoDisco makes at least partially use of a standardized meta model while JaMoPP completely relies on a proprietary specification. However, for the analyses, the AST model is the most important part which is proprietary in MoDisco as well. From our personal experience, we discovered some lacks in MoDisco's AST model extraction (e.g resolving labeled statement references, and handling of qualified type accesses) but had no issues

with the JaMoPP extraction. From our personal experience, the MoDisco project lacks support for bug fixes. Even provided patches are not integrated into the project. In contrast, we received quick and helpful responses from the JaMoPP project.

## 2.4 MoDisco-JaMoPP Integration

As outlined in the comparison above, on the one side, JaMoPP provides a more reliable and lightweight extraction (parser). Additionally, the code generation (printing) out-performs MoDisco’s capabilities. On the other side, MoDisco provides a powerful infrastructure for queries and model decoration. Due to the query and facet infrastructure’s applicability to Ecore models in general, they are not limited to models extracted by MoDisco discoverers. As illustrated in Figure 1, we have integrated JaMoPP’s parsing and printing capabilities, with MoDisco’s query and presentation features. Due to the common Ecore infrastructure, this integration can be used without any tool adaptation.

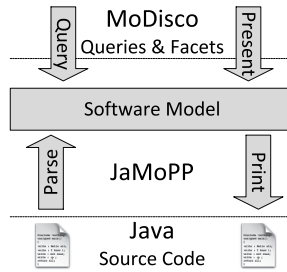


Figure 1: JaMoPP - MoDisco Integration

**Illustrating Example Singleton Analysis** To give an idea about the straight forward implementation of MoDisco queries for JaMoPP-parsed Java models, we have prepared an example to analyze a software for instances of the Singleton pattern (see [GHJV95] page 127ff). We have published this as an Eclipse project on GitHub <sup>1</sup>.

In [Mar13], Lars Martin presents an example using MoDisco to detect Singleton Patterns in Java source code based according to an algorithm described in [Nau01]. A class is identified as singleton if it has i) a static field typed with itself or one of its subclasses, ii) a static getter, and iii) a private constructor.

We have migrated this singleton-detection-query to analyze JaMoPP-extracted software models. In addition, we have created a MoDisco facet representing instances of the Singleton pattern as described by Martin [Mar13].

Thanks to JaMoPP’s concept of treating source code as a textual representation of a model, you can drag a Java file and drop it into the MoDisco Model Browser. Activating the facet for the Singleton pattern, the MoDisco model browser shows instances of the Singleton pattern in the file. The screenshot in Figure 2 presents the result of analyzing the exemplary singleton in Listing 1.

<sup>1</sup> <https://github.com/kopl/misc/tree/master/examples/org.kopl.jamopp.modisco.pattern.query>

```

package org.kopl.singleton.example;
public class MySingleton {
    private static MySingleton instance
        = new MySingleton();
    private MySingleton() {}
    public static MySingleton getInstance() {
        return instance;
    }
}

```

Listing 1: Analyzed Singleton Example

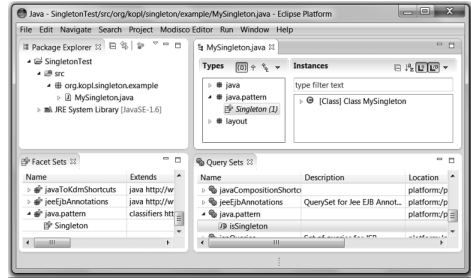


Figure 2: Screenshot Singleton Result

### 3 Best Practices in Analyses

Applying standard, strongly integrated code analyses is common practice today. Supporting individual challenges with custom code analyses is still not completely understood yet. As far as our experience with industrial projects goes, development teams either do not know about today’s possibilities for code analyses at all or do not know how to apply them in their own use case.

We follow four major steps: 1. Problem analyses, 2. Query design, 3. Presentation and KPIs, 4. Actions derivation. To ensure the development of a useful and expressive analyses, first, the problem and analysis goals must be understood and structured to answer the right questions later on. Next, the query must be designed to consider the right elements (e.g. expressions or types) and to produce the appropriate type of results (e.g. true/false or lists of elements). In addition, the analysis-algorithm must be designed with respect to correctness, precision, and performance. Afterwards, an appropriate result presentation for the intended target audience must be developed. The presentation can range from element lists, to visual diagrams, to aggregated Key Performance Indicators (KPIs). Finally, in nearly all cases analyses are performed as a preparation for further actions. Actions include refactoring decisions, task lists, or automated software modifications.

While presented in a strict order, the process must be done in an iterative manner. Meaning that each previous step should be considered again if a potential improvement is detected in a latter one, if the system evolves or quality requirements change.

We have successfully applied this process in a broad range of industrial applications. For example, in a project in the automation industry, we analyzed a software’s dependency to third party libraries. As described in [KDK<sup>+</sup>12], we have developed individual analyses to provide KPIs, beside others, about the adaptability and future perspective of a software system. As a completely different example, we have developed code analyses to accompany a company during a software migration from EJB 2 to EJB 3. In this use case, the analyses have been used to identify new hotspots of potential problem patterns, which come up during the migration. While the migration of the second example is done now, the analyses developed in the first project example, especially the KPIs, are nowadays embedded in some of the company’s business units.

In all cases, the automation lowers the effort for analyzing software systems. Hence, larger software systems can be analyzed more frequently and more thoroughly.



## 4 Conclusion and Outlook

In this paper, we have introduced MoDisco and JaMoPP as Eclipse-based state-of-the-art reverse-engineering infrastructures for Java. We opposed the project's tools and presented a value-adding integration of both of them. Furthermore, we gave examples of industrial applications for individual code analyses.

The results of those analyses lead to ongoing research in this area. We investigate especially in the combination and reuse of existing reverse-engineering tools, code analysis infrastructures, and model-driven modernization techniques.

The experience from industrial applications has led to the KoPL project<sup>2</sup>. The project aims to provide automated support for consolidating customized product copies into a common flexible and sustainable software product line.

## References

- [BCJM10] Hugo Bruneliere, Jordi Cabot, Frédéric Jouault, and F. Madiot. MoDisco: a generic and extensible framework for model driven reverse engineering. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 173–174. ACM, 2010.
- [Che13] Checkstyle Team. Checkstyle. <http://checkstyle.sourceforge.net/>, 2013.
- [Dev13] DevBoost. EMF Text. <http://www.emftext.org/>, 2013.
- [Ecl13a] Eclipse Foundation. Eclipse JDT – Java Development Tools. [www.eclipse.org/jdt/](http://www.eclipse.org/jdt/), 2013.
- [Ecl13b] Eclipse Foundation. Eclipse Modeling Framework Project (EMF). <http://www.eclipse.org/modeling/emf/>, 2013.
- [Fin13] FindBugs Development Team. FindBugs – Find Bugs in Java Programs. <http://findbugs.sourceforge.net/>, 2013.
- [GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1995.
- [HJSW09] Florian Heidenreich, Jendrik Johannes, Mirko Seifert, and Christian Wende. Closing the gap between modelling and java. In *Proceedings of the Second international conference on Software Language Engineering (SLE'09)*, pages 374–383. Springer-Verlag Berlin, Heidelberg, 2009.
- [KDK<sup>+</sup>12] Benjamin Klatt, Zoya Durdik, Heiko Kozirolek, Klaus Krogmann, Johannes Stammel, and Roland Weiss. Identify Impacts of Evolving Third Party Components on Long-Living Software Systems. In *2012 16th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 461–464, Szeged, Hungary, March 2012. Ieee.
- [Mar13] Lars Martin. Tanz unter der MoDisco Kugel. *Eclipse Magazin*, 06.13:38–44, 2013.
- [Nau01] Sebastian Naumann. *Reverse- Engineering von Entwurfsmustern*. Diploma thesis, TU Illmenau, 2001.
- [SSM06] Olaf Seng, Frank Simon, and Thomas Mohaupt. *Code Quality Management*. dpunkt Verlag, Heidelberg, 2006.

---

<sup>2</sup><http://www.kopl-project.org>

# FZI House of Living Labs – interdisziplinärer Technologietransfer 2.0

Stefan Hellfeld

FZI House of Living Labs  
FZI Forschungszentrum Informatik  
Haid-und-Neu-Str. 10-14  
76131 Karlsruhe  
hellfeld@fzi.de

**Abstract:** Das FZI House of Living Labs ist eine neuartige Forschungsumgebung, die den Forschungsansatz der Living Labs inklusive der partizipativen Forschung mit Partnern aus der Industrie aufgreift, wesentlich erweitert und so den interdisziplinären Technologietransfer über Domängengrenzen hinweg unterstützt. Innovationen, die vornehmlich an den Schnittstellen der Disziplinen entstehen, können durch den domänenübergreifenden Austausch der im FZI House of Living Labs arbeitenden und forschenden Experten aus Wissenschaft und Wirtschaft vereinfacht identifiziert werden. Darüber hinaus wird der Technologietransfer um den Einbezug der Endanwender wesentlich erweitert. Im FZI House of Living Labs wird nicht nur punktuell für einen spezifischen Teilprozess oder ein Teilgebiet in einem einzigen Living Lab partizipativ geforscht. Vielmehr werden vollständige Abläufe des alltäglichen Lebens mittels mehrerer Living Labs abgebildet und entsprechend findet der Technologietransfer anwenderbezogen und domänenübergreifend statt. Der interdisziplinäre Technologietransfer im FZI House of Living Labs hat darüber hinaus die Funktion eines Innovationsmotors, da Innovationen aufgrund der geschaffenen Dialogsituationen zwischen Wissenschaft, Wirtschaft und Gesellschaft im FZI House of Living Labs vereinfacht generiert werden können.

## 1 Einleitung

Der Forschungsansatz der Living Labs wird seit mehreren Jahren in der Forschung erfolgreich angewandt, um Technologietransfer zu betreiben [Bal05, Guz13, HrK13, MuS13]. Living Labs ermöglichen partizipative Forschung von Experten<sup>1</sup> der Wissenschaften gemeinsam mit Experten der Wirtschaft sowie eine vereinfachte Integration des Endanwenders in den Entwicklungsprozess einer zu entwickelnden Lösung (z.B. einer neuen Technologie, eines neuen Produktes oder einer neuen Dienstleistung). In einem Living Lab werden, in einer der Realwelt nachempfundenen Umgebung, Fragestellungen einer Anwendungsdomäne untersucht und bearbeitet. Dabei wird meist ein spezifischer Ausschnitt der Realität in einem Living Lab nachgebildet.

---

<sup>1</sup> Aufgrund der Einfachheit wird im Beitrag lediglich die männliche Form verwendet. Die weibliche Form ist selbstverständlich eingeschlossen.

Eine Verknüpfung verschiedener Living Labs hin zu einer Forschungsumgebung, die einen möglichst großen Bereich der Realwelt darstellt, existierte bislang nicht. Demzufolge konnte ein vollständiger Tagesablauf eines Endanwenders in einer einzigen Forschungsumgebung nicht abgebildet werden. Eine derartige Abbildung wird in der Forschungsumgebung des FZI House of Living Labs realisiert. Darüber hinaus wird in der Forschungsumgebung der aktive Austausch der Beteiligten aus Wissenschaft, Wirtschaft und Gesellschaft wesentlich unterstützt. Infolgedessen erlaubt das FZI House of Living Labs eine neue Form des interdisziplinären Technologietransfers – eine zweite Version des Technologietransfers.

Der Beitrag ist wie folgt aufgebaut. In Abschnitt zwei werden der Forschungsansatz des Living Lab und das FZI House of Living Labs sowie der Technologietransfer in den einzelnen Living Labs vorgestellt. Abschnitt drei beinhaltet die Erläuterung der neuartigen Form des Technologietransfers im FZI House of Living Labs. Mit einer Zusammenfassung in Abschnitt vier schließt der Beitrag.

## **2 Der Forschungsansatz Living Lab und das FZI House of Living Labs**

Unter dem Forschungsansatz "Living Lab" versteht man das partizipative Forschen an Fragestellungen der Zukunft mit Einbezug der zukünftigen Anwender der zu entwickelnden Lösungen. William J. Mitchell benutzte den Begriff Living Lab erstmalig im Zusammenhang mit der Einbindung von Bürgern in die städtische Planung [Mit03]. Gegenwärtig werden Living Labs in der Forschung unterschiedlicher Anwendungsdomänen eingesetzt (z.B. im smart Home Bereich zur Untersuchung von Gebäudeautomatisierungstechnologien). Eines der größten Netzwerke aus Living Labs ist das European Network of Living Labs (ENoLL)<sup>2</sup>, in welchem über 270 verschiedene Living Labs vertreten sind.

Existierende Living Labs fokussieren spezifische Problemstellungen in einer einzelnen Anwendungsdomäne. Dies bedeutet, dass Abläufe, die über die Living Lab Grenzen bzw. die Grenzen der Anwendungsdomäne des Living Lab hinausreichen, nicht oder nur sehr schlecht im Living Lab adressiert werden können. Diese Problemstellung wurde im FZI House of Living Labs aufgegriffen und mit der Installation verschiedener FZI Living Labs in einer Umgebung und einer neuen Form des Technologietransfers gelöst.

Das FZI House of Living Labs ist ein zweistöckiges Gebäude, in welches sieben verschiedene, thematisch aneinander angrenzende FZI Living Labs, mehrere Besprechungsräume, Dialogzonen und Büroräumlichkeiten integriert sind. Im FZI House of Living Labs wird in folgenden FZI Living Labs Technologietransfer betrieben.

---

<sup>2</sup> Das European Network of Living Labs ist ein Verbund aus über 270 europäischen Living Labs, die sich mit unterschiedlichen Themen auseinander setzen wie z.B. Energieeffizienz, Gesundheit und Wohlergehen, Tourismus, etc. (<http://www.openlivinglabs.eu/>).

## 2.1 Technologietransfer in den FZI Living Labs

Der Technologietransfer in den FZI Living Labs ist domänenspezifisch und auf das einzelne FZI Living Lab beschränkt, wird aber durch die Integration in das FZI House of Living Labs erweitert (vgl. Abschnitt 3).

### *FZI Living Lab smartAutomation*

Im FZI Living Lab smartAutomation werden intelligente hochleistungsfähige Automatisierungssysteme für industrielle Anwendungen sowie deren Wartung und Interaktion mit dem Menschen erforscht und entwickelt. Das Lab ist darüber hinaus Testlabor für die standardisierte Kommunikationstechnologie PROFIBUS. Im Rahmen des Technologietransfers können sich Unternehmen im FZI Living Lab mit der Kommunikations-technologie PROFIBUS auseinandersetzen und darüber hinaus eigene Lösungen in das vorhandene Kommunikationsnetzwerk integrieren und anschließend selbstständig evaluieren oder evaluieren lassen.

### *FZI Living Lab smartEnergy*

Im FZI Living Lab smartEnergy wird das Energiesystem der Zukunft entwickelt. Dieses System beschränkt sich nicht nur auf die effiziente Nutzung thermischer und elektrischer Energie im FZI House of Living Labs, sondern es wird auch die intelligente Nutzung von erzeugter Energie aus erneuerbaren Ressourcen erforscht. Unternehmen besitzen im Rahmen des Technologietransfers bspw. die Möglichkeit, das FZI Living Lab als Schulungsumgebung zu nutzen. Darüber hinaus kann das Zusammenspiel von im Unternehmen entwickelter Lösungen mit existierenden Sensoren und Aktoren im FZI Living Lab evaluiert werden.

### *FZI Living Lab smartHome/Ambient Assisted Living (AAL)*

Das FZI Living Lab smartHome/AAL ist eine Forschungsinfrastruktur in Form einer Zwei-Zimmer-Wohnung mit zahlreichen Sensoren, Aktoren und Schnittstellen sowie Haushaltsgeräten verschiedener Hersteller. Im FZI Living Lab smartHome/AAL werden Szenarien des Wohnens älterer Menschen, unterstützt durch Informationstechnologie (IT), sowie die Nutzung von Gebäudeautomatisierungstechnologien zum Energiemanagement erforscht. Unternehmen können im Rahmen des Technologietransfers eigene Lösungen in die existierende Umgebung integrieren und entsprechend gemeinsam mit den Anwendern im FZI Living Lab evaluieren.

### *FZI Living Lab mobileIT/mobileBusiness*

Im FZI Living Lab mobileIT/mobileBusiness werden neue Technologien und Lösungen erforscht und entwickelt, die Unternehmensmitarbeitern situationsabhängige Informationen zur Verfügung stellen und sie bei ihrer Aufgabenerfüllung im Unternehmen und unterwegs mittels mobiler IT unterstützen. Das FZI Living Lab stellt für den Technologietransfer bspw. eine Lokalisierungsinfrastruktur zur Verfügung, die basierend auf verschiedenen Technologien eine Lokalisierung im FZI House of Living Labs zulässt. Unternehmen können diese Infrastruktur zum Testen ihrer Anwendungen nutzen.

### *FZI Living Lab SmartMobility*

Im FZI Living Lab smartMobility wird an neuen Formen der Mobilität geforscht. Neben

der Elektromobilität und damit einhergehenden Verbesserungen im Bereich der Routenplanung etc. werden auch Modelle intermodaler Mobilität sowie die Unterstützung einer Mobilität auf der letzten Meile untersucht. Die existierenden Mobilitätsplattformen können zur Entwicklung spezifischer Dienstleistungen oder Produkten von Unternehmen im Rahmen des Technologietransfers genutzt werden.

#### *FZI Living Lab Service Robotics*

Im FZI Living Lab Service Robotics werden neue, robotische Basistechnologien zur Umwelterfassung, Robotersteuerung, Greifplanung und Mensch-Roboter-Interaktion entwickelt, die anschließend in verschiedenen Anwendungsszenarien eingesetzt werden. Entsprechend werden die Plattformen zur Evaluation von Lösungen, die von Unternehmen entwickelt wurden, den Unternehmen zur Evaluation oder zu Testzwecken zur Verfügung gestellt.

#### *FZI Living Lab Automotive*

Im FZI Living Lab Automotive werden Komponenten für Fahrzeuge kommender Generationen entwickelt, erforscht und evaluiert. Dabei gilt es, das Fahrzeug der Zukunft mittels IT intelligenter zu machen und im Hinblick auf unterschiedliche Szenarien zu erweitern (z.B. autonomes Fahren oder Erfassung der Fahrerintention). Unternehmen können Lösungen simulativ im Fahrsimulator oder in der Realität im autonomen Fahrzeug des FZI entwickeln und/oder evaluieren.

### **3 Interdisziplinärer Technologietransfer 2.0 im FZI House of Living Labs**

Das FZI House of Living Labs erlaubt durch die Integration der verschiedenen FZI Living Labs in eine einzige Forschungsumgebung eine verbesserte, effizientere Form des Technologietransfers, die sich im Wesentlichen durch drei spezifische Eigenschaften des FZI House of Living Labs erklären lässt.

#### **3.1 Interdisziplinarität**

Das FZI House of Living Labs unterscheidet sich von anderen Forschungsumgebungen durch die Vielzahl an Domänenexperten aus Wissenschaft und Wirtschaft, die in den Räumlichkeiten zum Dialog zusammengebracht werden. Interdisziplinäre Diskussionen und Ideen, die in den FZI Living Labs an der Quelle der Problemstellung angestoßen werden, können abseits der realitätsnahen Umgebungen zunächst in Dialogzonen und Besprechungsräumen in der Theorie bearbeiten werden. Im Anschluss kann wiederum in den FZI Living Labs die entwickelte Theorie in prototypischer Form in der realitätsnahen Umgebung auf Praxistauglichkeit mit Anwendern evaluiert werden. So werden im FZI House of Living Labs bspw. spezifische IT-gestützte Lösungen einer Domäne in eine andere Domäne transportiert. Demzufolge ergeben sich gänzlich neue Einsatzszenarien für existierende IT-gestützte Lösungen. Zum Beispiel werden unterschiedliche IT-gestützte Lösungen, die bisher den Gesundheitszustand älterer Menschen in deren Haushalten überwacht haben, im Fahrzeug zur Unterstützung und

Überwachung des Gesundheitszustandes des Fahrers verwendet.

Die Interdisziplinarität ist Nährboden für die Generierung von Innovationen, da diese vermehrt an den Schnittstellen der Disziplinen entstehen. Um den kreativen Prozess der Innovationsgenerierung über die Disziplinen hinweg zu unterstützen, wurde ein Portfolio an Dienstleistungen (sogenannte HoLL-Services) innerhalb des FZI House of Living Labs definiert. Diese Dienstleistungen unterstützen die Entwicklung von anwenderbezogenen Lösungen und halten den interdisziplinären Dialog über die einzelnen Entwicklungsphasen hinweg aufrecht (vgl. Abbildung 1).

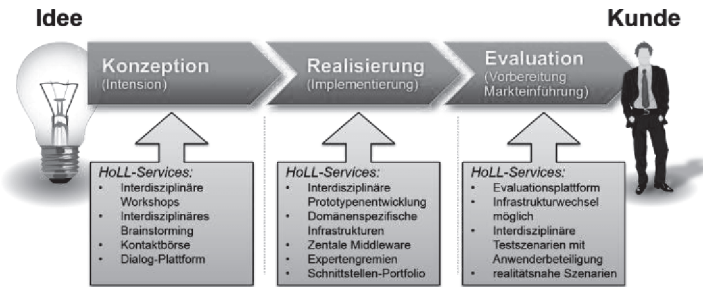


Abbildung 1: Auszug der Dienstleistungen im FZI House of Living Labs

### 3.2 Integrierte Büroräumlichkeiten als Living Labs

Das FZI House of Living Labs besitzt in die Forschungsumgebung integrierte Büroräumlichkeiten, die ebenfalls mit intelligenter IT ausgestattet sind. In diesen Büroräumlichkeiten im FZI House of Living Labs sind die FZI Mitarbeiter parallel zu ihrem Tagesgeschäft an ihrem Arbeitsplatz auch Probanden und dienen der Forschung. So kann bspw. in Langzeitstudien der Energieverbrauch eines Büroangestellten ermittelt werden. Die FZI Mitarbeiter werden dabei in der Ausführung ihres Tagesgeschäftes nicht gestört und sind über die Datenerfassung informiert. Durch die intelligente Vernetzung und vollständige Integration in die Forschungsumgebung ist es so möglich, die Nutzung einer Kaffeemaschine oder der Kochmöglichkeiten im FZI Living Lab smartHome/AAL in das Energieprofil eines einzelnen FZI Mitarbeiters, der die Geräte in der Mittagspause nutzt, miteinzubeziehen. Diese Daten werden dann im Rahmen des Technologietransfers in Kooperationen mit Unternehmen zur Entwicklung unterschiedlicher IT-gestützter Lösungen genutzt (z.B. der Erweiterung einer Kaffeemaschine, die gekoppelt an den Arbeitsplatzrechner der Mitarbeiter erst mit dem Starten des Rechners aus dem Standby erwacht).

### 3.3 Domänenübergreifende Szenarien

Mittels der FZI Living Labs und der räumlichen Struktur des FZI House of Living Labs ist es möglich, den gesamten Tagesablauf eines Menschen lab-übergreifend darzustellen. Vereinfacht ist dies in Abbildung 2 inkl. der verwendeten FZI Living Labs dargestellt. Die vollständige Abbildung eines Tagesablaufs erhöht die Integrationstiefe und die

Akzeptanz eines Anwenders in der Evaluation einer Lösung. Der Anwender muss sich nicht nur auf einen Ausschnitt der nachgebildeten Realität konzentrieren, sondern kann einen vollständigen Tagesablauf in den FZI Living Labs "durchleben".

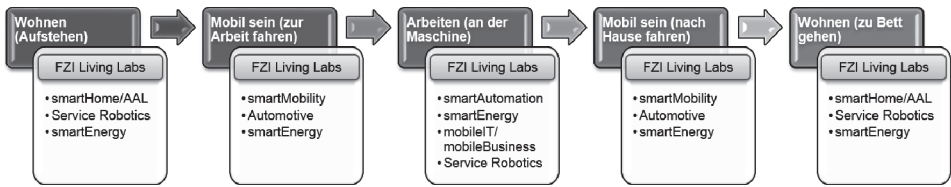


Abbildung 2: Tagesablauf eines Menschen und zugehörige FZI Living Labs

Diese vollständige Abbildung ist auch Grundlage für die Entwicklung domänenübergreifender Lösungen. Ein Beispiel sind Assistenzsysteme in Form von Smartphone-Apps, die Anwender in jeder Phase ihres Tagesablaufs unterstützen. Diese Apps können im FZI House of Living Labs in den einzelnen Umgebungen (in der Wohnung, im Fahrstuhl, in der Garage, im Fahrzeug, bei der Arbeit, etc.) evaluiert und verbessert werden.

## 4 Zusammenfassung

Mit dem FZI House of Living Labs wurde eine Forschungsumgebung geschaffen, die mittels der Interdisziplinarität, der integrierten Büroräumlichkeiten sowie domänenübergreifender Szenarien und der damit verbundenen annähernd vollständigen Abbildung der Realwelt eine neue Form des interdisziplinären Technologietransfers ermöglicht. Darüber hinaus können Querschnittsthemen wie bspw. IT-Sicherheit im FZI House of Living Labs vereinfacht adressiert werden, da sich die Experten der einzelnen Domänen besser austauschen können. Im Rahmen einer kritischen Bewertung des Forschungsansatzes FZI House of Living Labs muss der koordinative Aufwand des Zusammenführens der Domänenexperten bzw. das Finden einer gemeinsamen Austauschform/Sprache der Domänenexperten erwähnt werden. Eben dieser Austausch ist es aber, der die Generierung von Innovationen wesentlich fördert.

## Literaturverzeichnis

- [Bal05] Pieter Ballon et al: Test and Experimentation Platforms for Broadband Innovation, SMIT, Brussels, 2005
- [Guz13] Guzman, J. G., et al. "Living Labs for User-Driven Innovation: A Process Reference Model." Research-Technology Management 56.3, 2013; S. 29-39
- [HrK13] Hronszky, I.; Kovács, K.: Interactive Value Production through Living Labs. Acta Polytechnica Hungarica 10.2, 2013.
- [Mit03] Mitchell, W. J.: Me++ : the cyborg self and the networked city, MIT Press, Cambridge, Mass., 2003.
- [MuS13] Mulvenna, M.; Suzanne M.: Living Labs: Frameworks and Engagement. Innovation through Knowledge Transfer 2012. Springer Berlin Heidelberg, 2013; S. 135-143

# Management operativer Logistikprozesse mit Future-Internet-Leitständen: Erfahrungen aus dem LoFIP-Projekt

Andreas Metzger, Philipp Schmidt, Christian Reinartz, Klaus Pohl

Paluno (The Ruhr Institute for Software Technology)  
Universität Duisburg-Essen, Gerlingstraße 16, 45127 Essen  
<vorname>.<nachname>@paluno.uni-due.de

**Abstract:** Die robuste und ressourceneffiziente Prozessdurchführung zählt zu den bedeutendsten Herausforderungen der Logistikbranche. Diese Herausforderungen adressiert das HighTech.NRW-Projekt LoFIP. Gemeinsam mit Marktführern im Logistikbereich werden neuartige Future-Internet-Leitstände für die Überwachung und Steuerung operativer Logistikprozesse entwickelt. Unsere Vorgehensweise zur Unterstützung des Wissenstransfers aus Forschung in die industrielle Praxis zeichnet sich durch drei Kernaspekte aus: (1) Szenariobasierte Erhebung von Anforderungen, (2) Erlebbarmachen im Logistik-Living-Lab und (3) frühzeitige Validierung mit Domänenexperten. Dieser Beitrag beschreibt diese Kernaspekte und geht auf unsere Erfahrungen mit der Vorgehensweise ein.

## 1. Einleitung

Der Jahresumsatz der Transport- und Logistikbranche macht ca. 15% des EU-Bruttoinlandsprodukts aus. Die CO<sub>2</sub>-Emissionen machen global 14% der Gesamtemissionen aus. Die robuste und ressourceneffiziente Durchführung von Logistikprozessen zählt somit zu einer der bedeutendsten Herausforderungen der Branche.

Vor diesem Hintergrund erarbeitet das Projekt LoFIP („Logistik Future-Internet-Plattform“; [www.lofip.de](http://www.lofip.de)) neuartige IT-Lösungen für die Überwachung und Steuerung operativer, d.h. real laufender, Logistikprozesse. Als Lösungsansatz adressiert LoFIP *software-basierte Leitstände*, welche Future-Internet-Technologien zur Unterstützung von Geschäftsprozessen (Internet der Dienste [NGM08]) und zur Überwachung und Steuerung physikalischer Prozesse und Materialströme (Internet der Dinge [MF10]) verknüpfen. Die LoFIP-Leitstände bieten somit Logistik-Dispositionen in Echtzeit Hinweise auf Probleme oder Optimierungspotenziale, schlagen geeignete Handlungsalternativen vor und erlauben den steuernden Eingriff in laufende Prozesse.

LoFIP-Leitstände sind komplementär und ergänzend zu existierenden IT-Systemen zur Prozessausführung, wie z.B. SCADA oder ERP [Bil07]. Durch die angestrebte Kombination des Internet der Dienste mit dem Internet der Dinge grenzen sich LoFIP-Leitstände zusätzlich von bisherigen Leitständen für das Geschäftsprozess-Management ab (z.B. BAM [Nes04]). In der Literatur existieren erste Ansätze zur Integration des Internet der Dinge in Geschäftsprozesse, allerdings gehen diese Ansätze von der Annahme aus, dass Geräte im Internet der Dinge aktiv vom Prozess aufgerufen und analog



zu anderen Prozessressourcen (z.B. Menschen) genutzt werden können [MRM13]. Aufgrund der Dynamik und der rasanten Zunahme von Geräten im Internet der Dinge ist absehbar, dass diese Annahme zu einschränkend ist. Löst man diese Annahme folglich auf, ergibt sich eine neue software-technische Herausforderung: Wie können vom Internet der Dinge gelieferte Beobachtungsdaten zur Laufzeit korrekt der betroffenen Prozessinstanz und letztlich korrekt der richtigen Prozessaktivität zugeordnet werden?

LoFIP wird vom Land Nordrhein-Westfalen im Rahmen des Ziel2-Programms High-Tech.NRW gefördert ([www.ziel2.nrw.de](http://www.ziel2.nrw.de)). Das Programm zielt darauf ab, Beiträge zur Weiterentwicklung der Leit- und Zukunftsmärkte in NRW zu leisten. Es fokussiert auf gemeinsame Forschungs- und Entwicklungsprojekte von Wissenschaft und Industrie, aus denen kurz- und mittelfristig neue Produkte und Dienstleistungen entstehen können. Eine wesentliche Komponente von LoFIP ist daher der Wissenstransfer in die industrielle Praxis. Im Projekt arbeiten Forschungspartner eng mit Industriepartnern aus der Anwendungsdomäne zusammen.

Auf Basis früherer Erfahrungen der Projektteilnehmer wurde eine Vorgehensweise gewählt, die sich durch drei Kernaspekte auszeichnet:

- Szenariobasierte Erhebung von Anforderungen (siehe Abschnitt 2);
- Erlebbarmachen im Logistik-Living-Lab (siehe Abschnitt 3);
- Frühzeitige Validierung mit Domänenexperten (siehe Abschnitt 4).

## **2. Szenariobasierte Erhebung von Anforderungen**

LoFIP betrachtet zwei für die Domänenpartner relevante Anwendungsfälle: (1) den Hinterlandtransport von Containern in einem multimodalen Transportnetzwerk und (2) den Paketvorlauf in einem nationalen Versandnetzwerk. Erster Schritt war, konkrete Future-Internet-Leitstände für diese beiden Anwendungsfälle zu konzipieren und prototypisch umzusetzen (siehe auch Abschnitt 4), um in einem anschließenden Schritt die Erkenntnisse und Lösungen zu verallgemeinern.

Die Anforderungserhebung erfolgte szenariobasiert. Szenarien beschreiben konkrete Abläufe in der Nutzung eines Systems und beziehen sich jeweils auf bestimmte Ziele [Poh08]. Die Nutzung der Leitstände wurde in den LoFIP-Szenarien entlang der jeweils betrachteten Transportkette beschrieben, was im weiteren Verlauf die Integration von Technologien des Internets der Dinge vereinfachte. In LoFIP relevante Ziele für die Szenarien sind „Ressourceneffizienz“ und „Robustheit“. Unsere Erfahrung bestätigt frühere Erkenntnisse der SE-Community: Szenarien unterstützen maßgeblich die Anforderungserhebung und tragen darüber hinaus zu einem Austausch und gegenseitigem Lernen bei. Als geeignete Dokumentationsform haben sich textlich beschriebene instanziierte Szenarien herausgestellt. Sie unterstützten den Austausch mit allen Partnern und boten eine sehr gute Basis für die weitere Anforderungserhebung und -abstimmung. Typ-Szenarien (z.B. BPMN- oder Aktivitätsdiagramme) wurden von Domänenpartnern als komplex und unübersichtlich empfunden.

Kern der Anforderungserhebung war die Identifikation von Zukunftsszenarien, welche mögliche Situationen in 3–5 Jahren skizzieren. Im Rahmen von Kreativitätsworkshops

wurden hierzu gemeinsam Visionen erarbeitet, welche Ideen für einen möglichen Einsatz von Future-Internet-Technologien skizzierten. Im Nachgang erfolgte die Konkretisierung der Ideen durch einen intensiven Austausch zwischen Forschungs- und Industriepartnern. Wir wollten, anders als bei reiner Auftragsentwicklung, zeigen, welches Potenzial mittelfristig verfügbare Technologie haben kann.

Die Logistik gilt als relativ konservativ bezüglich Innovationen. Dies bedeutete, dass auf Seiten der Industriepartner vor allem Bedenken z.B. hinsichtlich unrealistischer Visionen oder dem Anstreben völliger Automation ausgeräumt werden mussten, während die Forschungspartner angehalten waren, entsprechend „praxisnäher“ zu denken. Wo möglich stellten die Zukunftsszenarien daher den Bezug zu initial erhobenen Ist-Szenarien her. Dieses Vorgehen erlaubte es auch, die Dokumentation der Zukunftsszenarien um geeignetes Präsentationsmaterial zu ergänzen. So konnten grafische Abbildungen der Abläufe durch Bildmaterial existierender, realer Abläufe und Ressourcen ergänzt werden, um die Szenarien den Domänenpartnern besser zugänglich machen.

### **3. Erlebbar machen im Logistik-Living-Lab**

Ein Living-Lab hat generell das Ziel, die Beteiligten enger in den Forschungs- und Entwicklungsprozess künftiger Produkte und Dienstleistungen einzubinden<sup>1</sup>. Bei dem LoFIP-Living-Lab für Logistik handelt es sich um eine quasi-realistische Umgebung, die es gestattet, Future-Internet-Leitstände gemeinsam mit Domänenpartnern zu erproben und neuartige technische Logistiklösungen zu evaluieren. Solche gemeinsamen Erprobungen fanden in regelmäßigen, mind. 3-monatigen, Zyklen statt. Zudem wurden externe Logistik- und IT-Experten in das Living Lab eingeladen. Aufgrund der quasi-realistischen Umgebung konnten diese schnell die Szenarien „begreifen“ und aus ihrer Sicht neue Anregungen zu den Szenarien und Demonstratoren liefern.

Zentrales Element des LoFIP-Living-Labs sind drei große Multitouch-Bildschirme (mit bis zu 85 Zoll), die gleichzeitig bis zu 32 Touchpunkte erkennen können. Diese Monitore bieten die Möglichkeit, unterschiedliche Arten der Interaktionen von Leitständen zu evaluieren. In unseren Szenarien hat sich herausgestellt, dass für die Interaktion mit Leitständen meist zwei Touchpunkte ausreichen. Wir beobachteten, dass die Nutzer so mit dem Leitstand interagierten wie sie es von einem Smartphone gewöhnt sind. Auch dort unterstützen viele Anwendungen bisher nur Gesten mit bis zu zwei Touchpunkten. Interessant war aber die Größe der Monitore, welche erlaubte, dass Personen gemeinsam vor dem Bildschirm stehen und mit den Leitständen interagieren konnten.

Der gewählte Einsatz von drei Bildschirmen bietet eine sehr hohe Variationsmöglichkeit bei der Präsentation von Inhalten. So können Leitstände, Szenarien aber auch reales Bildmaterial geeignet präsentiert und auch eindeutig zu einzelnen Bildschirmen zugeordnet werden. Abbildung 1 stellt eine typische Konfiguration der Bildschirme dar, die sich in LoFIP bewährte. Bildschirm 1 (links) zeigt externe Dinge und Informationsquellen aus dem Feld und erlaubt somit auch die Einbindung von Informationen externer Standorte. Bildschirm 2 (Mitte) stellt das Szenario und den aktuell demonstrierten Sze-

---

<sup>1</sup> <http://global.sap.com/corporate-de/innovation/living-labs/index.epx>

nario-Schritt dar, was die Orientierung und den Bezug zu der Domäne unterstützt. Bildschirm 3 (rechts) zeigt den eigentlichen Leitstands-Prototypen und ermöglicht die direkte Interaktion durch den Nutzer.

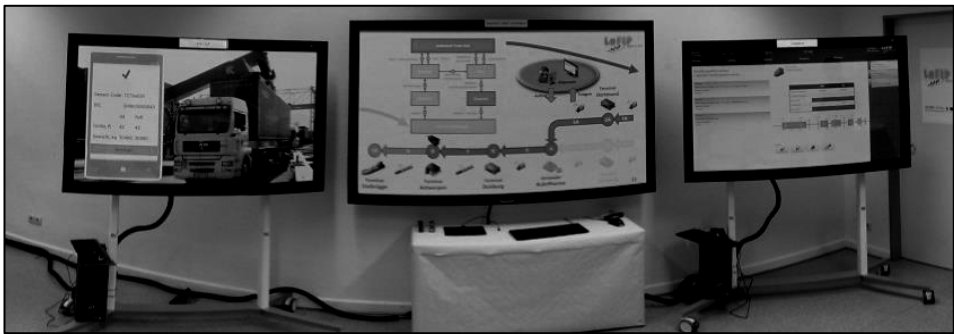


Abbildung 1: Bildschirme im Living-Lab zur Einbindung von Nutzern (Leitstand rechts)

In anderen Disziplinen haben sich in den vergangenen Jahrzehnten „Konstruktionsfallen“ herauskristallisiert, welche die Fähigkeit eines Disponenten bei der Bedienung von Leitständen deutlich einschränken können [EBJ03]. Zu diesen „Fallen“ gehören beispielsweise die kognitive Überlastung des Disponenten mit zu vielen Informationen („data overload“) und die Notwendigkeit, dass sich Disponenten Dinge im Gedächtnis merken müssen und diese daher leicht in Vergessenheit geraten können („requisite memory trap“). Abbildung 2a zeigt die sog. *Ereignisleiste* eines LoFIP-Leitstands, welche wichtige Ereignisse farblich klassifiziert (z.B. orange für eine „Warnung“) und unwichtige ausblendet (grau). Abbildung 2b demonstriert die Nutzung der sog. *Pinnwand*, auf welcher Disponenten wichtige zu bearbeitende Ereignisse per Touch-Geste ablegen können, wie z.B. die „Warnung“ aus (a).

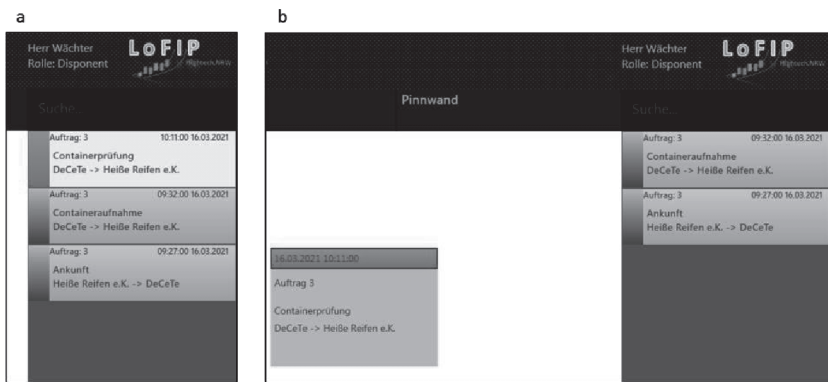


Abbildung 2: LoFIP-Leitstand mit (a) Ereignisleiste und (b) Pinnwand

Neben der Darstellung von Abbildungen realer Dinge auf den Living-Lab-Bildschirmen zeichnet sich das LoFIP-Living-Lab dadurch aus, dass echte physikalische Gegenstände und Objekte aus der Domäne eingebunden werden. Somit lässt sich das Erlebnis im

Living Lab durch die Nutzung physikalischer Objekte intensivieren. Neben reiner Demonstration, lassen sich auch technische Schwierigkeiten in einer quasi-realen Umgebung ausprobieren und zusammen mit den Domänenexperten analysieren.

Ein interessantes Beispiel für die Einbindung physikalischer Objekte ist die Erkennung von Paketen auf Rollcontainern. Die Paketmenge wird optisch durch eine Tiefenkamera erfasst (siehe Abbildung 3) und in Echtzeit an einen Leitstand gesendet. Dies erlaubt, die tatsächlich zu transportierende Paketmenge zu erfassen und diese Information ohne Verzögerung über Leitstände nutzbar zu machen. So können Abweichungen von zur Abholung *geplanten* Paketmengen erkannt und frühzeitig vom Disponenten adressiert werden.

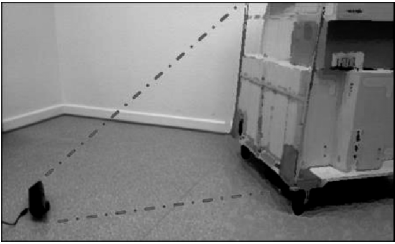


Abbildung 3: Internet-der-Dinge-Gerät zur Erfassung physikalischer Objekte (hier: Pakete)

Die physikalische Umsetzung und Einbindung solcher Geräte in das Living-Lab erlaubt Domänenexperten, diese neue Technologie live zu erleben und „anzufassen“. Als mögliche Schwierigkeiten für den Praxiseinsatz wurden so z.B. harsche Einsatzumgebungen für die Geräte oder mögliche Behinderungen der typischen Arbeitsprozesse identifiziert.

4. Frühzeitige Validierung mit Domänenexperten

Aufgrund des Innovationsgrads der LoFIP-Lösungen wurden frühzeitig Domänenexperten zur Validierung der Ergebnisse eingebunden. Die Erhebung und Validierung der Szenarien und Anforderungen erfolgte iterativ und in kurzen Abstimmungszyklen. Die Szenarien wurden dazu im Verlaufe der Entwicklung vom Ist- zum Zukunftsszenario zunächst durch Papierskizzen visualisiert (siehe Abbildung 4a), dann mittels Powerpoint-Mockups der Bedienkonzepte animiert (b) und schließlich in horizontalen Prototypen umgesetzt (c). Ein Animationsvideo zur Veranschaulichung des Container-Szenarios findet sich unter [www.lofip.de](http://www.lofip.de).

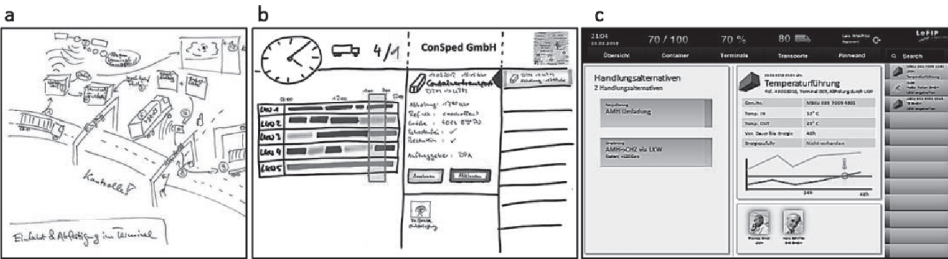


Abbildung 4: Visualisierung der Szenarien mittels (a) Skizzen, (b) Mockups und (c) Prototypen

Die dargestellte schrittweise Vorgehensweise ermöglichte es, Domänenpartner sehr früh und durchgängig einzubinden und den Nutzer in den Mittelpunkt zu stellen. Dies erlaubte den Schwerpunkt auf die für die Industriepartner relevanten Probleme zu legen. Gleichzeitig bot es aber auch den Forschungspartnern die Chance, Innovationen, die durch aktuelle Technologie ermöglicht werden, zu demonstrieren. Eine wichtige Erfahrung war, dass man insbesondere zu Beginn deutlich machen sollte, dass es sich nicht um das endgültige Erscheinungsbild der Leitstände handelt, weil sich sonst das Feedback im Wesentlichen auf Layout und Farbgebung bezieht. Hierzu waren in der Tat handgezeichnete Grafiken (siehe (a) und (b) in obiger Abbildung) sehr hilfreich, da diese nur „abstrakte“ Interaktionskonzepte darstellen.

## 5. Fazit und Ausblick

Die frühe und regelmäßige Einbindung von Domänenexperten war ein wesentlicher Erfolgsfaktor bei dem Transfer von Know-How. Den Industriepartnern konnten hierdurch die Potenziale der neuartigen Lösungen aufgezeigt werden, was wiederum weitergehende Ideen stimulierte, wie z.B. die Föderation von Leitständen über Organisationsgrenzen hinweg. Die häufigen Abstimmungszyklen erforderten einen erheblich Aufwand in Vor- und Nachbereitung, waren aber essentiell aufgrund der Interdisziplinarität des Projekts (IT und Logistik).

**Danksagung:** Unser großer Dank gilt allen Kolleginnen und Kollegen, sowie den externen Experten des LoFIP-Projekts. Wir bedanken uns auch für die sehr hilfreichen Gutachterkommentare. Die Forschungsarbeiten wurden gefördert im Rahmen des aus dem EFRE ko-finanzierten Operationellen Programms für NRW im Ziel 2 „Regionale Wettbewerbsfähigkeit und Beschäftigung“ 2007-2013 ausgewählten Projekts „LoFIP“ (Förderkennzeichen: 005-1010-0012).

## Literaturverzeichnis

- [Bil07] R. Bildmayer, „Logistischer Leitstand“, INFORMATIK 2007 – Informatik trifft Logistik, Band 2, Beiträge der 37. Jahrestagung der GI, Bremen, 2007
- [EBJ03] M. R. Endsley, B. Bolté, and D. G. Jones, Designing for Situation Awareness: An Approach to User-Centered Design. New York: Taylor and Francis, 2003.
- [MF10] F. Mattern, C. Flörkemeier: Vom Internet der Computer zum Internet der Dinge, Informatik-Spektrum, Sonderheft „Future Internet“. 33(2), April 2010
- [MRM13] S. Meyer, A. Ruppen, C. Magerkurth, „Internet of Things-Aware Process Modeling: Integrating IoT Devices as Business Process Resources“, 2th Int’l Conference on Advanced Information Systems Engineering (CAiSE), Valencia, 2013
- [Nes04] D. Nesamoney, „BAM: Event-driven business intelligence for the real-time enterprise,“ DM REVIEW, 14, 2004.
- [NGM08] E. Di Nitto, C. Ghezzi, A. Metzger, M. P. Papazoglou, and K. Pohl, „A journey to highly dynamic, self-adaptive service-based applications,“ Autom. Softw. Eng., 15(3-4), 2008.
- [Poh08] K. Pohl, Requirements Engineering - Grundlagen, Prinzipien, Techniken. 2. korrigierte Auflage, dpunkt.Verlag, Heidelberg, 2008

# Ein gutes Bild erfordert mindestens 1000 Worte – Datenvisualisierungen in der Praxis

Steffen Kruse, Philipp Gringel

Architekturentwicklung und Interoperabilität  
OFFIS - Institut für Informatik  
Escherweg 2  
26121 Oldenburg  
{steffen.kruse, philipp.gringel}@offis.de

**Abstract:** In diesem Beitrag werden Erfahrungen aus einem langjährigen Industrieforschungsprojekt vorgestellt, in dem ein Generator für Visualisierungen strukturierter Daten aus der Domäne Enterprise Architecture Management entwickelt wurde. Obwohl die Generierung von Visualisierung erhebliche Vorteile hat, waren im Entwicklungsprozess auf dem Weg zur Praxistauglichkeit unerwartete Hürden zu nehmen. Diese werden am Beispiel eines konkreten Projektes erläutert. Wir gehen davon aus, dass unsere Erfahrungen wertvoll für künftige Forschungsprojekte mit Industriepartnern sind.

## 1 Einleitung

Viele Unternehmen haben sich in den letzten Jahren der Aufgabe des Unternehmensarchitektur-Managements (Enterprise Architecture Management, EAM, [Lan13]) gestellt, um besser mit der wachsenden Komplexität der Unternehmensarchitektur umgehen zu können und um das Geschäft besser mit der IT zu verzahnen ([HV93]). Seitdem sind viele Werkzeuge am Markt platziert, die eine unterstützende Rolle im EAM einnehmen können ([MBLS08]). Der Zweck dieser Werkzeuge ist zum einen die Dokumentation der unterschiedlichen Entitäten der Unternehmensarchitektur, die Verbindung von Elementen unterschiedlicher Ebenen ([WF07]) sowie die Analyse der Daten. Unterschiedliche Stakeholder der Unternehmensarchitektur können sich Sichten bedienen, die auf ihre Rolle zugeschnittene Analysen und unterstützende Visualisierungen (sog. Softwarekarten, [Wit07], [KAPS09]) bieten.

Dieser Erfahrungsbericht beruht im Wesentlichen auf dem in Abschnitt 2 beschriebenen Projektkontext, der die vergangenen Jahre abdeckt, sowie auf einem aktuellen Projekt, auf welches in Abschnitt 3 eingegangen wird. In Abschnitt 4 werden die wichtigsten gesammelten Erfahrungen, und ggfs. ein Interpretationsversuch der Ursachen, zusammengestellt. Der Beitrag endet in Abschnitt 5 mit allgemeinen Empfehlungen, die wir aus den gemachten Erfahrungen ableiten.

## 2 Projektkontext

Die Autoren sind auf Seiten eines Forschungsinstituts seit 2009 in einer seit langem bestehenden Kooperation mit einem Industriepartner tätig, in deren Rahmen unter Anderem auch EAM-Projekte bearbeitet wurden und werden. Dieser Partner hat bereits vor 2005 begonnen, Daten über seine IT-Infrastruktur (Hard-/Softwaresysteme) systematisch in einem eigenentwickelten Werkzeug zu dokumentieren. Dieses wird von unterschiedlichen Stakeholdern genutzt, um beispielsweise Informationen über den Applikationslebenszyklus einer bestimmten Anwendung zu erhalten („Welche Abteilung nutzt welches Softwaresystem, und in welcher Lebenszyklusphase (geplant, im Aufbau, Nutzung, Ablösung) befindet es sich?“), oder um festzustellen, in welchem Serverschrank ein bestimmtes Serverrack eingebaut ist. Einige Analysen, z.B. die Fragen „Welche Anwendung tauscht mit welcher anderen Anwendung über welche Schnittstelle / welches Protokoll welche fachlichen Informationsobjekte aus?“ oder „Aus welchen Modulen setzt sich eine Applikation zusammen und auf welchen (virtuellen) Servern sind diese gehostet?“, profitieren von grafischen Ergebnisdarstellungen.

Im Rahmen der Forschungsk Kooperation wurden in einem früheren Projekt (2008/2009) aus der Literatur (z.B. Softwarekarten, [Wit07], [MBLS08]) bekannte Visualisierungen auf Einsetzbarkeit im konkreten Projektkontext hin bewertet und, wo nötig, eigene Darstellungen entwickelt. Projektziel war es eine Software zu entwickeln, die auf Basis der Daten des Industriepartners, in bestimmten Eigenschaften flexible, Visualisierungen generieren konnte. Beim Entwurf und bei der Implementierung wurden die konkreten Anwendungsfälle berücksichtigt, die für die Nutzer der Eigenentwicklung am wichtigsten waren und die sich sinnvoll grafisch aufbereiten ließen. Beispielsvisualisierungen finden sich online<sup>1</sup>.

Aus den Anforderungen abgeleitet wurde ein Datenmodell, das in die erste Version der Software eingeflossen ist. Diese wurde als Komponente mit definierten Schnittstellen in die Eigenentwicklung des Industriepartners eingebunden. Zur Darstellung der Softwarekarten in der Webanwendung des Partners wurde das SVG-Format gewählt. Zur manuellen Anpassung der Darstellungen zur späteren Verwendung, z.B. in Präsentationen, konnten auch Microsoft-Visio-Dateien erzeugt werden.

Die anfängliche Fokussierung auf IT-Infrastrukturdaten hatte zur Folge, dass Instanzdaten aus anderen Bereichen der Unternehmensarchitektur, z.B. zu Prozessschritten und Geschäftsdaten, nur wenig verfügbar waren, so dass nicht alle der im Werkzeug realisierten Analysen und entsprechend auch nicht die damit gekoppelten Visualisierungen genutzt werden konnten.

Ab 2010/2011 trugen verschiedene andere Projekte zu verbesserter Datenmenge und Datenqualität bei. Endnutzer konnten dadurch erstmalig über bisher ungenutzte Visualisierungen der Realdaten verfügen. Die Adoption des SOA-Paradigmas, in Diensten und nicht mehr in „traditionellen“ monolithischen Applikationen zu denken, ist ein Beispiel für geänderte Anforderungen. Hinzu kamen zahlreiche neue Anforderungen, die in Summe den Ausschlag dafür gaben, Anfang 2013 gemeinsam ein Projekt zur Reimplementierung

---

<sup>1</sup>[http://srvbi05.offis.uni-oldenburg.de/EA\\_Demonstrator/public/index/help](http://srvbi05.offis.uni-oldenburg.de/EA_Demonstrator/public/index/help)

der bestehenden Software aufzusetzen. Das Projekt sollte mit agilen Methoden ([OW08]) umgesetzt werden.

Der erste Prototyp (vgl. Abschnitt 3, Abbildung 1) wurde positiv aufgenommen und die Projektarbeit verläuft erfolgreich. Die alte Software soll zum Jahreswechsel 2013/2014 von der Neuentwicklung nach und nach abgelöst werden.

### 3 Visualisierungen

Zu Beginn der Reimplementierung wurden die neuen Anforderungen an die „alten“ Visualisierungen aufgenommen. Insbesondere in Hinblick auf die Ausgabeformate SVG und Microsoft Visio sollte funktionale Abwärtskompatibilität gewährleistet werden. Vom Industriepartner bereits tabellarisch erhobene Daten und einige in Microsoft Visio gezeichnete Darstellungen der Daten bildeten die Grundlage für neue Arten von Visualisierungen.

#### Organisation, Prozesse, Anwendungsservices und IT-Systeme

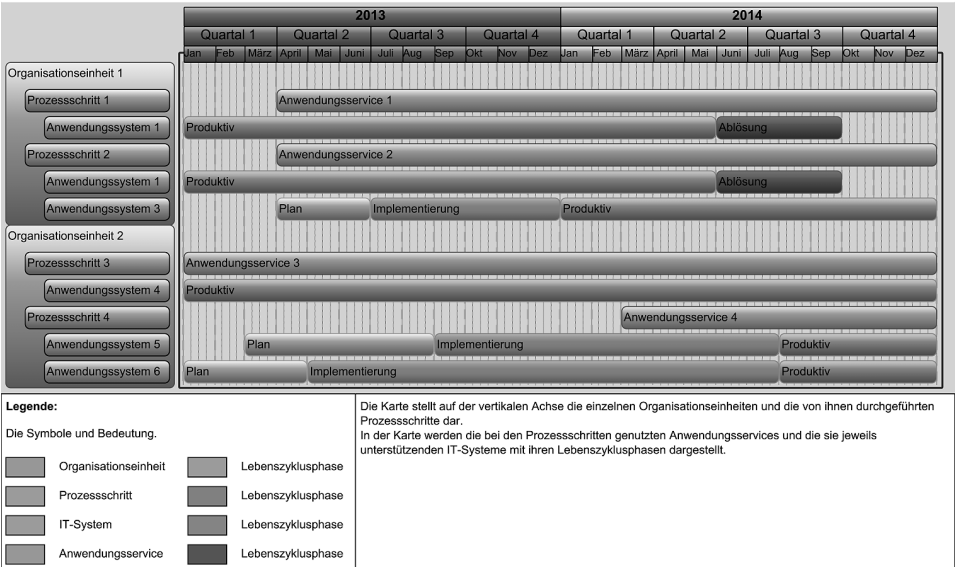


Abbildung 1: Die Visualisierung stellt die Anwendungsservices, die bei der Durchführung von Prozessschritten in bestimmten Organisationseinheiten genutzt werden, den Anwendungssystemen gegenüber, die diese Services zur Verfügung stellen.

Die entwickelte Software beherrscht zehn verschiedene Visualisierungstypen, die strukturierte Daten darstellen können. Dabei kann vom Nutzer festgelegt werden, welche Domänenobjekte dargestellt werden sollen. Die Visualisierungen können damit auch außerhalb der EAM-Domäne und dem in Abschnitt 2 genannten Projektkontext eingesetzt werden. Sofern strukturierte Daten mehrfach auf ähnliche Weise dargestellt werden müssen, wie



dies bei Reports die Regel ist, können die Visualisierungen leicht an die Domäne angepasst werden. Abbildung 1 zeigt eine „Intervallkarte“, die im Beispiel die Lebenszyklusphasen von Anwendungssystemen darstellt.

Allgemein stellt die Karte auf der Y-Achse Bäume dar, deren Knoten mit den zeitlichen Intervallen der Karte in Beziehung stehen. Die X-Achse stellt eine Zeitachse dar, deren Auflösung zwischen Sekunden, Minuten, Stunden, Tagen, Monaten, Quartalen und Jahren frei gewählt werden kann. Unterschiedliche Auflösungen können, wie in Abbildung 1 dargestellt, in derselben Visualisierung genutzt werden.

## 4 Lessons learned

Nach unseren Erfahrungen aus dem oben beschriebenen und anderen Kooperationsprojekten gestalten sich diese nicht grundsätzlich anders als bei anderen Projektarten. Es gibt jedoch einige Aspekte, die nach unserer (sicherlich subjektiven) Erfahrung bei der Zusammenarbeit von Wissenschaftlern und Kooperationspartnern aus der freien Wirtschaft eine besondere Rolle spielen. Hier mögen in anderen Projekten andere Erfahrungen gemacht worden sein und für langjährige Mitarbeiter keine wesentlichen neuen Erkenntnisse enthalten sein – vielleicht hilft es aber neuen Kollegen, sich zu Beginn eines Kooperationsprojektes die folgenden Aspekte vor Augen zu führen:

**Ungleiche Partner** In Kooperationen zwischen Wissenschaft und Wirtschaft haben die Partner unterschiedliche Kulturen und Arbeitsweisen und verfolgen unterschiedliche Ziele bzw. bewerten die Güte von Ergebnissen unterschiedlich. Es ist unserer Erfahrung nach wichtig, diese Unterschiede bei Projektbeginn festzustellen, zu harmonisieren und immer wieder allen Stakeholdern zu kommunizieren. Für Wissenschaftler gilt in erster Linie die positive Bewertung einer fachkundigen Öffentlichkeit als Erfolgsmaß, wobei die objektive Nachvollziehbarkeit von Ergebnissen eine Schlüsselrolle spielt. Projektbeteiligte aus der freien Wirtschaft bewerten Projekte hingegen nach Aspekten wie Kosten, Nutzen oder Risiken. In der Wirtschaft sind die Berichtszeiträume kürzer und verwertbare Teilerfolge während des Projektverlaufs wichtig für eine fortlaufende Bewertung. Für Unternehmen steht der praktische Einsatz von Lösungen im Vordergrund, während Wissenschaftler bemüht sind, Lösungen zu verallgemeinern und die Gültigkeit für eine ganze Problemklasse zu zeigen. Es ist aus unserer Sicht sehr hilfreich, als Forschungseinrichtung frühzeitig die Möglichkeiten der Veröffentlichung von Projektergebnissen anzusprechen und um Verständnis zu werben, dass die Veröffentlichung einen erheblichen (wenn nicht sogar den einzigen) Mehrwert für wissenschaftliches Arbeiten bedeutet.

**Hürden des Alltags** Kooperationsprojekte starten in der Regel nicht auf der grünen Wiese und alle Kooperationspartner bringen ihre jeweilige organisatorische und technische Vorgeschichte mit in ein gemeinsames Projekt. Während Wissenschaftler eine Vorliebe für Open-Source-Produkte hegen, gelten auf Seiten der Industrie aus Sicherheits- und Lizenzgründen häufig umfassende Regeln zum Softwareeinsatz im Alltag, die deutlich re-

striktiver sind als in der Wissenschaft. Diese werden ausschlaggebend, wenn Ergebnisse beim Projektpartner im Regelbetrieb eingesetzt werden sollen. Dabei spielt die frühzeitige Planung der Evaluation eine wichtige Rolle, wenn hier reale Daten, Systeme und Prozesse herangezogen werden sollen. Auch ist zu bedenken, dass in großen Unternehmen die Entscheidungen zum Einsatz im Unternehmensalltag oft an anderer Stellen getroffen werden als von den Projektbeteiligten.

In unserem konkreten Projekt mussten wir z.B. ältere Versionen von Java und einem Web-Browser berücksichtigen, was weitreichende technische Konsequenzen hatte. Ansonsten wäre jeder Einsatz im Unternehmen von vornherein unmöglich gewesen. Des Weiteren konnten wir die häufig von EAM-Werkzeugh Herstellern getätigte, recht plakative Aussage bestätigen, Microsoft Excel sei das in Unternehmen am weitesten verbreitete EAM-Werkzeug (vgl. [MBLS08]). Dies lässt sich auf die weite Verbreitung von Excel zurückführen und auf den Umstand, dass EAM-Tätigkeiten auch dann anfallen, wenn kein EAM-Einführungsprojekt im Unternehmen durchgeführt wurde und keine explizite Werkzeugwahl stattgefunden hat. Da so eine beträchtliche Menge an relevanten Daten in der Form von Excel-Tabellen vorlag, und (wahrscheinlich noch wichtiger) Prozesse und Arbeitsweisen für die kontinuierliche Aktualisierung dieser Daten existierten, bestand eine weitere essentielle Prämisse in der Unterstützung von Excel (und anderen MS-Produkten).

**Langlebige Prototypen** Während der langen Laufzeit unserer Forschungsk Kooperation haben wir die Erfahrung gemacht, dass Software-Prototypen oft länger leben als anfänglich gedacht. Es ist ein Erfolg, wenn Prototypen direkt in den produktiven Einsatz im Unternehmensalltag eingehen, jedoch bringt dieser Erfolg weitere Herausforderungen mit sich. Zum einen sind Prototypen architektonisch meist nicht auf langfristige Weiterentwicklung und Wartung ausgelegt, was die normalen Software-Alterungsprozesse deutlich beschleunigt [Leh80]. Zum anderen kann eine Forschungseinrichtung kaum Softwareentwicklung auf professionellem Niveau betreiben und auch nicht entsprechende Wartungsleistungen anbieten. Dies ist auch nicht im Interesse einer Forschungsgruppe, da die langfristige Betreuung von Projektergebnissen nach Projektende kaum Möglichkeiten zur Gewinnung neuer Erkenntnisse bietet. Hier empfiehlt sich die frühzeitige Planung zur Verwertung von Projektergebnissen und die Schaffung grundsätzlicher Strukturen für weiterführende Verwertung.

## 5 Empfehlungen

Für uns ist die wichtigste Empfehlung für Kooperationen die gleiche wie für jede Art von Projekt, die frühzeitige und umfassende Abstimmung mit allen Stakeholdern über Erwartungen, Ziele und Inhalte ist unerlässlich für einen erfolgreichen Projektverlauf. Darüber hinaus ist es schwierig, einen konkreten Katalog an Empfehlungen für erfolgreiche Transferprojekte aufzustellen, da diese strukturell und inhaltlich sehr unterschiedlich sind. Jedoch haben wir sehr gute Erfahrungen mit agilen Projektmanagementmethoden gemacht, da diese die Kommunikation aller Projektpartner in den Vordergrund stellen und sehr fle-

xiblen Umgang mit veränderlichen Bedingungen und Zielen erlauben (siehe z.B. [OW08]). Der Fokus agiler Methoden auf die frühzeitige und kontinuierliche Ausarbeitung konkreter Ergebnisse (wobei es unerheblich ist, ob es sich dabei um Software-Prototypen, Berichte oder Studien handelt) erleichtert die Abstimmung der Erwartungen der unterschiedlichen Projektpartner und verhindert einen Totalausfall bei unerwarteten Problemen.

## Literatur

- [HV93] John C. Henderson und N. Venkatraman. Strategic Alignment: Leveraging Information Technology for Transforming Organizations. *IBM Systems Journal*, 32(1):472–484, Januar 1993.
- [KAPS09] Steffen Kruse, Jan Stefan Addicks, Matthias Postina und Ulrike Steffens. Decoupling Models and Visualisations for Practical EA Tooling. In Asit Dan, Frederic Gittler und Farouk Toumani, Hrsg., *Proceedings of the 2009 International Conference on Service-Oriented Computing, ICSOC/ServiceWave '09 Workshops*, Seiten 62–71, Berlin, Heidelberg, 2009. Springer-Verlag.
- [Lan13] Marc M. Lankhorst. *Enterprise Architecture at Work - Modelling, Communication and Analysis*. The Enterprise Engineering Series. Springer, 3. Auflage, 2013.
- [Leh80] MM Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9), 1980.
- [MBLS08] Florian Matthes, Sabine Buckl, Jana Leitel und Christian M. Schweda. *Enterprise Architecture Management Tool Survey 2008*. Chair for Informatics 19 (sebis), Technische Universität München, 2008.
- [OW08] Bernd Oestereich und Christian Weiss. *APM - Agiles Projektmanagement*. dpunkt.Verl., Heidelberg, 1. aufl.. Auflage, 2008.
- [WF07] Robert Winter und Ronny Fischer. Essential Layers, Artifacts, and Dependencies of Enterprise Architecture. *Journal of Enterprise Architecture*, 3(2):7–18, Mai 2007.
- [Wit07] André Wittenburg. *Softwarekartographie: Modelle und Methoden zur systematischen Visualisierung von Anwendungslandschaften*. Dissertation, Technische Universität München, Institut für Informatik, München, 2007.

SE | 14  
SOFTWARE ENGINEERING

**Software & Systems  
Engineering Essentials**



# Releasemanagement in einem sehr komplexen Projekt

Katrin Heymann

IBM Deutschland  
Global Business Services  
IBM-Allee 1  
71139 Ehningen  
katrin.heyman@de.ibm.com

**Abstract:** Erfahrungsbericht über den Aufbau und die Umsetzung eines Releasemanagement Prozesses in einem schnell gewachsenen, sehr komplexen Projekt mit überlappenden Releases für verschiedene Kunden. Dabei werden die Herausforderungen und Lösungen dargestellt, um trotzdem den Überblick über den aktuellen Stand zu behalten. Der Ansatz basiert auf bewährten agilen Techniken und zeigt nötige Anpassungen an die Projektrealität. Dabei wird auch das hierfür eingesetzte Tool RTC (Rational Team Concert) vorgestellt.

## 1 Ausgangssituation

Basis für dieses Projekt ist ein Produkt das ursprünglich für einen Kunden von einem kleinen Team entwickelt wurde. Hierfür eignet sich ein an Scrum angelegter Prozess mit sequentiellen Releases. Das Produkt wird inzwischen von mehreren Kunden eingesetzt. Diese fordern individuelle Releases (Weiterentwicklung und Wartung). Aufgrund eines hohen Zeitdrucks erfolgt die Entwicklung der Releases überlappend. Das Team ist inzwischen auf über 100 Personen angewachsen.

Das kleine agile Team benötigte keinen definierten Releasemanagement Prozess. Bei den inzwischen zum Teil 2-5 Auslieferungen pro Woche ging der Überblick über den aktuellen Status, Inhalt der Releases und aktuelle Teamzuordnungen verloren. Ein Reporting gegenüber dem Kunden war nur mit großem Aufwand möglich.

## 2 Definition eines Releasemanagement Prozesses

Nach der Analyse des aktuellen Standes und der vorhandenen Informationen wurde ein Releasemanagement Prozess definiert, der alle Schritte vom Requirement bis zur Auslieferung beinhaltet und über einen definierten Workflow sicherstellt.

Alle Requirements werden spezifiziert und mit geplanten Aufwänden hinterlegt. Anhand dieser Kriterien kann eine realistische Zeitplanung und damit Zuordnung zu Releases und Teams erfolgen. Damit ist auch ein Überblick auf die Auswirkungen bei inhaltlichen bzw. Termin-Verschiebungen sichergestellt.

### **3 Umsetzung des definierten Prozesses**

Nur durch paralleles Arbeiten an mehreren Releases für verschiedene Kunden konnte die zeitliche Vorgabe erfüllt werden. Die größte Herausforderung war, trotz definiertem Releasemanagement Prozess den Überblick zu behalten. Hierbei unterstützt das Tool RTC (Rational Team Concert). Durch Verwendung eines Datenbestandes sind alle Informationen stets aktuell und Veränderungen werden in der Historie festgehalten.

Dank intensiver Kommunikation mit allen Beteiligten, konsequentes Workmanagement und Statusabgleich im RTC konnte sichergestellt werden, dass zur jeweiligen Auslieferung die Inhalte, der erfolgreiche Test und die Release Notes zur Verfügung stehen. Zusätzlich ist ein Reporting über den aktuellen Status gegenüber dem Kunden mit dem Tool RTC jederzeit möglich.

# Open Source als Triebfeder für erfolgreiche Softwareprojekte in der öffentlichen Verwaltung

Christian Werner, Ulrike Schneider  
Zentralabteilung  
Bundesamt für Strahlenschutz  
Willy-Brandt-Str. 5  
D - 38226 Salzgitter  
epost@bfs.de

Öffentliche Einrichtungen erbringen Dienstleistungen für Bürgerinnen und Bürger. Die dafür benötigte Prozesslandschaft ist in Deutschland dezentral organisiert (Föderalismus). Die Praxis zeigt, dass bei IT-gestützten Prozessen eine dezentrale Organisation sich mitunter nachteilig auswirkt. Der Open-Source-Gedanke verbindet die Idee einer Dienstleistung für die (Nutzer-)Gemeinschaft mit einem dezentral ablaufendem, evolutionären Entwicklungsprozess. In diesem Beitrag beleuchten die Autoren Besonderheiten bei der Entwicklung behördlicher Spezialsoftware und arbeiten die Vorteile eines Open-Source-Ansatzes heraus.

Die Organisation in Bund und Ländern folgt dem Ressortprinzip, d. h. jedes Ministerium gestaltet den eigenen Geschäftsbereich inklusive der erforderlichen IT-Verfahren selbst. Dies führt dazu, dass innerhalb der einzelnen Bereiche zwar eine stark ausdifferenzierte IT-Landschaft entstehen kann, die Schnittstellen dazwischen jedoch extrem komplex werden können. Um diese Heterogenität zu mindern, gibt es im Bund seit 2007 den *Rat der IT-Beauftragten der Ressorts (IT-Rat)* sowie die *IT-Steuerungsgruppe des Bundes*. Weiterhin gibt es seit 2009 den *IT-Planungsrat* zur IT-Koordinierung zwischen Bund und Ländern. Obwohl diese Gremien dazu beitragen, die Abstimmung zwischen den einzelnen Akteuren zu verbessern, sind durch Föderalismus und Ressortprinzip Parallelentwicklungen sowie Schnittstellenprobleme nicht ausgeschlossen.

Bisher wurde Open-Source-Software im öffentlichen Bereich vor allem vor dem Hintergrund entfallender Lizenzkosten bei Standardsoftware betrachtet [WM05] – in Deutschland z. B. im Rahmen des bekannten LiMux-Projekts der Landeshauptstadt München.

Wird der Quelltext einer Software veröffentlicht, so hat dies jedoch weitere positive Effekte: 1) Behörden können sich einen Überblick verschaffen, welche Softwarelösungen im Rahmen öffentlicher Aufträge bereits erstellt wurden. Der Aufwand für Mehrfachentwicklungen in verschiedenen Behörden kann so vermieden bzw. reduziert werden. 2) Fehler und Design-Schwächen werden ebenfalls öffentlich. Auf diese Weise werden die Entwickler (in aller Regel externe Dienstleister) gezwungen, nicht nur eine vertragsgemäße Leistung abzuliefern, sondern haben (mit Blick auf etwaige Folgeaufträge bei anderen Kunden) ein Eigeninteresse daran, eine besonders gute Software abzuliefern. Dies ist ein wichtiger Ausgleich zu mitunter mangelnder IT-Expertise beim Auftraggeber. 3) Mit dem Quell-



text werden auch alle technischen Schnittstellen offengelegt, so dass der Informationsfluss zwischen Behörden untereinander sowie zwischen Behörden und Bürgern optimiert wird (Förderung von OpenData). 4) Ein Vendor-Lock-In wird vermieden, da andere Dienstleister die bereits erstellte, quelloffene Software weiterentwickeln können. 5) Das Ressortprinzip wird in der Weise unterstützt, dass eine vorgesetzte Behörde eine Software für ihren nachgeordneten Bereich verbindlich vorgeben kann, die nachgeordneten Behörden jedoch Erweiterungen für fachspezifische Aufgaben ergänzen können. Auch zwischen Bund und Ländern können Synergien bei IT-Projekten mit einer gemeinsamen, offenen Code-Basis entstehen. 6) Für den Bürger, die Bürgerin ergibt sich eine hohe Transparenz (vgl. Informationsfreiheitsgesetz): Jeder kann sich darüber informieren, welche Behörde welche Softwareprojekte mit welchen Ergebnissen abschließt.

Das Konzept *Open Source* sollte daher nicht auf Kostenaspekte reduziert werden. Vielmehr bietet es das Potenzial, die Prozessabläufe in öffentlichen Einrichtungen der Öffentlichkeit gegenüber transparent zu machen. Interessant ist letztlich die Frage, ob der Bürger nicht das Recht bekommen sollte, jede Software, die mit Steuermitteln finanziert ist, im Sinne der Open Source Initiative<sup>1</sup> nutzen zu können.

Ein zweiter Aspekt ist die Möglichkeit der Verzahnung von Prozessabläufen in verschiedenen Behörden auf Bund- und Landesebene durch Nutzung einer gemeinsamen, offenen Codebasis. Auf diese Weise kann behördliche Softwareentwicklung in einem dezentralen Prozess evolutionär ablaufen.

Auch der rechtliche Rahmen für Open Source ist mittlerweile umfassend aufgearbeitet [JM11].

Das Bundesamt für Strahlenschutz entwickelt beispielsweise zurzeit einen Nachfolger für das *Integrierte Mess- und Informationssystem (IMIS)*<sup>2</sup> auf Basis von Open Source. Ausschlaggebend hierfür war unter anderem eine vereinfachte Vertragsgestaltung bei der Anpassung des Systems an die Bedürfnisse ausländischer Projektpartner.

Als Nachteile der Offenlegung des Quelltextes behördlicher IT-Systeme sind Angriffe denkbar, die auf dadurch erst bekannt gewordenen Sicherheitslücken basieren. „Security through obscurity“ ist in der Fachwelt jedoch umstritten.

Offen bleibt die Frage, durch welche organisatorischen Maßnahmen die Synergieeffekte zwischen Behörden bei Open-Source-Projekten noch verstärkt werden können.

## Literatur

- [JM11] Till Jaeger und Axel Metzger. *Open Source Software: Rechtliche Rahmenbedingungen der Freien Software*. C. H. Beck, 3. Auflage, 2011.
- [WM05] Teresa Waring und Philip Maddocks. Open Source Software implementation in the UK public sector: Evidence from the field and implications for the future. *Int. Journal of Information Management*, 25(5):411–428, Oktober 2005.

---

<sup>1</sup><http://www.opensource.org/>

<sup>2</sup>[http://www.bfs.de/de/ion/imis/imis\\_uebersicht.html](http://www.bfs.de/de/ion/imis/imis_uebersicht.html)

# Effiziente Erstellung von Software-Factories

Ralf Leonhard, Gerhard Pews, Simon Spielmann

Bundesverwaltungsamt - Barbarastraße 1, 50735 Köln

Capgemini - Berliner Str. 76, 63065 Offenbach

Ralf.Leonhard@bva.bund.de

{Gerhard.Pews|Simon.Spielmann}@capgemini.com

**Abstract:** Im BVA existiert seit 2007 mit der Register Factory ein Ansatz, um große Registeranwendungen effizient und zuverlässig umzusetzen und zu warten. Die Erfahrungen sind so gut, dass er auf andere Anwendungen übertragen werden soll. Die entstehenden Factories sollen kostengünstig erstellt und weiterentwickelt werden und voneinander gegenseitig profitieren. Dieser Artikel stellt vor, wie im BVA eine Factory-Familie für den öffentlichen Bereich entwickelt wird.

## 1 Die Register Factory

Die Register Factory ist ein Komplettpaket zur Erstellung großer Registeranwendungen. Grundlage sind Standards für Architekturen und Software Engineering-Vorgehen, auf denen Werkzeuge und Lösungsbausteine bereitgestellt werden, sowie eine Betriebsplattform, auf der die Anwendungen lauffähig sind. Seit 2007 wurden so zahlreiche große, komplexe Systeme erstellt (Visa-Verfahren, Nationales Waffenregister, Ausländerzentralregister). Alle liefen im geplanten Budget, zum vereinbarten Zeitpunkt und von Beginn an stabil. Die Wartung speist sich kostengünstig aus einem Pool von Wartungskräften und erlaubt eine Steuerung von Aufwänden. Für den Erfolg der Register Factory sind Standardisierung und praxisorientierte und leicht verständliche Vermittlung entscheidend. Aufgrund sehr guter Erfahrungen wird der Factory-Ansatz auf weitere Anwendungsgebiete übertragen.

## 2 Factories für andere Anwendungsdomänen

Die neu zu erschließende Anwendungsdomäne ist die kleinerer Fachverfahren. Diese wird kurz charakterisiert und den Registeranwendungen gegenübergestellt. Danach wird vorgestellt, wie Register Factory und Fachverfahrens Factory strukturiert werden, um Synergien zu nutzen, um einen schnellen Aufbau der Fachverfahrens Factory zu leisten.

Registeranwendungen verwalten große Datenbestände und definieren Arbeitsprozesse, die auf diesen Datenbeständen operieren. Sie sind Bestandteil einer SOA-Landschaft und haben typischerweise einen großen Nutzerkreis. Im Gegensatz dazu haben kleinere Fachverfahren einen Nutzerkreis von wenigen Personen und stehen für sich allein. Sie haben selten Schnittstellen zu anderen Systemen und kaum verfahrensübergreifende

Prozesse. Sie können aber sehr umfangreich sein. Die Wahl fiel auf diese Domäne, weil sie im Behördenumfeld weit verbreitet ist.

Die Register Factory wurde dazu neu strukturiert und eine Basis bestehend aus Standards und Bausteinen definiert, die als PLIS (Plattform für Informations-Systeme) bezeichnet wird. Die PLIS-Standards sind Referenzarchitekturen für die Software-Technik einzelner Anwendungen und die Kommunikation zwischen Anwendungen, Infrastruktur und Netzen. Weiterhin enthalten die PLIS-Standards Bibliotheken, die Vorgaben umsetzen und direkt zum Bau von Anwendungen nutzbar sind und einen Produktkatalog, der hauptsächlich auf Open Source Produkten beruht. Anwendungen, die mit Hilfe der PLIS-Standards entwickelt werden, können einheitlich durch den Systembetrieb betreut werden und sind so aufgebaut, dass einfach höherwertige Software-Bausteine zu erstellen sind, die mit allen Anwendungen kompatibel sind. Die Bausteine bilden die PLIS-Factory, die auf den PLIS-Standards aufsetzt. In ihr befinden sich Software-Bausteinen in Form von Bibliotheken oder querschnittlichen Services und Methodenbausteine und Werkzeuge.

PLIS-Standards und PLIS-Factory bilden die Grundlage für unterschiedliche, domänenspezifische Factories, die spezielle Hilfsmittel für die jeweilige Domäne enthalten. Sie machen Vorgaben, welche Lösungsbausteine der darunter liegenden Schicht genutzt werden sollen und in welcher Art die Nutzung erfolgen soll. In der Register Factory verbleiben somit z. B. fachliche Referenzarchitekturen, die den Aufbau von Registern beschreiben und die typische Abläufe und Referenzdatenmodelle enthalten, Register-spezifische Lösungsbausteine wie alphanumerische oder biometrische Suchverfahren beim BVA oder die spezielle Betriebsplattform für Register. Analog ist die Fachverfahrens Factory aufgebaut.

### **3 Ausblick**

Die Tragfähigkeit des Ansatzes wurde mit der Erstellung eines Fachverfahrens validiert. Aktuell läuft die Konsolidierung und weitere Fachverfahren werden realisiert. Die Aufteilung in Standards, allgemeine Factory und spezielle Factories reduziert deutlich den Aufwand für Erstellung und Pflege neuer Factories. Für Standards soll eine Community entstehen, die Aufwände für Pflege und Risiko bei Innovationen auf viele Schultern verteilt. Für die allgemeinen Factory soll ein Marktplatz für Lösungsbausteine entstehen, der durch die vielfältigen Einsatzmöglichkeiten der Bausteine auch kommerziell interessant sein kann. Die fokussierten Factories schließlich ermöglichen es einer Behörde, sich auf ihr fachliches Kerngeschäft zu fokussieren.

### **Literaturverzeichnis**

- [PLS13] Gerhard Pews, Ralf Leonhard, Simon Spielmann: *Register Factory – Standards für die Cloud*, Proceedings SEE 2013  
[RFWeb] [www.register-factory.de](http://www.register-factory.de)

# Sieben Strategien gegen beißende Hunde

Dipl.-Ing. Dr. Gerald Zincke  
SQS Software Quality Systems GmbH  
Hietzinger Kai 67  
1130 Wien  
gerald.zincke@sqs.com

**Abstract:** Der Erfolg von Projekten hängt auch davon, ob ausreichend Ressourcen für die Qualitätssicherung und den Test zur Verfügung stehen. Das Paper beschreibt sieben Strategien, die, wenn sich das Projekt verzögert und der Plan unter Druck gerät, Testmanagern dabei helfen, eine Reduktion ihrer Ressourcen zu verhindern, den Umfang der Reduktion zu verkleinern oder zumindest die Folgen einer Reduktion zu mildern.

## 1 Einleitung

Das Sprichwort „Den Letzten beißen immer die Hunde“ hat für Software Tester besondere Bedeutung, weil schon seit Jahrzehnten gilt: „Software is always late“ [LA85] und es auch im Rahmen agiler und iterativer Vorgehensmodelle heißt: „getestet wird am Schluss“ (eben mehrmals im Projekt). Wenn Projekte unter Druck geraten und Projektleiter zwischen einer Terminverschiebung, einem Canossagang um mehr Budget, einer Reduktion des Funktionsumfangs oder einer Kürzung beim Tests entscheiden müssen, so entscheiden sie sich nach unserer Erfahrung meist für die Kürzung des Tests. Testmanager benötigen deshalb Strategien, wie sie das Risiko, schwerer, unentdeckter Fehler – für die sie verantwortlich gemacht werden könnten - minimieren können.

## 2 Die Strategien

1) Politik der kleinen Schritte. Diese Strategie steht zum Beispiel hinter Teststufen-Plänen nach ISTQB [IS11]. Der Testaufwand wird in Teststufen aufgeteilt und es wird möglichst früh mit den Testarbeiten begonnen. Zu dem Zeitpunkt, an dem im Projektverlauf eine Verkürzung der Testzeiten zur Diskussion steht, ist dann zumindest ein Teil der Tests schon gelaufen und nicht mehr von Kürzungen betroffen.

2) Das Risiko transparent machen. Ziel der Strategie ist es konkret und nachvollziehbar darzustellen, was in einem verkürzten Test alles nicht getestet werden kann und welche Risiken damit verbunden sind. Die Identifikation der ungetesteten Bereiche ist relativ einfach, wenn die Testfälle mit Anforderungen und Testobjekten verknüpft sind. Um transparent darzustellen, was das für den Anwender bzw. Auftraggeber bedeutet, sind diese Bereiche dem durch die Tests abgedeckten Teil gegenzustellen.

3) Risikominimierung. Durch gute Testvorbereitung ist dafür zu sorgen, dass zumindest Wartezeiten und Leerläufe vermieden und die verbleibende Zeit möglichst effizient genutzt wird. Die wichtigste Maßnahme dazu ist systematische Testvorbereitung, die Voraussetzung für eine Parallelisierung von Tests durch Verteilung vorbereiteter Testfälle an mehrere Tester ist, die Testautomation oder die Methode des Risiko-rasierten Testens (vgl. [BA99]).

4) Das Team absichern. Testressourcen können wirkungsvoll für den Test gesichert werden, indem man für den Test ein eigenes Team aufbaut, das aufs Testen spezialisiert ist. Tester, die nicht programmieren können, können auch nicht dem Testteam entzogen und für Programmierarbeiten eingeteilt werden. Mitarbeiter, die sich selbst als Tester sehen und nicht Testen als eine Rolle (von mehreren) sind zuverlässiger planbar.

5) Starke externe Verbündete sichern. Testressourcen recht zuverlässig vor einem Zugriff geschützt werden, wenn für die Testarbeiten externe Dienstleister beauftragt werden. Der externe Dienstleister hat natürlich ein Interesse daran bzw. oft auch einen vertraglichen Anspruch darauf die Leistung wie vereinbart zu erbringen. Allerdings funktioniert diese Strategie nur dann zuverlässig, wenn der beauftragte Dienstleister auch aufs Testen spezialisiert ist und die Leistungen nicht einfach „umwidmet“.

6) Widerstand leisten. Es ist nicht gesagt, dass ein Testmanager sofort zustimmen muss, wenn das Ansinnen vorgetragen wird, dass die geplante Testzeit verkürzt oder Testressourcen beschnitten werden könnten. Es liegt in der Verantwortung des Testmanagers auch anderen Personen bewusst zu machen, schlimmen Folgen für das Projekt eine Reduktion von Testressourcen haben kann.

7) Schadensbegrenzung. Wenn es nicht möglich ist, mit einer der obigen Strategien ausreichend Testzeit vor dem Produktivsetzungs- bzw. Liefertermin zu sichern bzw. wichtige Testziele zu erreichen, bleibt nur mehr Schadensbegrenzung indem die Tests nach der Produktivsetzung bzw. nach dem Liefertermin fortgesetzt werden. Das ist zumindest besser als den Test ganz zu beenden, unter anderem weil nicht jede neue Funktion von den Benutzern gleich nach der Produktivsetzung auch verwendet wird.

## 2 Zusammenfassung

Die Strategien sollen nicht dazu dienen dem Test im Wettstreit um Ressourcen einen unfairen Vorteil zu verschaffen, sondern dazu beitragen einen Ausgleich zu schaffen. Dieser Ausgleich ist letztendlich wichtig für den Gesamterfolg eines Projekts.

## Literaturverzeichnis

- [BA99] Bach, j, Risk and Requirements Based Testing, IEEE Computer 6/1999
- [IS11] ISTQB, Certified Tester Foundation Level Syllabus 2011,  
<http://www.istqb.org/downloads/finish/16/15.html>.
- [LA85] Lawrence J.L., Why is Software always late? ACM Sigsoft Engineering Notes 19

# **Gegen den Trend? Neue Software-Teststandards ISO/IEC/IEEE 29119**

Matthias Daigl

imbus  
Hauptstraße 8a  
91096 Möhrendorf  
matthias.daigl@imbus.de

Der Trend scheint in eine andere Richtung zu weisen: agile Vorgehensweisen sind vielerorts eingeführt, werden angewendet, verändert... und in den jungen, dynamischen Teams zählt das eigene Urteil, gestützt auf Blogs, Foren und frei zugängliche Informationsquellen. Da erscheinen Standards mit ihrer jahrelangen Entwicklungszeit "old-school". Es ist schwer vorstellbar, wie sie Bezug zu den neuesten Entwicklungen haben können.

Welche Berechtigung hat also heute ein nagelneuer Standard, oder sogar mehrere Standards, zum Thema "Software Testen"?

Zunächst einmal kann sich nicht jeder den agilen Strömungen anschließen. Seien es Projektdimensionen, Inhalte, Risiken oder externe Anforderungen, die dafür sprechen – das sogenannte "klassische" Vorgehen hat häufig seine Berechtigung. Und gerade da, wo Sicherheit benötigt wird, gilt das auch für die Begründung der Vorgehensweise. Sich auf internationale Standards verlassen zu können, ist hilfreich – und Standards stellen den "State of the art" dar. Sie werden erarbeitet von internationalen Experten, abgestimmt durch staatlich anerkannte Organisationen und stellen so einen Wissensschatz dar, der nur genutzt werden will.

Und die Nutzung eines solchen, aufwändig erarbeiteten Wissensschatzes ist natürlich nicht nur beschränkt auf klassische Projektmodelle. Auch für agile Projekte gelten die allgemeingültigen Wahrheiten und Prinzipien, die von der internationalen Gemeinschaft unter Mitarbeit von Experten mit unterschiedlichsten Hintergründen als sinnvoll empfunden werden. Vorausgesetzt, diese Prinzipien passen in den Kontext.

Unter diesen Vorzeichen sind im August 2013 die ISO/IEC/IEEE Standards 29119 in den Teilen 1 bis 3 erschienen, der vierte Teil wird Mitte 2014 erwartet. Die Vorgängerstandards, die hier abgelöst werden sollen, sind prominent: IEEE 829, BS 7925-1, BS 7925-2 und IEEE 1008 sind für Viele alte Bekannte.

Mit den neuen Standards liegt weit mehr als nur eine kosmetische Überarbeitung vor – hier wurde eine ganze Familie von Standards entwickelt, die derzeit die Themen

"Konzepte und Begriffe" (Teil 1), "Testprozesse" (Teil 2), "Testdokumentation" (Teil 3) und "Testtechniken" (Teil 4) im Umfeld Software-Testen umfassen.

Der Vortrag widmet sich den Fragestellungen, wie sich die neuen Standards von ihren Vorgängern abheben, wo Neues dazukommt – zum Beispiel mit der Würdigung der agilen Vorgehensmodelle -, und wo Bewährtes erhalten bleibt. Schließlich geht es darum, wie aus der Anwendung dieser Standards konkreter Nutzen gezogen werden kann.

Unabhängig vom Entwicklungsmodell kann dieser konkrete Nutzen in mehrere Richtungen gehen:

Der Standard kann mit dem Teil 1 als Richtlinie für Softwaretesten verwendet werden. In diesem Teil werden wichtige Konzepte des Softwaretestens erläutert. An erster Stelle steht hierbei das risikobasierte Testen: alle Entscheidungen im Softwaretest sollten im Bezug auf die vorhandenen Risiken getroffen werden, einschließlich der Auswahl der im weiteren Prozess einzusetzenden Praktiken. Die hier vorhandene Darstellung der Grundprinzipien des Testens von Software erleichtert nicht nur den Einstieg in das Thema Softwaretest, sondern wird auch für viele erfahrene Testspezialisten nützlich sein, daBbekanntes strukturiert in einen Kontext gebracht wird.

Weiterer Nutzen liegt in den Testprozessen, die in Teil 2 beschrieben sind. Die hier vorzufindende feine Granularität der Prozesse gab es bisher in keinem Teststandard. Das bedeutet für die Anwender oft sehr wertvolle Anregungen zur Verbesserung der Vorgehensweise. Zudem ist hier ein nicht zu vernachlässigender Gewinn an Sicherheit im Vorgehen zu verzeichnen, da eine Orientierung an den von vielen Testexperten erarbeiteten Leitplanken möglich wird.

Praktiker können darüber hinaus in den Vorlagen für Dokumente große Unterstützung finden. Hier lässt sich für die wichtigsten im Softwaretest vorkommenden Dokumente ablesen, was an Inhalt minimal benötigt wird.

Für Verfechter agiler Vorgehensweisen kann hier der Eindruck entstanden sein, dass dies alles gar nicht zu agilen Ideen passt: schlanke Prozesse, schlanke Ideen – Orientierung nicht an Vorlagen, sondern an Notwendigkeiten steht im Vordergrund.

Arbeiten nach ISO/IEC/IEEE 29119 kann jedoch sehr schlank sein. Während für dokumentationsintensive Großprojekte in einem sicherheitskritischen Bereich von Haus aus mehr dokumentiert wird, kann das in einem kleineren agilen Projekt ganz anders sein - was an Prozessen und Dokumenten nötig ist, kann selber entschieden werden.

Sofern Anspruch auf Konformität besteht, darf nicht beliebig auf alles verzichtet werden, aber die Anpassung der Prozesse und des Dokumentationsgrades ist in den Standards vorgesehen, und es gibt eine Vielzahl von Beispielen in den Standards, wie eine Umsetzung in klassischen, aber gerade auch in agilen Projekten möglich ist.

SE | 14  
SOFTWARE ENGINEERING

## **Workshops**





# **1st Collaborative Workshop on Evolution and Maintenance of Long-Living Systems (EMLS'14)**

Robert Heinrich<sup>1</sup>, Reiner Jung<sup>2</sup>, Marco Konersmann<sup>3</sup>,  
Thomas Ruhroth<sup>4</sup>, Eric Schmieders<sup>3</sup>

<sup>1</sup>Karlsruher Institut für Technologie  
Am Fasanengarten 5, 76131 Karlsruhe  
robert.heinrich@kit.edu

<sup>2</sup>Christian-Albrechts-Universität zu Kiel  
Christian-Albrechts-Platz 4, 24118 Kiel  
reiner.jung@email.uni-kiel.de

<sup>3</sup>Universität Duisburg-Essen  
paluno - The Ruhr Institute for Software Technology  
Gerlingstraße 16, 45127 Essen  
{marco.konersmann, eric.schmieders}@paluno.uni-due.de

<sup>4</sup>Technische Universität Dortmund  
Otto-Hahn-Str. 14, 44221 Dortmund  
thomas.ruhroth@cs.tu-dortmund.de

Langlebige Software- und Automatisierungssysteme sind während ihrer langen Lebensdauer vielen Änderungen der Anforderungen und des Kontextes ausgesetzt, welche zu Problemen bei der Evolution führen (u.a. Architekturerosion, inkonsistente Anforderungen sowie Produktlinien). Langfristig führen diese Probleme zu hohen Kosten bei der Evolution und Wartung der Software. Das Thema wird durch verschiedenartige Evolutionsansätze von Forschung und Industrie adressiert, welche auf unterschiedlichen Perspektiven (u.a. Automatisierung und Softwaretechnik) und Erfahrungen beruhen.

Der Workshop richtet sich an Forscher und Praktiker und basiert auf einem innovativen Konzept, das Erfahrungsaustausch und zukünftige Zusammenarbeit fördern soll. Ziel des Workshops ist es die Diskussion aktueller Fragen, Probleme und Lösungsansätze zum Thema langlebige Software- und Automatisierungssysteme zu fördern. Daher ist dieser Workshop als Arbeitstreffen mit Diskussionsgruppen konzipiert. Der Workshop wird mit einem Vortrag aus der Industrie eingeleitet, um die Perspektive und die Herausforderungen der Industrie zu verdeutlichen. In mehreren Arbeitsgruppen wird daraufhin in je einem kurzen Vortrag, dem Problem-Statement, ein Problem zusammen mit den benötigten Hintergrundinformationen vorgestellt. Dieses Problem-Statement schließt mit der Vorstellung einer offenen Fragestellung, dem Problem, ab, welches in der Arbeitsgruppe diskutiert

wird. In dieser - durch Moderationstechniken unterstützten - Diskussion werden verschiedene Lösungsansätze und -ideen aus unterschiedlichen Domänen gesammelt und deren Anwendbarkeit auf das Problem diskutiert. Mit diesem Aufbau soll einerseits eine Sensibilität für die Herausforderungen in Forschung und Industrie erzeugt werden, andererseits wird eine Plattform für Forschungs- und Projekt-Kooperationen geschaffen. Die Ergebnisse des Workshops werden auf der Workshop-Webseite<sup>1</sup> gesammelt veröffentlicht.

---

<sup>1</sup><http://www.spp-1593.de/emls14/>

## 7. Arbeitstagung Programmiersprachen (ATPS 2014)

Volker Stolz<sup>1</sup>      Baltasar Trancón Widemann<sup>2</sup>

<sup>1</sup>Institutt for informatikk, Univ. Oslo  
Gaustadalléen 23B, N-0373 Oslo, Norwegen  
stolz@ifi.uio.no

<sup>2</sup>Fakultät für Informatik und Automatisierung, TU Ilmenau  
Ehrenbergstr. 29, D-98693 Ilmenau, Deutschland  
baltasar.trancon@tu-ilmenau.de

Die Tagung dient dem Austausch zwischen Forschern, Entwicklern und Anwendern, die sich mit Themen aus dem Bereich der Programmiersprachen beschäftigen. Dabei sind alle Programmierparadigmen gleichermaßen von Interesse: imperative, objektorientierte, funktionale, logische, parallele, graphische Programmierung, auch verteilte und nebenläufige Programmierung in Intra- und Internet-Anwendungen, sowie Konzepte zur Integration dieser Paradigmen. Typische, aber nicht ausschließliche Themenbereiche der Tagungsreihe insgesamt und auch dieser Tagung sind:

- Entwurf von Programmiersprachen und anwendungsspezifischen Sprachen
- Implementierungs- und Optimierungstechniken
- Analyse und Transformation von Programmen
- Ressourcenanalyse (Zeit, Speicher, Leistungsverbrauch)
- Typsysteme
- Semantik und Spezifikationstechniken
- Modellierungssprachen, Objektorientierung
- Intra- und Internet-Programmierung
- Programm- und Implementierungsverifikation
- Werkzeuge und Programmierumgebungen
- Frameworks, Architekturen, generative Ansätze
- Erfahrungen bei exemplarischen Anwendungen
- Verbindung von Sprachen, Architekturen, Prozessoren

Ebenfalls von Interesse für die Tagungsreihe insgesamt und auch diese Tagung sind Arbeiten zu Techniken, Methoden, Konzepten oder Werkzeugen, mit denen Sicherheit und Zuverlässigkeit bei der Ausführung von Programmen erhöht werden können. Neben neuen Arbeiten sind stets auch Beiträge erwünscht, die existierende Arbeiten oder Projekte zusammenfassen oder aus einem anderen und neuen Blickwinkel präsentieren und so insbesondere einem deutschsprachigen Publikum vorstellen. Die Tagung richtet sich ausdrücklich auch an Interessenten aus Wirtschaft und Industrie.

## **Programmkomitee**

Eric Bodden	TU Darmstadt, DE
Clemens Grelck	Univ. Amsterdam, NL
Michael Hanus	CAU Kiel, DE
Christian Heinlein	HTW Aalen, DE
Jens Knoop	TU Wien, AT
Herbert Kuchen	Univ. Münster, DE
Ralf Lämmel	Univ. Koblenz-Landau, DE
Andres Löh	Well-Typed LLP, UK
Thomas Noll	RWTH Aachen, DE
Ina Schaefer	TU Braunschweig, DE
Volker Stolz	Univ. Oslo, NO, Co-Vorsitzender
Baltasar Trancón Widemann	TU Ilmenau, DE, Co-Vorsitzender
Wolf Zimmermann	Univ. Halle-Wittenberg, DE

# **CeMoSS - Certification and Model-Driven Development of Safe and Secure Software**

Software is a key factor driving the innovation of many technical products and infrastructures for everyday use. Dependable software requires rigorous quality assurance in particular to achieve an adequate level of safety and information security. In many domains like avionics, power generation and distribution, industrial automation, railway and automotive, as well as medical devices and health information systems, dependable systems and the software therein have to be formally approved with respect to safety and security and certified according to international standards, before being put in operation. In spite of all domain-specific singularities, the following issues challenging the development of safe and secure software shall be addressed in the workshop and discussed by a cross-domain audience:

- Model-driven software development with extensive tool support becomes more and more accepted in industry. Although safety standards recommend the formal foundations and systematics a model-driven approach relies on, advanced features of model driven development frameworks like automated code generation, model transformation and model checking are not yet addressed in detail in the standards. Thus using new methods and tools brings new risks into the certification of a product since it requires additional arguments in the assurance case just because the development deviates from the best-practice procedure, even if there is strong evidence that the new approach outperforms the older one with respect to error avoidance or error disclosure.
- Critical infrastructures and dependable systems are no longer operated in isolation, but connected to other company networks or accessed by mobile devices using the public telecommunication infrastructure like in smart home environments or car to car communication. Connectivity is an enabling factor for enhanced services, but also raises new threats from the newly introduced dependency between functional safety and IT security. New modeling approaches shall integrate safety and IT security issues in risk and safety analysis and design. Open issues are in particular the integration of safety and security processes, risk acceptance criteria that take both, safety and security into account, even for architectures of commercial systems with a life-cycle of a few decades and a holistic view on safety and security in the assurance case. A particular challenge arises in safety critical industrial infrastructures where life cycles are much longer than 10 years. Then security updates may be required without impact on functional safety and the resp. safety cases and certificates.
- Open source software has been introduced in the realm of dependable systems as tools in supporting processes, but first usages of open source components as part of

the dependable system itself are under development (See the European project OpenETCS). In case open source software is used exactly as versioned and documented, it may be certified according to IEC 61508 or other relevant standards. However, the development or adaption of dependable software in an open source project raises new questions about adequacy of processes, tools and the competences of personnel as well as liability issues. Finally, the balance between the developing and operating staff and bodies, and the users of dependable systems has to be newly discussed.

Topics of interest include but are not limited to: risk analysis, integration of safety and security, seamless tracing and decomposition of safety and security constraints, safety and assurance cases, modular certification, qualification of methods and tools, external proof checking of formal validation & verification results. The CeMoSS-workshop shall foster the cross-domain discussion of challenges and possible solutions in the development of high assurance systems and infrastructure between academia and industry. Thus contributions reporting on original research ideas as well as experience reports raising open questions from industrial practice are most welcome.

December 2013

Michaela Huhn (TU Clausthal)  
Stefan Gerken (Siemens AG)  
Carsten Rudolph (Fraunhofer SIT)  
PC chairs CeMoSS'14

# **International Workshop on Comparison and Versioning of Software Models (CVSM 2014)**

Pit Pietsch<sup>1</sup>, Udo Kelter<sup>1</sup> and Jan Oliver Ringert<sup>2</sup>

<sup>1</sup> Universität Siegen

Hölderlinstraße 3, 507076 Siegen, Germany

pietsch, kelter@informatik.uni-siegen.de

<sup>2</sup> Software Engineering RWTH Aachen University

Ahornstraße 55, 52074 Aachen, Germany

ringert@se.rwth-aachen.de

The International Workshop Series on Comparison and Versioning of Models brings together scientists and practitioners in the field of model versioning. Particularly technologies like model comparison and differencing, model patching and model merging are addressed. Subtopics of interest include, but are not limited to, differences between models, recognition of user operations in differences, merging and patching of models, versioning for meta models, quality aspects of model merging and patching, formal approaches to variant management, visualization of model differences and applications of model differences in Software Engineering. All mentioned topics and technologies are indispensable to support model-based development methods and a significant number of algorithms and tools in this area have been developed in the last decade. Still, good model versioning tools seem to be unavailable in many practical contexts. Existing prototypes appear to have been used mainly by "innovators" or "early adopters" of the new technology. The extent in which state-of-the-art tools are meeting, or not meeting, practical requirements and the reasons of any deficiencies are not well documented. This workshop aims at collecting and consolidating experience gained in this new technology, at distinguishing unresolved from solved questions, at identifying reasons why questions have remained unsolved, and at identifying new technical challenges which emerged after first practical applications.

Furthermore, CVSM'14 has a special focus on the advances of the community benchmark set which has been initiated in the last issues of the workshop series. The benchmark sets aim at comparing model differencing approaches with respect to coverage of requirements, performance or tool integration issues. To this end, several benchmarks are made available on the workshop website. Developers of model comparison and differencing tools are invited to present the differences reported by their algorithms, to compare them with the expected differences as specified in the benchmark and to discuss, if applicable, the effort needed to configure the algorithms.





# Workshop „Lehre für Requirements Engineering“ (LehRE)

Andrea Herrmann<sup>1</sup>, Anne Hoffmann<sup>2</sup>, Dieter Landes<sup>3</sup>, Rüdiger Weißbach<sup>4</sup>

<sup>1</sup>Freie Software Engineering Trainerin  
Daimlerstr. 121, 70372 Stuttgart  
herrmann@herrmann-ehrlich.de

<sup>2</sup>Siemens AG  
Freyeslebenstr. 1, Erlangen  
anne.hoffmann@siemens.com

<sup>3</sup>Hochschule für angewandte Wissenschaften Coburg  
Friedrich-Streib-Str. 2, 96450 Coburg  
dieter.landes@hs-coburg.de

<sup>4</sup>Hochschule für Angewandte Wissenschaften (HAW) Hamburg  
Berliner Tor 5, 20099 Hamburg  
ruediger.weissbach@haw-hamburg.de

**Abstract:** LehRE ist ein Workshop über Lehre und Training für Requirements Engineering. Der Workshop dient dem Erfahrungsaustausch zwischen Personen, die RE im akademischen oder im berufsbegleitenden Umfeld lehren. Auf der Basis eigener Erfahrungen der Teilnehmer soll ein tieferes Verständnis dafür entstehen, welche Methoden unter welchen Bedingungen sinnvoll in der Lehre eingesetzt werden können.

## 1 Zielsetzung des Workshops

Der Workshop dient dem Erfahrungsaustausch zwischen Personen, die RE im akademischen oder im berufsbegleitenden Umfeld lehren.

Die Teilnehmer des Workshops sollen - erfolgreiche, aber gerne auch fehlgeschlagene - Beispiele für Seminarkonzepte und einzelne Übungen aus ihrer eigenen Erfahrung mitbringen, sie im Workshop vorstellen und dort diskutieren. Kürzere Aufgaben können im Workshop mit den Teilnehmer/innen durchgespielt werden. Als Ergebnis soll ein tieferes Verständnis dafür entstehen, welche Methoden unter welchen Bedingungen sinnvoll in der Lehre eingesetzt werden können.

Die Organisatoren des Workshops haben lange Lehrerfahrungen, auch und gerade im Bereich RE. Sie sind Gründungsmitglieder des 2012 gegründeten Arbeitskreises „Requirements Engineering in der Lehre“ der Fachgruppe 2.1.6, Requirements Engineering (RE), der Deutschen Gesellschaft für Informatik e.V. (GI).

## 2 Zielgruppe

Der Workshop richtet sich an Personen, die Requirements Engineering (RE) schulen und lehren. Von Interesse sind Erfahrungsberichte aus der Lehre im Hochschulbereich, aber auch aus Schulungen in der beruflichen Weiterbildung, wie Aspekte des Requirements Engineering effektiv vermittelt, Problembewusstsein für Requirements Engineering erzeugt und relevante Kompetenzen trainiert werden können. Insbesondere sollen die Teilnehmer/innen Übungen etc., die sich in ihrem Umfeld bewährt haben (oder sich als problematisch erwiesen haben) vorstellen, diskutieren und möglichst praktisch präsentieren.

## 3 Workshop-Ziele

Erwartete Ergebnisse des Workshops sind

- eine dokumentierte Sammlung von „Good Practices“ für die Lehre im Bereich RE unter Berücksichtigung der jeweiligen Rahmenbedingungen (z.B. Einsatz bei der Entwicklung von kaufmännischen Systemen oder Embedded Systems, Einsatz in linearen oder iterativen Vorgehensmodellen etc.; universitäre Lehre, kommerzielle Schulungen oder Zertifizierungsvorbereitung),
- ggf. eine Sammlung von „Fallstricken“,
- ggf. eine Gegenüberstellung von „Good Practices“ aus der Lehre des Hochschulwesens und aus beruflichen Trainings, die im jeweils anderen Bereich entweder (noch) nicht verbreitet oder nicht akzeptiert sind.

## 4 Einreichungen

Einzureichen ist eine mindestens zweiseitige Beschreibung des Beitrags im LNI-Format: <http://www.gi.de/service/publikationen/lni.html>. Reichen Sie Ihren Beitrag bitte per E-Mail ein unter [herrmann@herrmann-ehrlich.de](mailto:herrmann@herrmann-ehrlich.de). Alle Beiträge werden von mindestens zwei Gutachter/innen im Blind Peer Review begutachtet. Die Beiträge sollen nach Annahme zu einem sechsseitigen Artikel ausformuliert werden für die Verteilung als Handouts auf dem Workshop.

Auswahlkriterien für die Annahme sind:

- Klare Beschreibung der Lehrmethode, möglichst so, dass sie wiederholt werden kann
- Evaluierung der Lehrmethode und überzeugender Beleg für ihre Nützlichkeit
- Wissenschaftliche Qualität des Beitrags
- Neuheit der Veröffentlichung (nicht unbedingt der beschriebenen Methode)

## **4. Workshop zur Zukunft der Entwicklung softwareintensiver eingebetteter Systeme (ENVISON2020)**

Ottmar Bender<sup>1</sup>, Wolfgang Böhm<sup>2</sup>, Stefan Henkler<sup>3</sup>, Oliver Sander<sup>4</sup>,  
Andreas Vogelsang<sup>2</sup>, Thorsten Weyer<sup>5</sup>

<sup>1</sup> EADS Deutschland GmbH  
Geschäftsbereich Cassidian  
Wörthstr. 85  
89077 Ulm  
[ottmar.bender@cassidian.com](mailto:ottmar.bender@cassidian.com)

<sup>2</sup> Institut f. Informatik  
Technische Universität München  
Boltzmannstr. 3  
85748 Garching b. München  
[boehmw@in.tum.de](mailto:boehmw@in.tum.de)  
[vogelsan@in.tum.de](mailto:vogelsan@in.tum.de)

<sup>3</sup> OFFIS e.V.  
Industriebereich Verkehr  
Escherweg 2  
26121 Oldenburg  
[stefan.henkler@offis.de](mailto:stefan.henkler@offis.de)

<sup>4</sup> Karlsruher Institut für Technologie (KIT)  
Institut für Technik der Informationsverarbeitung  
Engesserstr. 5  
76131 Karlsruhe  
[sander@kit.edu](mailto:sander@kit.edu)

<sup>5</sup> paluno – The Ruhr-Institute for Software Technology  
Universität Duisburg-Essen  
Gerlingstr. 16  
45127 Essen  
[thorsten.weyer@paluno.uni-due.de](mailto:thorsten.weyer@paluno.uni-due.de)

Heutzutage unterstützen softwareintensive eingebettete Systeme mehr oder weniger sichtbar den Menschen in vielen Bereichen des täglichen Lebens. Experten prognostizieren für die Zukunft eine rasante Zunahme softwareintensiver, eingebetteter Systeme und deren Vernetzung in Systemverbünden. Durch den Einsatz von Multicore-Technologien wird eine entsprechende technologische Basis geschaffen. Diese Entwicklung wird in dramatischer Weise durch das entstehen umfassender "Cyber Physical Systems" verstärkt, die in immer stärkeren Maße die transparente Integration von Softwaresystemen und der realen Welt forcieren.

Existierende Ansätze und Methoden müssen aufgrund der wachsenden Herausforderungen in Frage gestellt und in Teilen neu konzipiert werden. Dabei ist es wesentlich diese neuen Entwicklungsansätze in einem repräsentativen Industriekontext zu validieren.

Der Workshop ENVISION 2020 verfolgt das Ziel, die Entwicklung und Diskussion zukünftiger Ansätze in den Bereichen Multicore, sowie Vorgehensweisen und Methoden zur Entwicklung softwareintensiver, eingebetteter Systeme anzuregen und deren Überführung in die industrielle Praxis zu fördern.

Wesentliches Ergebnis des Workshops sollte ein intensiver Erfahrungsaustausch zwischen Akademia auf der einen und Vertretern der Industrie auf der anderen Seite sein. Es soll ein gemeinsames Verständnis entwickelt werden, wie und in welchen Schritten modellbasierte Ansätze zur Entwicklung eingebetteter Systeme in die industrielle Praxis überführt werden können.

Im Workshop werden die Beiträge in Vortrags-Sessions präsentiert. Für jeden Beitrag ist ein Moderator vorgesehen, der die Diskussion durch vorbereitete Fragen basierend auf einer Begutachtung des Papiers unterstützt.

In drei Kategorien von Beiträgen werden im Rahmen des Workshops jeweils dedizierte Fragestellungen adressiert:

**Kategorie 1 "Grundlagen und Methoden":** Beiträge dieser Kategorie sollen aktuell erkannte Herausforderungen beschreiben und Grundlagen und Methoden zu deren Lösung vorstellen. Die Lösungsansätze und Methoden sind anhand eines aussagekräftigen Beispiels (zum Beispiel durch Entwicklungsartefakte) zu illustrieren.

**Kategorie 2 "Werkzeuge":** In dieser Kategorie sollen Ansätze für die Entwicklung neuer Werkzeuge zur Umsetzung der Methoden vorgestellt werden.

**Kategorie 3 "Transfer in die Praxis":** Fallstudien und Lösungsansätze zum Transfer neuer Methoden in die industrielle Praxis. Papiere dieser Kategorie sollen die positiven und/oder negativen Erfahrungen bei dem Transfer bzw. bei der Evaluation eines neuen Ansatzes zur Lösung einer bekannten Herausforderung in der Industrie beschreiben und nach Möglichkeit anhand eines Beispiels illustrieren.

Dieser Workshop wird gemeinsam organisiert von den BMBF Förderprojekten "Automotive, Railway and Avionics Multicore Systems" - ARAMiS und "Softwareplattform Embedded Systems 2020 XT" - SPES\_XT.

SE | 14  
SOFTWARE ENGINEERING

## **Tutorien**



# **Tutorial: Zukunftssichere Software Systeme mit Architekturbewertung: Wann, Wie und Wieviel?**

Thorsten Keuler, Jens Knodel, Matthias Naab

Fraunhofer IESE  
Fraunhoferplatz 1  
67663 Kaiserslautern  
thorsten.keuler@iese.fraunhofer.de  
jens.knodel@iese.fraunhofer.de  
matthias.naab@iese.fraunhofer.de

Moderne Software Systeme sind sehr komplex und müssen üblicherweise durch große Entwicklungsteams erstellt, getestet, erweitert und gewartet werden. Der Schlüssel, um Software Systeme zukunftssicher zu machen, liegt in deren Softwarearchitektur. Die Softwarearchitektur beschreibt essentielle Zusammenhänge zwischen Softwareeinheiten, den dabei verwendeten Technologien, der Aufteilung der Software auf Teams, oder auch die physische Verteilung von Software in der realen Welt – also genau die getroffenen Entscheidungen, die sich bei eintretenden Änderungen positiv oder negativ auswirken.

Um die Auswirkungen hinsichtlich der Erfüllbarkeit von Qualitäts-, Kosten-, und Terminzielen abschätzen zu können, haben sich Architekturbewertungen als effektives Mittel bewährt. Architekturbewertungen sollten dabei jedoch nicht nur zu Beginn einer Systementwicklung eine Rolle spielen, sondern über den gesamten Software-Lebenszyklus hinweg systematisch zu verschiedenen Zeitpunkten sinnvoll zum Einsatz kommen.

In der Praxis bietet sich jedoch meist ein eindeutiges Bild: Im Rahmen von mehr als 50 Architekturbewertungen bei Kunden unterschiedlichster Branchen hat das Fraunhofer IESE immer wieder folgende, typische Problembilder identifizieren können:

- 1.) Architekturen, die den Anforderungen nicht (mehr) angemessen sind.
- 2.) „Mis-match“ zwischen Architekturen zu integrierender Systeme.
- 3.) Keine oder rein zufällige Verbindung von Architekturkonzepten und der Implementierung.
- 4.) „Mis-match“ zwischen Architektur und umsetzender Organisation, geplanten Entwicklungsprozessen oder Projektplänen.



Im Vortrag wird eine Auswahl konkreter Fragestellungen aus den Projekten, mit dabei angewandten Methoden, Techniken und Ergebnissen, vorgestellt – wie zum Beispiel die Bewertung von Migrationsentscheidungen (Neuentwicklung vs. Restrukturierung), Technologieauswahl (Anbieter A vs. Open Source), Auftraggeber-Auftragnehmer-Situationen (neutrale Begutachtung der Qualität, interne Konfidenzbildung) und Begleitung bei langfristigem Qualitäts- und Risikomanagement.

In diesem Tutorial stellen wir eine anpassbare Bewertungsmethode vor, mit der solche Problembilder frühzeitig erkannt, die Konsequenzen abgeschätzt, und Gegenmaßnahmen ergriffen werden können.

Es wendet sich an alle Praktiker (Architekten, Projektleiter, Senior-Developer, ...) und Entscheider, die erfahren wollen, wie man systematische Architekturbewertung einsetzen kann, um die Zukunftsfähigkeit ihrer Software und Systemlandschaften zu bewerten und nachhaltig zu sichern.

# **Der Werkzeug-und-Material-Ansatz für die Entwicklung interaktiver Software-Systeme**

Guido Gryczan, Henning Schwentner

Universität Hamburg  
Arbeitsbereich Softwaretechnik  
und  
C1 WPS GmbH  
Vogt-Kölln-Straße 30  
22527 Hamburg  
guido.gryczan@informatik.uni-hamburg.de  
henning.schwentner@c1-wps.de

Der Werkzeug- und Materialansatz (WAM-Ansatz) ist eine iterativ inkrementelle Methode zur anwendungsorientierten Entwicklung interaktiver Software.

Der WAM-Ansatz wurde Ende der 80er-Jahre des letzten Jahrhunderts bei der Gesellschaft für Mathematik und Datenverarbeitung ausgearbeitet. Seitdem wurde er in zahlreichen universitären und Praxisprojekten erfolgreich angewendet und weiterentwickelt.

Ursprünglich für die Konstruktion von Programmierumgebungen konzipiert, wird der Ansatz heute an verschiedenen deutschsprachigen Universitäten als Methode zur Software-Entwicklung interaktiver Systeme gelehrt und in der industriellen Praxis verwendet.

Die transdisziplinäre Wurzel des Ansatzes besteht darin, die in der Softwaretechnik gebräuchlichen Konzepte „Werkzeug“, „Automat“ und „Material“ auf eine philosophisch und arbeitspsychologisch tragfähige Basis zu stellen. Namentlich die Hermeneutik Heideggers und die Tätigkeitstheorie nach Leontjev spielen eine wesentliche Rolle.

Auf dieser Basis werden Richtlinien für das fachliche Design und die (objektorientierte) Architektur und Konstruktion interaktiver Softwaresysteme entwickelt. Ergänzt um eine evolutionäre Vorgehensweise entsteht damit eine Methode, die erfolgreich fachliche und organisatorische Erfordernisse der Anwendungsentwicklung technisch umsetzbar macht.

Im Tutorium werden wir fundamentale Konzepte des Ansatzes an industriellen Beispielen behandeln:

- Die Rolle von Leitbildern und Entwurfsmetaphern für die Anwendungsentwicklung

- Software-Werkzeuge, -Automaten und -Materialien in der Arbeitsumgebung
- WAM und Geschäftsprozeßsteuerung (Business Process Modelling (BPM))
- Der Zusammenhang von fachlichem Modell und technischer Architektur
- Iterativ inkrementelle Vorgehensweise und agile Methoden – Die WAM-Interpretation
- Der WAM-Architekturstil und der Zusammenhang zu Service-Orientierten Architekturen

Sämtliche vorgestellten Konzepte des WAM-Ansatzes werden an Beispielen aus der industriellen Praxis (u.a. aus der medizinischen Versorgungsforschung) erläutert.

Übergeordnetes Ziel des Tutoriums ist es zu verdeutlichen, dass eine solide fachliche Orientierung jenseits technischer und betriebswirtschaftlicher Moden eine tragfähige Basis für langfristig erfolgreich einsetzbare Software bildet.

# Früherkennung fachlicher und technischer Projektrisiken mit dem Interaction Room

Simon Grapenthin, Matthias Book, Volker Gruhn

paluno – The Ruhr Institute for Software Technology  
Universität Duisburg-Essen  
Gerlingstr. 16, 45127 Essen

{simon.grapenthin, matthias.book, volker.gruhn}@paluno.uni-due.de

**Abstract:** Zur Unterstützung der industriellen Software-Technik existiert eine Vielzahl an Methoden, Notationen und Werkzeugen. Trotzdem überschreiten viele Software-Projekte ihren finanziellen und zeitlichen Rahmen oder werden im Extremfall abgebrochen. Meist liegt der Grund in unzureichender Zusammenarbeit der Stakeholder und zu später Erkennung bzw. Auseinandersetzung mit fachlichen und technischen Risiken. Mit dem Interaction Room vermitteln wir eine Methode, um die Kommunikation im Team pragmatisch auf diese Herausforderungen zu fokussieren. Stakeholdern mit unterschiedlichen Hintergründen wird so eine Kommunikationsbasis gegeben, die ohne hohe Werkzeuginvestitionen und ohne hohen Lernaufwand genutzt werden kann.

## 1 Motivation

Software-Entwickler haben heute die Wahl unter einer Vielzahl von Prozessmodellen; in Projekten steht ihnen ein breites Spektrum an Notationen und Werkzeugen zur Unterstützung konstruktiver und organisatorischer Arbeiten zur Verfügung. Trotz der umfangreichen Hilfsmittel überschreiten viele Entwicklungsprojekte jedoch ihre zeitlichen und finanziellen Rahmenbedingungen, verfehlen notwendige funktionale und Qualitätsanforderungen oder werden sogar vorzeitig abgebrochen [BF11]. In vielen Fällen lässt sich der vertane Aufwand auf Mängel in der Kommunikation und Koordination im Softwareprozess zurückführen [Ar11].

Agile Prozessmodelle werden als Lösung für solche Kommunikationsprobleme beworben – sie sollen das gemeinsame Verständnis zwischen Business und IT-Stakeholdern fördern, geben jedoch tatsächlich nur einen organisatorischen Rahmen zur Strukturierung von Aufgaben und Meetings vor und liefern kaum Hilfestellung bei der inhaltlichen Auseinandersetzung mit den Projektherausforderungen. Lösungsvorschläge aus dem akademischen Bereich münden oft in ausgesprochen formalen Verfahren, die zu komplex für den kommerziellen Einsatz sind, oder Werkzeugen, deren Reifegrad für den praktischen Einsatz selten ausreicht [Wh07]. Wünschenswert wäre eine Methode, um weniger gut verstandene, besonders komplexe oder wertschöpfende inhaltliche Aspekte eines Projekts in den Fokus zu rücken und so die Kommunikation und Interaktion im Team gezielt auf die daraus erwachsenden Projektrisiken zu lenken. Eine Herausforderung dabei ist, dass Projektrisiken sowohl technischer als auch fachlicher

Natur sein können, das Wissen über sie also verteilt ist. Daher muss eine Kommunikationsbasis geschaffen werden, die alle Stakeholder nutzen können, ohne besondere Kenntnisse über die Domäne anderer Stakeholder zu besitzen. Erst auf dieser Grundlage wird ein effektiver Austausch über Unsicherheiten und Projektrisiken möglich.

## 2 Tutoriums-Inhalte

Um ein gemeinsames Verständnis für die Spezifika eines Projekts herzustellen, stellen wir das Konzept des sogenannten Interaction Rooms vor [BGG12]. Ein Interaction Room ist ein physisch begehbbarer Raum, dessen Wände mit großen Whiteboards und Pinnwänden ausgestattet sind. An den Wänden des Interaction Rooms kann ein Informationssystem skizzenhaft auf hoher Abstraktionsebene aus verschiedenen Perspektiven (Prozesse, Daten, Interaktion, Integration, Migration) modelliert werden. Durch diese Rundumsicht auf das Projekt können Zusammenhänge und Abhängigkeiten schnell identifiziert und diskutiert werden. Der Verzicht auf formale Notationen und lernintensive Werkzeuge ermutigt insbesondere IT-ferne Stakeholder zur aktiven Teilnahme. Die skizzenhafte Darstellung der Modelle betont die Veränderlichkeit und Unsicherheit der Modelle und fördert innovative Lösungen.

Zur Veranschaulichung und Dokumentation von Wert- und Aufwandstreibern – d.h. Aspekten, von denen besonderer Nutzen erwartet wird, oder die besondere Risiken darstellen können – stellen wir eine Palette grafischer Annotationen vor, die in die Modelle im Interaction Room geheftet werden können [BGG13]. Wir zeigen, wie durch die Diskussion über diese Annotationen Unsicherheiten und Risiken identifiziert und dazu Lösungsstrategien, Aufwandsschätzungen, etc. entwickelt werden können. Auf Basis der Annotation der Modelle durch die Stakeholder lassen sich Metriken ableiten, die zur Risikobeurteilung und zur Projektsteuerung (Priorisierung, Kosten-/Nutzenverhältnis von User Stories etc.) genutzt werden können.

## Literaturverzeichnis

- [Ar11] Arnuphaptrairong, T.: Top Ten Lists of Software Project Risks: Evidence from the Literature Survey: Proceedings of the International MultiConference of Engineers and Computer Scientists (IMCECS'11), 2011.
- [BF11] Budzier, A.; Flyvbjerg, B.: Why Your IT Project Might Be Riskier than You Think: Harvard Business Review, 2011; S. 1–4.
- [BGG13] Book, M.; Grapenthin, S.; Gruhn, V.: Risk-aware Migration of Legacy Data Structures: 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2013), 2013.
- [Wh07] Whitehead, J.: Collaboration in Software Engineering: A Roadmap: Future of Software Engineering, 2007. FOSE '07, 2007; S. 214–225.

SE | 14  
SOFTWARE ENGINEERING

## **Doktorandensymposium**



# AGREEMENT - An Approach for Agile Rationale Management

Michaela Gluchow

Fakultät für Informatik (TU München) | Cassini Consulting  
michaela@gluchow.de

**Abstract:** Agile methods affected IT project management in many ways. This also applies for the area of decision making processes. In agile projects it is more common to spread the responsibilities equally among the project roles, instead of distributing it along hierarchies. Further groups are now more often in charge than before. This led to an increased importance of information sharing and transparency.

Coming from this situation, we assume that agile methods would benefit from the utilization of Rationale Management (RM). As none of the existing RM approaches seem to do the job right away, we propose our own approach called AGREEMENT, that specifically copes with the needs of agile methods. AGREEMENT is currently in the state of a rough draft, but early case studies already strengthened our hypothesis. This paper describes the current status of AGREEMENT as well as which actions are planned to reach a refined, mature version of it.

## 1 Motivation

Agile methods changed the way of managing IT projects drastically. This also includes the way of decision making. Decisions are no longer made hierarchically, but are distributed across the different project roles. Further it is more usual that a group of people rather than a single person is in charge of making decisions [MB09]. Therefore information sharing and transparency across all stakeholders is more crucial than ever before [Hal07].

The idea of Rationale Management (RM) is to support this subject area [DMMP06]. Since Kunz and Rittel started with IBIS in 1970 [KR78] a variety of approaches<sup>1</sup> was developed. Although a lot of attempts were undertaken, none of these approaches made its way into IT project management (PM). Prevented was this amongst others by the widespread opinion that RM is time-consuming, disruptive and too inflexible [DMMP06].

This leads to the hypothesis on which the herein described thesis is based on. The hypothesis is, that agile methods would greatly benefit from RM. But unfortunately there seems to be no existing approach that would do the job. As they were even too heavyweight for classical IT PM approaches, they are not considered to fulfill the needs of agile methods. Therefore this thesis develops AGREEMENT<sup>2</sup>, the first specifically agile RM approach.

---

<sup>1</sup>Examples: Procedural Hierarchy of Issues (PHI); Questions, Options, and Criteria (QOC); Decision Representation Language (DRL); Design Recommendation and Intent Model (DRIM). [DMMP06]

<sup>2</sup>AGREEMENT is an acronym for **AGile Rationale Management**



In our opinion AGREEMENT will only be successful if it provides a good usability [Rou07]. Further it has to blend seamlessly into existing management processes in order to be accepted by the users as integral and important part of PM. But AGREEMENT should not reinvent the wheel. Instead it should combine existing solutions, that fit to the circumstances, with new ideas, that fill the remaining gaps.

The following sections describe the current draft of AGREEMENT including two case studies and give an overview of the further course of action.

## 2 Early Results

The idea for AGREEMENT arose in the context of the large<sup>3</sup> project courses we conduct each semester at the chair for Applied Software Engineering (TU München). As project managers we realized, that it is hard to keep an overview of which decisions are made, by whom and why. Further we noticed that students often struggle with decision making processes. If they at all apply an structured approach, they often tend to loose track within it.

In the winter term 2012/2013 we therefore started with the integration of RM aspects into our courses. As model we used an adapted version of the RM model introduced in [BD09]. It is based on issues as root elements and further consists of proposals, criteria, and resolutions (Figure 1).

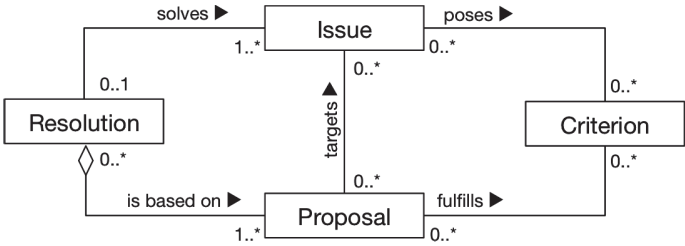


Figure 1: Model used in the first version of AGREEMENT

One of our goals was to blend RM seamlessly into our PM. As meetings were one of our central organizational elements, we made it part of them. Our meetings consist of a status report (similar to the daily Scrum) and a discussion part. In the status report issues replaced blockers/impediments. Further they were used to report on important decisions, that were made since the last meeting. In the discussion part, issues build the basis in form of agenda items.

To support the tracking of work, requirements as well as of rationale we used the project tracker Atlassian JIRA [Atl13] as tool. Each AGREEMENT model element was mapped

<sup>3</sup>From 40 to 90 students organized in different projects and teams with 4 to 8 members.

to an individual JIRA element. These could be linked amongst each other with labeled links according to the model. Also linking to all other elements like requirements or action items was possible, so that AGREEMENT could be used to track open issues to basically everything done in the project.

Overall the general idea of integrating RM into the course went well, except that we had big usability problems. Having one JIRA element for each model element caused too much work in creating and linking, as well as it led to an unclear representation. Further the opportunity of linking elements of different decisions (e.g. linking one resolution to more than one issue) amongst each other was under-utilized.

In the follow up case study we therefore tried to improve the usability by combining all model elements in one JIRA element. We further advanced the support of Scrum specific practices by integrating AGREEMENT into the requirements elicitation process. Open questions on user stories were formulated as issues. Only if all issues linked to an user story were resolved, this story was considered as fully understood by the team and therefore detailed enough to be taken over into the sprint backlog.

Both studies showed with anecdotic evidence, that it is possible and useful to integrate RM into agile projects. But they also made clear, that seamless integration into the processes as well as good usability are crucial elements for its acceptance. This is the starting point for the further development of AGREEMENT, which is described in the following.

### 3 Course of Action

The early versions of AGREEMENT were developed from a very pragmatic point of view and therefore mainly considered project specific needs. Now AGREEMENT has to be enhanced in order to support various agile methods. The first step to reach this is to formulate appropriate requirements. The following sources will be taken into account.

- **Agile methods**

Decision making processes of two to four well-established agile methods will be analysed. Questions that should answered here are: Which processes exist? How are the decision making responsibilities handled? Where can these agile methods benefit from RM? The so found requirements will be enriched by general guidelines retrieved from the agile mindset.

- **Rationale management**

Similar to the agile methods, the area of RM will be examined. This includes on the one hand a general overview of RM and its usage scenarios. On the other hand concrete RM approaches will be analysed regarding their benefits and deficiencies as well as their reusable elements. Special attention will be turned to mentioned usability issues of the approaches.

- **Tool support**

Good tool support seems to be another crucial element of an RM approach. If possible, AGREEMENT should be integrated into already used tools, as we hope to

reach a better integration into existing processes. The tool solution should be easy-to-learn, easy-to-use but powerful [Rou07]. Another goal is to automate AGREEMENT as far as possible in order to reduce the workload associated with RM.

#### – Social factors

Besides RM's reputation of being too heavyweight, social factors are assumed as another reason for not being successful. Therefore these also have to be taken into account. Possible phenomena to look on are social networks, active discourse, culture of blame, or group thinking.

The so elicited requirements will then be implemented in the current version of AGREEMENT. The goal is to provide a full-fledged RM approach that suites different agile methods. AGREEMENT should further include proposals for tool support for capturing, formalizing and accessing rationale [DMMP06].

To ensure that AGREEMENT at the end fulfills the given requirements, it should be evaluated frequently during the development. But this poses a big issue as we are not sure how to acquire testing grounds that produce scientifically valuable results. Case studies seem in theory to be the best way. But to find appropriate participants and setups is very complex. Another idea is to conduct surveys, which introduce AGREEMENT and then retrieve opinions of practitioners.

In summary one can say, that first promising steps towards an agile RM approach are made with AGREEMENT. Two case studies showed anecdotically, that agile projects benefit from RM integration. But they also identified factors like usability, adaptability and seamless integration, that have to be present in order to be successful in the long run.

## References

- [Atl13] Atlassian JIRA, <https://www.atlassian.com/software/jira>, 2013, received at 07.10.2013.
- [BD09] Bernd Bruegge and Allen H Dutoit. *Object Oriented Software Engineering: Using UML, Patterns, and Java: International Version*. Prentice Hall, Boston, 3rd revised edition, August 2009.
- [DMMP06] Allen H Dutoit, Raymond McCall, Ivan Mistrík, and Barbara Paech, editors. *Rationale Management in Software Engineering*. Springer-Verlag, Berlin, February 2006.
- [Hal07] William E Halal. The Logic of Knowledge: KM Principles Support Agile Systems. In Kevin Desouza, editor, *Agile Information Systems: Conceptualization, Construction, and Management*, pages 31–40. Elsevier, Oxford, 2007.
- [KR78] Werner Kunz and Horst W. J. Rittel. Issues as elements of information systems, 1978.
- [MB09] John McAvoy and Tom Butler. The role of project management in ineffective decision making within Agile software development projects. *European Journal of Information Systems*, 18(4):372–383, 2009.
- [Rou07] William B Rouse. Agile Information Systems for Agile Decision Making. In Kevin Desouza, editor, *Agile Information Systems: Conceptualization, Construction, and Management*, pages 16–30. Elsevier, Oxford, 2007.

# Pattern-Based Detection and Utilization of Potential Parallelism in Software Systems

Christian Wulf

Department of Computer Science  
Kiel University  
D-24118 Kiel  
chw@informatik.uni-kiel.de

**Abstract:** Due to the paradigm shift from single-core to multi-core processors within the last ten years, software engineers not only need to have technical and domain-specific knowledge to add new features and solve any bugs. They also need to have knowledge about concurrency issues, e.g., to meet performance requirements.

Since introducing concurrency in existing software systems is often error-prone and difficult, we propose a semi-automatic, pattern-based approach to support the software engineer in the parallelization process and related concurrency tasks. We propose the use of patterns for both the detection of code regions with high potential of parallelism and for the corresponding parallel version utilizing information gathered by static and dynamic analysis. Besides describing the approach itself, we focus on our goals and research questions, and illustrate ideas on how to conduct a meaningful evaluation.

## 1 Introduction

Since processor performance cannot be improved anymore by increasing the clock frequency, many parallelization approaches have been proposed. For instance, parallel compilers [HAM<sup>+</sup>05, e.g.] or recommendation systems [MCGP07, e.g.] use the given structure of a software system either to detect parallelization potential or even to utilize such potential resulting in a parallel execution.

However, they often do not restructure the original source code by breaking dependencies to exploit further parallelization potential [URT11]. Moreover, fully automatic approaches need to over-approximate dependencies that are unknown or indeterminable at compile-time. Although semi-automatic approaches overcome this drawback with the help of dynamic analysis, all existing approaches require an parallelization expert instead of a general software engineer.

We present a semi-automatic parallelization approach for non-expert software engineers that provides solutions to the problems described above. Our approach allows to iteratively introduce parallelization by applying a pattern-matching restructur-

ing technique on the system dependency graph<sup>1</sup> of the given software system. In this paper, we focus on our goals, research questions, and the planned evaluation.

*Structure of this paper:* In Section 2, we describe the goals and research questions of our approach. Afterwards in Section 3, we present our approach. Finally, we present our planned evaluation in Section 4.

## 2 Goals and Research Questions

We envision a pattern-based, semi-automatic parallelization approach as solution to systematically guide and support the non-expert software engineer (in the following called *user*) in the parallelization process<sup>2</sup> without sacrificing flexibility and speedup for the sake of abstraction. This section provides an overview of the goals and research questions of the planned PhD thesis.

### **G1: Systematic Guidance and Support in the Parallelization Process**

We see a need for a systematic parallelization approach to guide and support the user in all the five phases in the parallelization process<sup>2</sup>: discovery, planning, transformation, code generation, and runtime management. *Q1: To what extent can we systematically guide and support the user in each of the five parallelization phases?*

**G2: Hide Concurrency-Specific Aspects from the User** Optimally, the approach should be executed automatically. If this is not possible, it should hide most of the concurrency-specific aspects from the user, e.g., the correct implementation of synchronization, to focus on the issues that are not automatically decidable.<sup>3</sup> *Q2: To what extent can we hide concurrency-specific aspects from the user?*

**G3: Structure- and Language Independence** Our approach should be able to parallelize any software system that can be represented as a system dependency graph, e.g., object-oriented software systems. In particular, it may not be tailored to one specific control or data structure, but should be open for all possible constructs. Furthermore, it should provide support for fine-grained as well as coarse-grained introduction of parallelism. *Q3: How to encapsulate structure- and language dependent information to provide a general parallelization concept?*

**G4: Extensibility** Our approach should be extensible to improve and enrich its parallelization phases with new insights from the research area. In particular, it should support adding new patterns at arbitrary levels of granularity without writing a single line of code. *Q4: How to achieve extensibility in each step?*

**G5: Parallelism** Finally, our approach should parallelize software systems to increase their performance. *Q5: To what extent can our approach parallelize software systems?*

---

<sup>1</sup>A system dependency graph represents a software system by nodes and edges where nodes are statements and edges are control or data dependencies between those statements.

<sup>2</sup>See [GJLT11] for the taxonomy of the five parallelization phases

<sup>3</sup>One example is when a code section is not parallelizable for all, but only for particular input values that are in fact guaranteed, but not directly encoded in the software system.

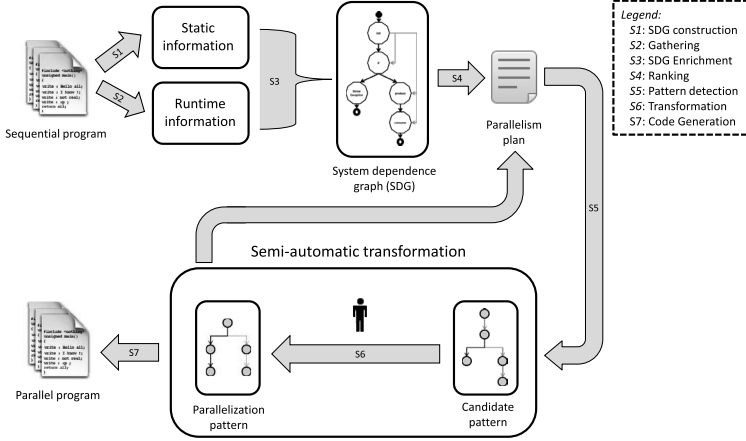


Figure 1: Overview of our semi-automatic parallelization approach

### 3 Approach

Our approach targets software engineers who need to parallelize existing software systems. It serves as guidance in the parallelization process and provides support for a pattern-based, iterative introduction of parallelism. Figure 1 gives an overview of the approach using seven steps to reveal and exploit parallelization potential.

The first three steps S1-S3 build a system dependency graph (SDG) representing the given software system using information gathered by static and dynamic analysis. It stores the control flow and data flow as well as further information about the structure and the runtime behavior.<sup>4</sup> In S4, a parallelism plan is constructed on the basis of the SDG. After construction, the plan consists of an ordered list of code sections that are most promising for a transformation to a parallel version. For example, assuming that long running methods have a higher parallelization potential, a simple plan would list all method declarations ordered by their average execution times.

The software engineer may then successively process the plan by executing the steps S5 and S6 on each code section. While S5 represents the pattern detection step to find code regions that have a high potential for parallelization, S6 constitutes the transformation from a matched instance of S5 to a semantically equivalent parallel version. For these two steps, we will provide an extensible pool of so-called candidate and corresponding parallelization patterns each represented as a dependency graph similar to the SDG. In this way, S5 and S6 can be executed automatically. However, before applying S6, the software engineer has the possibility to validate and adapt the proposed parallel version. The last step S7 is responsible for the code generation and can be executed after each iteration.

<sup>4</sup>For example, the type hierarchy and method execution times

Besides parallelizing loop iterations and array accesses, this approach also allows to parallelize, e.g., I/O accesses and to reveal further parallelization potential by restructuring and resolving dependencies with the help of runtime information.

## 4 Planned Evaluations

This section describes our planned evaluations for the goals mentioned in Section 2.

We evaluate **G1** and **G2** by implementing a prototype and conducting a questionnaire survey. Our prototype will contain patterns each encapsulates as much concurrency-related knowledge as possible. We then let two professional software engineers and 30 master students parallelize several example applications (including a financial risk assessment application of a German bank). Finally, the subjects fill in a questionnaire that consists of questions about the interaction with and usability of our prototype.

We evaluate **G3** by parallelizing loop control structures, method invocations, and I/O operations with our prototype. We also implement support for Java and C# to show that our approach is not targeted at one specific programming language.

We evaluate **G4** by providing our prototype with at least two different ranking strategies for S4. Moreover, we define several candidate patterns for S5 and corresponding parallelization patterns for S6 with different levels of granularity.

We evaluate **G5** by conducting a performance evaluation of our prototype. We use several input programs from different application domains for which a manually parallelized and an unparallelized version exist. We then execute our prototype for each of the unparallelized version and measure their resulting speedups. Afterwards, we compare our performance results with those of the parallelized versions.

## References

- [GJLT11] S. Garcia, D. Jeon, C.M. Louie, and M.B. Taylor. Kremlin: Rethinking and Rebooting gprof for the Multicore Age. In *Proc. of the 32nd ACM SIGPLAN Conference on Programming Lang. Design and Impl.*, 2011.
- [HAM<sup>+</sup>05] Mary W. Hall, Saman P. Amarasinghe, Brian R. Murphy, Shih-Wei Liao, and Monica S. Lam. Interprocedural Parallelization Analysis in SUIF. *ACM Trans. Program. Lang. Syst.*, 27, 2005.
- [MCGP07] T. Moseley, D.A. Connors, D. Grunwald, and R. Peri. Identifying Potential Parallelism via Loop-Centric Profiling. In *Proc. of the 4th Int. Conf. on Comp. Frontiers*, CF '07, pages 143–152. ACM, 2007.
- [URT11] A. Udupa, K. Rajan, and W. Thies. ALTER: Exploiting Breakable Dependences for Parallelization. *SIGPLAN Notices*, 46, 2011.

# Synchronizing Heterogeneous Models in a View-Centric Engineering Approach

Max E. Kramer  
Karlsruhe Institute of Technology  
max.e.kramer@kit.edu

## Abstract:

When software systems are modelled from different viewpoints using different notations, it is necessary to synchronize these heterogeneous models in order to sustain consistency. To realize this for a specific system, developers need two competences: They have to express the conceptual relationships between the elements of different modelling languages and domains. But they also have to master various techniques of Model-Driven Engineering (MDE), such as transformation languages. Current synchronization approaches, however, do not address these requirements separately and mix technical and conceptual challenges. To ease heterogeneous modelling, we propose a view-centric engineering approach, in which incremental synchronization transformations are generated from abstract synchronization specifications. This will make it possible to declare mappings and invariants for model concepts with a domain-specific language, for which developers can reuse and customize technical solutions of a generator. In this paper, we introduce the according research goals and questions and sketch our plans for realization and evaluation.

## 1 Introduction and Motivation

Many software systems are modelled from different viewpoints using different modelling languages in order to describe and analyse the systems appropriately for specific tasks. On the one hand, these different models may conform to different meta-models and on the other hand, they may describe identical parts of a system. A component-based system, for example, may be modelled in an abstract way with an architectural description language, while the structure and behaviour of individual components may be modelled with UML class and sequence diagrams. Because of the semantic overlap between these models, it is necessary to keep them consistent.

Current approaches for the synchronization of heterogeneous models, however, mix the conceptual challenge of identifying and expressing relationships between domain elements with technical challenges of transformation techniques. In this paper, we present our plans for a synchronization language for a view-centric engineering approach that separates technical solutions from conceptual synchronization specifications. It uses declarative mappings between metaclasses, normative invariants and imperative transformation customization code. From these three language parts, incremental synchronization transformations are derived using a customizable generator. Moreover, synchronization specifications may be used for the definition of views that integrate information from multiple metamodels and vice-versa.



## 2 Background and Related Work

In Model-Driven Engineering, models have to conform to a metamodel in order to be processed in transformations. This syntactic constraint makes it possible to derive source code, documentation, and other artefacts from appropriate models.

To cope with different views and notations, various model synchronization approaches have been presented. The multi-view point approach [RJV09], for example, defines direct correspondences between views. For such synthetic approaches, the link complexity grows exponentially with the number of views. In contrast to this, projective approaches like Orthographic Software Modelling [ASB10] synchronize views with a central model. Such a central model limits the expressive power and has to be designed upfront with all possible views, which makes evolution and extensions difficult and hinders support for legacy metamodels and views. Tool integration approaches like ModelBus [HRW09] provide advanced features like model merging but lack concepts for expressing the semantic relations between metamodels. Triple-Graph Grammars (TGGs) have been successfully used for model synchronization, for example, for SysML and AUTOSAR [GHN10]. The morphisms that map an interface part of a TGG to the left and right hand side are similar to our mappings. But TGGs do not separate technical transformation details from domain concepts.

## 3 Goals and Questions

Our plans for model synchronization are a part of the view-centric VITRUVIUS approach [KBL13]. They are led by the following two research goals: *G1: Ease the synchronization of heterogeneous models in a way that sustains consistency between those models. G2: Support the definition of integrated views on heterogeneous models through the use of synchronization information and vice-versa.*

We pursue these goals by answering the following two research questions: *Q1: Can we synchronize heterogeneous models automatically using transformations that are generated from abstract synchronization specifications? Q2: Can we couple view type definitions and model synchronization specifications?* Each of these questions can be divided into three subquestions: *Q1.1: Can we specify synchronization with a precise and expressive domain-specific language based on declarative mappings, normative invariants and custom response transformation snippets? Q1.2: Can we generate synchronization transformations from these mappings and embed the response snippets? Q1.3: Can we synchronize models by triggering these transformations after atomic changes and after invariant violations? Q2.1: Can we derive synchronization specifications from view type specifications and vice-versa? Q2.2: Can we restrict editability in view type specifications according to synchronization specifications? Q2.3: Can we derive synchronization requirements from view type specifications?*

## 4 Approach

We will answer our research questions by developing and evaluating a Domain-Specific Language (DSL) for synchronization specifications within the VITRUVIUS approach [KBL13]. A generator for this DSL will produce incremental transforma-

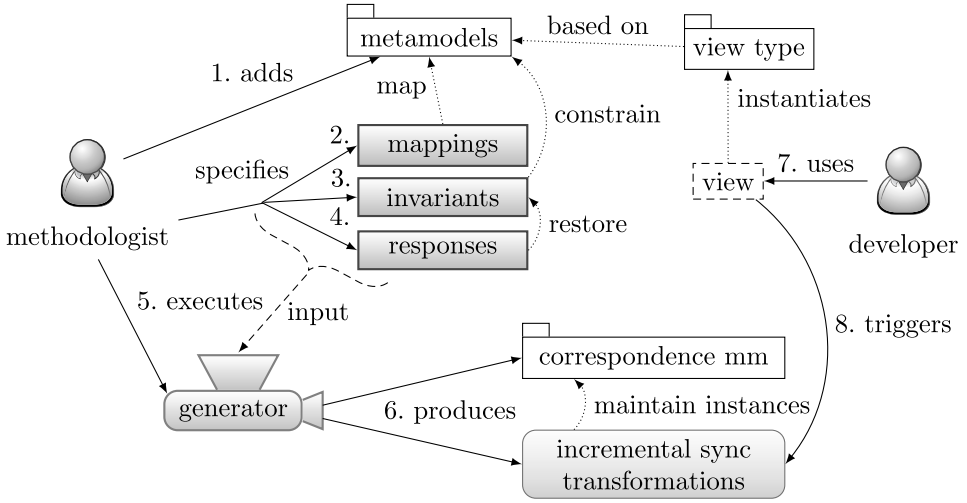


Figure 1: Process for defining and executing synchronizations in view-centric engineering

tions from specifications. Before we explain the language and its use, we sketch the overall approach and mention an important assumption: The approach combines all information of a software system in a virtual single underlying model that can be accessed solely by views conforming to view types. All views have to report sequences of atomic changes if changes that shall be persisted and propagated occurred in a view. The incremental transformations that we generate from the DSL are triggered using these atomic changes. In general, they cannot be derived unambiguously from model differences but have to be recorded using customized listeners, which may be based on refactoring commands for textual languages. The virtual model is a modular combination of individual models conforming to different metamodels. This decoupled layout makes it possible to integrate legacy metamodels and allows for independent evolution. The definition of view types and the integration of the modular metamodel are carried out by a special role called *methodologist*.

The synchronization language consists of three parts for *specifying*, *checking* and *preserving* consistency. In the first part, declarative *mappings* specify semantic correspondences between metaclasses, their attributes, and references using conditions that are based on attribute and reference values. In the second part, consistency checks within and between models may be defined with normative *invariants* that are formulated using the Object Constraint Language (OCL). Invariants may be specified for individual metamodels or combinations thereof and may expose parameters that can be used by the third language part. In this part *responses* to specific modifications or invariant violations can be encoded using a general-purpose model transformation language (QVT-O or ATL). Thus the power and expressivity of a general-purpose language can be used if the language constructs for synchronization mappings are not sufficient, for example, for clean-up actions or conflict resolution.

The detailed process for defining and executing synchronization behaviour using the language and generator is shown in Figure 1. The methodologist, who is responsible for the virtual metamodel, view types, and synchronization, first adds metamod-els to the virtual metamodel. Then, he specifies mappings between metamod-els, invariants that constrain these metamod-els, and optional responses that may re-store these invariants. Afterwards, the methodologist executes a generator, which produces a correspondence metamodel and incremental sync transformations from the three specification parts. For every mapped metaclass and every modification type a separate transformation, which embeds response transformation snippets, is produced. These transformations are triggered if a developer changes instances of the corresponding metaclasses in a view conforming to a view-type.

## 5 Evaluation

We will evaluate our synchronization language and generator in a case study that combines a component-based Architectural Description Language (ADL), UML class diagrams and Java source code. With this case study, we evaluate whether it is possible to synchronize architectural models, class diagrams and source code with our approach. In order to assess the benefits of our approach with respect to *G1*, we are planning to conduct a quasi-experiment with graduate students and engineers. In it, subjects will design and implement component-based software and (a) keep all artefacts manually in sync, (b) use a general-purpose transformation language or (c) our synchronization language and generator. Consistency and effectivity will be measured, but because of the duration we cannot control all influencing variables.

## References

- [ASB10] C. Atkinson, D. Stoll, and P. Bostan. “Orthographic Software Modeling: A Practical Approach to View-Based Development”. In: *Evaluation of Novel Approaches to Software Engineering*. Vol. 69. Communications in Computer and Information Science. Springer Berlin / Heidelberg, 2010, pp. 206–219.
- [GHN10] H. Giese, S. Hildebrandt, and S. Neumann. “Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent”. In: *Graph Transformations and Model-Driven Engineering*. Vol. 5765. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, pp. 555–579.
- [HRW09] C. Hein, T. Ritter, and M. Wagner. “Model-Driven Tool Integration with ModelBus”. In: *Workshop Future Trends of Model-Driven Development*. 2009.
- [KBL13] M. E. Kramer, E. Burger, and M. Langhammer. “View-centric engineering with synchronized heterogeneous models”. In: *Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*. VAO ’13. Montpellier, France: ACM, 2013, 5:1–5:6.
- [RJV09] J. R. Romero, J. I. Jaén, and A. Vallecillo. “Realizing Correspondences in Multi-viewpoint Specifications”. In: *Enterprise Distributed Object Computing Conference, 2009. EDOC ’09. IEEE International*. 2009, pp. 163–172.

# Summarizing, Classifying and Diversifying User Feedback

Emitza Guzman  
Technische Universität München  
Department of Applied Software Engineering  
Boltzmannstr. 3, 85748 Garching  
emitza.guzman@mytum.de

**Abstract:** Current software users can give feedback about a software application through diverse online mediums such as blogs, forums, instant messaging and product review sites. This produces a large amount of textual information, making it difficult to take user feedback into consideration in the software evolution process. We propose to automatically summarize, classify and diversify textual user feedback in order to reduce information overload and give users a stronger voice in software evolution. In this paper we describe the research questions, the envisioned approach and the achieved progress related to the presented thesis.

## 1 Introduction

In the early days of the digital age, software users were a few engineers or scientists with specific technical requirements. However, with the evolution of computing power, the appearance of more affordable personal computers, the Internet and smartphones, the definition of user has extended to include more heterogeneous groups of people with a wide variety of needs and expectations [Pag13]. Current software engineering research has pointed out the importance of taking these needs and expectations into account in order to make useful and usable software, and reduce maintenance efforts [Kuj03, KKLK05, VMSC02]. In order to keep software useful and relevant through its evolution it is necessary that user needs and expectations are considered in the post-deployment phase [BD04, KLF<sup>+</sup>11]. With the growing trend of Internet use, more users are writing feedback about software applications, requesting features and reporting bugs through social media, online forums, specialized user feedback platforms or directly in the application distribution platforms (e.g. GooglePlay<sup>1</sup> or the AppStore<sup>2</sup>) by means of their integrated review system. This tendency produces a large amount of textual data which can be burdensome to manually analyze and process. Because of this, it is difficult for developers to remain aware of feature requests and bug reports that users write, as well as the general opinion that users have about an application and its features. This thesis researches possible ways to overcome this problem. More concretely, it analyzes summarization, classification and diversification techniques commonly used in the natural processing language and information retrieval communities and proposes to apply them to feedback given by users of software

---

<sup>1</sup><https://play.google.com>

<sup>2</sup><http://www.apple.com/iphone-5s/app-store/>

applications. The summarization of user feedback will allow analysts and developers to have a quicker overview of the aggregated user feedback content, whereas classification will separate user feedback into categories allowing developers and analysts to concentrate on the categories that are relevant to the evolution task they are performing. Furthermore, diversification will retrieve user feedback that varies in its content, allowing developers to obtain user feedback with conflicting opinions and with a varied set of topics.

## 2 Research Questions

This thesis aims to reduce developers' and analysts' information overload when analyzing and processing user feedback by automatically summarizing, classifying and diversifying it. The research questions that guide this work are the following:

### Summarization

**RQ1:** How can user feedback be automatically summarized?

**RQ2:** Do analysts and developers benefit from summarized user feedback?

### Classification

**RQ3:** What are the characteristics of the different types of user feedback (e.g. feature requests, bug reports, reviews about existing features)? What are the characteristics of useful feedback in the perspective of developers and analysts?

**RQ4:** How can the different types of feedback be automatically classified? How can useful feedback be automatically classified?

### Diversification

**RQ5:** How can we retrieve diverse feedback in terms of the mentioned software features and the sentiments associated to them?

**RQ6:** What are the benefits of retrieving diverse feedback in the software evolution process?

## 3 Envisioned Approach

We will apply the approach to feedback given by users through application distribution platforms, such as GooglePlay and the AppStore, and through product review platforms, such as Epinions<sup>3</sup> and Ohloh<sup>4</sup>. The following paragraphs describe the approach that will be followed to answer the research questions mentioned in Section 2.

**Summarization** We plan to analyze different summarization techniques previously used to summarize content written in social media and product reviews [HL04, MLW<sup>+</sup>07, PE05].

---

<sup>3</sup><http://www.epinions.com>

<sup>4</sup><http://www.ohloh.net/>

We will focus on summarizing software features and the sentiments associated to these features. For this initial step we will apply topic modeling algorithms, frequent item set mining and sentiment analysis. We will evaluate our approach against manually generated summaries. Additionally, we will do a controlled experiment to measure the effort differences when developers and analysts deal with summarized and non-summarized user feedback.

**Classification** We will apply content analysis to a random sample of user feedback in order to find the characteristics that define feature requests, bug reports and reviews about existing software features. Furthermore, we will also use content analysis to define the characteristics of useful and useless feedback in the developers' and analysts' perspective. We will combine these results with data mining algorithms for the automatic classification of feedback. To evaluate the approach we will compare our results against manually labeled user feedback.

**Diversification** We will focus on retrieving feedback that mentions different sets of software features and sentiments. Our assumption is that retrieving feedback which mentions a wide spectrum of software features and sentiments can benefit the decision process of developers and analysts in software evolution concerning which features to implement, remove or improve. For extracting diverse user feedback we will use information retrieval algorithms employed to extract diverse product reviews [KD11, TNT11] and adapt them to the peculiarities of software feedback if needed. We will use metrics for diversification commonly used in information retrieval [CKC<sup>+</sup>08] for the evaluation of our approach.

## 4 Current State

We have designed and implemented an initial approach to summarize the content and sentiments present in user feedback [GB13] and are currently evaluating the approach on user reviews from the AppStore and GooglePlay. The approach summarizes the content present in user feedback as sets of co-occurring words and assigns each feedback a quantitative value which expresses the sentiment present in the feedback. It uses a topic modeling algorithm [BNJ03] for content summarization and lexical sentiment analysis [TBP<sup>+</sup>10] for extracting sentiments from text. We plan to improve the current approach by including other summarization techniques, and by evaluating more extensively against manually generated summaries and, as mentioned in Section 3, by measuring the effect that the summaries have on developers' and analysts' effort through controlled experiments.

## References

- [BD04] Bernd Bruegge and Allen H Dutoit. *Object-Oriented Software Engineering Using UML, Patterns and Java-(Required)*. Prentice Hall, 2004.
- [BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

- [CKC<sup>+</sup>08] Charles L A Clarke, Maheedhar Kolla, Gordon V Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st International Conference on Research and Development in Information Retrieval*, pages 659–666. ACM, 2008.
- [GB13] Emitza Guzman and Bernd Bruegge. Towards Emotional Awareness in Software Development Teams. In *Symposium on the Foundations of Software Engineering - FSE '13*, 2013.
- [HL04] Mingqing Hu and Bing Liu. Mining opinion features in customer reviews. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining - KDD '04*, pages 755–760. AAAI Press, 2004.
- [KD11] Ralf Krestel and Nima Dokoohaki. Diversifying product review rankings: Getting the full picture. In *Web Intelligence and Intelligent Agent Technology - WI-IAT '11*, volume 1, pages 138–145. IEEE, 2011.
- [KKLK05] Sari Kujala, Marjo Kauppinen, Laura Lehtola, and Tero Kojo. The role of user involvement in requirements quality and project success. In *Proceedings. 13th International Conference on Requirements Engineering, 2005.*, pages 75–84. IEEE, 2005.
- [KLF<sup>+</sup>11] Andrew J Ko, Michael J Lee, Valentina Ferrari, Steven Ip, and Charlie Tran. A case study of post-deployment user feedback triage. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 1–8. ACM, 2011.
- [Kuj03] Sari Kujala. User involvement: a review of the benefits and challenges. *Behaviour & information technology*, 22(1):1–16, 2003.
- [MLW<sup>+</sup>07] Qiaozhu Mei, Xu Ling, Matthew Wondra, Hang Su, and ChengXiang Zhai. Topic sentiment mixture: modeling facets and opinions in Weblogs. In *Proc. of the 16th international conference on World Wide Web - WWW '07*, pages 171–180, 2007.
- [Pag13] Dennis Pagano. *Portneuf - A Framework for Continuous User Involvement*. Doctoral thesis, Technische Universität München, 2013.
- [PE05] Ana-Maria Popescu and Oren Etzioni. Extracting product features and opinions from reviews. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing - HLT '05*, pages 339–346. Association for Computational Linguistics, 2005.
- [TBP<sup>+</sup>10] Mike Thelwall, Kevan Buckley, Georgios Paltoglou, Di Cai, and Arvid Kappas. Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*, 61(12):2544–2558, 2010.
- [TNT11] Panayiotis Tsaparas, Alexandros Ntoulas, and Evimaria Terzi. Selecting a comprehensive set of reviews. In *Proceedings of the 17th International Conference on Knowledge Discovery and Data Mining*, pages 168–176. ACM, 2011.
- [VMSC02] Karel Vredenburg, Ji-Ye Mao, Paul W Smith, and Tom Carey. A survey of user-centered design practice. In *Proceedings of the Conference on Human factors in Computing Systems: Changing our world, changing ourselves*, pages 471–478. ACM, 2002.





## *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 Language to Information Systems, NLDB'2001.
- P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensor-gestützte 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 Magenheimer (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 und 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 Magenheimer, 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 – Erfahrungen 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 Machle, 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.): Persistence, 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, Wolfgang 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 Ad-hoc-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 informatische 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 its 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 Kowalczyk, 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. Morgens-tern, K. Luzzi, 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 Röbling, 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 Magenheimer, 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 1<sup>st</sup> 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 3<sup>rd</sup> 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 1<sup>st</sup> International Working Conference on Business Process and Services Computing BPSC 2007
- P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and management The 4<sup>th</sup> 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.)  
9<sup>th</sup> Workshop on Parallel Systems and Algorithms (PASA)  
Workshop of the GI/ITG Special 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.)  
10<sup>th</sup> 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 Dreö Rodosek (Hrsg.)  
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung
- P-131 Robert Krimmer, Rüdiger Grimm (Eds.)  
3<sup>rd</sup> International Conference on Electronic Voting 2008  
Co-organized by Council of Europe, Gesellschaft für Informatik und 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 Anwendungsentwicklung 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 Informationssysteme (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.)  
9<sup>th</sup> International Conference on Innovative Internet Community Systems  
I<sup>2</sup>CS 2009
- P-149 Paul Müller, Bernhard Neumair, Gabi Dreö 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 3<sup>rd</sup> 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 Brömmе, 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 Rannenberг (Hrsg.)  
Mobile und Ubiquitäre  
Informationssysteme  
Technologien, Anwendungen und  
Dienste zur Unterstützung von mobiler  
Kollaboration
- P-164 Arslan Brömmе, 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.)  
10<sup>th</sup> International Conference on Innovative Internet Community Systems (I<sup>2</sup>CS) – Jubilee Edition 2010 –
- P-166 Paul Müller, Bernhard Neumair, Gabi Dreö Rodosek (Hrsg.)  
3. DFN-Forum Kommunikationstechnologien  
Beiträge der Fachtagung
- P-167 Robert Krimmer, Rüdiger Grimm (Eds.)  
4<sup>th</sup> 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 Juhirsch (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 Brömmel, 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 Fährnrich, Bogdan Franczyk (Hrsg.)  
INFORMATIK 2010  
Service Science – Neue Perspektiven für die Informatik  
Band 1
- P-176 Klaus-Peter Fährnrich, Bogdan Franczyk (Hrsg.)  
INFORMATIK 2010  
Service Science – Neue Perspektiven für die Informatik  
Band 2
- P-177 Witold Abramowicz, Rainer Alt, Klaus-Peter Fährnrich, 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.)  
6<sup>th</sup> 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.)  
11<sup>th</sup> International Conference on Innovative Internet Community Systems (I<sup>2</sup>CS)
- P-187 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)  
4. DFN-Forum Kommunikationstechnologien, 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 Landwirtschaft  
Fokus Forstwirtschaft
- P-195 Neeraj Suri, Michael Waidner (Hrsg.)  
Sicherheit 2012  
Sicherheit, Schutz und Zuverlässigkeit  
Beiträge der 6. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
- P-196 Arslan Brömme, Christoph Busch (Eds.)  
BIOSIG 2012  
Proceedings of the 11<sup>th</sup> International Conference of the Biometrics Special Interest Group
- P-197 Jörn von Lucke, Christian P. Geiger, Siegfried Kaiser, Erich Schweighofer, Maria A. Wimmer (Hrsg.)  
Auf dem Weg zu einer offenen, smarten und vernetzten Verwaltungskultur  
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2012
- P-198 Stefan Jähnichen, Axel Küpper, Sahin Albayrak (Hrsg.)  
Software Engineering 2012  
Fachtagung des GI-Fachbereichs Softwaretechnik
- P-199 Stefan Jähnichen, Bernhard Rumpe, Holger Schlingloff (Hrsg.)  
Software Engineering 2012  
Workshopband
- P-200 Gero Mühl, Jan Richling, Andreas Herkersdorf (Hrsg.)  
ARCS 2012 Workshops
- P-201 Elmar J. Sinz Andy Schürr (Hrsg.)  
Modellierung 2012
- P-202 Andrea Back, Markus Bick, Martin Breunig, Key Pousttchi, Frédéric Thiesse (Hrsg.)  
MMS 2012: Mobile und Ubiquitäre Informationssysteme
- P-203 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)  
5. DFN-Forum Kommunikationstechnologien  
Beiträge der Fachtagung
- P-204 Gerald Eichler, Leendert W. M. Wienhofen, Anders Kofod-Petersen, Herwig Unger (Eds.)  
12<sup>th</sup> International Conference on Innovative Internet Community Systems (I2CS 2012)
- P-205 Manuel J. Kripp, Melanie Volkamer, Rüdiger Grimm (Eds.)  
5<sup>th</sup> International Conference on Electronic Voting 2012 (EVOTE2012)  
Co-organized by the Council of Europe, Gesellschaft für Informatik und E-Voting.CC
- P-206 Stefanie Rinderle-Ma, Mathias Weske (Hrsg.)  
EMISA 2012  
Der Mensch im Zentrum der Modellierung
- P-207 Jörg Desel, Jörg M. Haake, Christian Spannagel (Hrsg.)  
DeLFI 2012: Die 10. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V.  
24.–26. September 2012



- P-208 Ursula Goltz, Marcus Magnor, Hans-Jürgen Appelrath, Herbert Matthies, Wolf-Tilo Balke, Lars Wolf (Hrsg.)  
INFORMATIK 2012
- P-209 Hans Brandt-Pook, André Fleer, Thorsten Spitta, Malte Wattenberg (Hrsg.)  
Nachhaltiges Software Management
- P-210 Erhard Plödereder, Peter Dencker, Herbert Klenk, Hubert B. Keller, Silke Spitzer (Hrsg.)  
Automotive – Safety & Security 2012  
Sicherheit und Zuverlässigkeit für automobile Informationstechnik
- P-211 M. Clasen, K. C. Kersebaum, A. Meyer-Aurich, B. Theuvsen (Hrsg.)  
Massendatenmanagement in der Agrar- und Ernährungswirtschaft  
Erhebung - Verarbeitung - Nutzung  
Referate der 33. GIL-Jahrestagung  
20. – 21. Februar 2013, Potsdam
- P-212 Arslan Brömme, Christoph Busch (Eds.)  
BIOSIG 2013  
Proceedings of the 12th International Conference of the Biometrics Special Interest Group  
04.–06. September 2013  
Darmstadt, Germany
- P-213 Stefan Kowalewski, Bernhard Rump (Hrsg.)  
Software Engineering 2013  
Fachtagung des GI-Fachbereichs Softwaretechnik
- P-214 Volker Markl, Gunter Saake, Kai-Uwe Sattler, Gregor Hackenbroich, Bernhard Mitschang, Theo Härder, Veit Köppen (Hrsg.)  
Datenbanksysteme für Business, Technologie und Web (BTW) 2013  
13. – 15. März 2013, Magdeburg
- P-215 Stefan Wagner, Horst Lichter (Hrsg.)  
Software Engineering 2013  
Workshopband  
(inkl. Doktorandensymposium)  
26. Februar – 1. März 2013, Aachen
- P-216 Gunter Saake, Andreas Henrich, Wolfgang Lehner, Thomas Neumann, Veit Köppen (Hrsg.)  
Datenbanksysteme für Business, Technologie und Web (BTW) 2013 – Workshopband  
11. – 12. März 2013, Magdeburg
- P-217 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreö Rodosek (Hrsg.)  
6. DFN-Forum Kommunikations-technologien  
Beiträge der Fachtagung  
03.–04. Juni 2013, Erlangen
- P-218 Andreas Breiter, Christoph Rensing (Hrsg.)  
DeLFI 2013: Die 11 e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. (GI)  
8. – 11. September 2013, Bremen
- P-219 Norbert Breier, Peer Stechert, Thomas Wilke (Hrsg.)  
Informatik erweitert Horizonte  
INFOS 2013  
15. GI-Fachtagung Informatik und Schule  
26. – 28. September 2013
- P-220 Matthias Horbach (Hrsg.)  
INFORMATIK 2013  
Informatik angepasst an Mensch, Organisation und Umwelt  
16. – 20. September 2013, Koblenz
- P-221 Maria A. Wimmer, Marijn Janssen, Ann Macintosh, Hans Jochen Scholl, Efthimios Tambouris (Eds.)  
Electronic Government and Electronic Participation  
Joint Proceedings of Ongoing Research of IFIP EGOV and IFIP ePart 2013  
16. – 19. September 2013, Koblenz
- P-222 Reinhard Jung, Manfred Reichert (Eds.)  
Enterprise Modelling and Information Systems Architectures (EMISA 2013)  
St. Gallen, Switzerland  
September 5. – 6. 2013
- P-223 Detlef Hühnlein, Heiko Roßnagel (Hrsg.)  
Open Identity Summit 2013  
10. – 11. September 2013  
Kloster Banz, Germany
- P-224 Eckhart Hanser, Martin Mikusz, Masud Fazal-Baqaie (Hrsg.)  
Vorgehensmodelle 2013  
Vorgehensmodelle – Anspruch und Wirklichkeit  
20. Tagung der Fachgruppe Vorgehensmodelle im Fachgebiet Wirtschaftsinformatik (WI-VM) der Gesellschaft für Informatik e.V.  
Lörrach, 2013
- P-227 Wilhelm Hasselbring, Nils Christian Ehmke (Hrsg.)  
Software Engineering 2014  
Fachtagung des GI-Fachbereichs Softwaretechnik  
25. Februar – 28. Februar 2014  
Kiel, Deutschland

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](mailto:druckverlag@koellen.de)