Gesellschaft für Informatik e.V. (GI)

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

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

The volumes are published in German or English.

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

ISSN 1617-5468
ISBN 978-3-88579-646-6

This volume contains the contributions of the Software Engineering (SE) 2016 conference held from 23.02.2016 - 26.02.2016 in Vienna, Austria.

The SE proceedings contain extended abstracts from the scientific program, the technology transfer program, and the workshop program.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016

252

GI-Edition

Lecture Notes in Informatics

Jens Knoop, Uwe Zdun (Hrsg.)

Software Engineering 2016

Fachtagung des GI-Fachbereichs Softwaretechnik

23.–26. Februar 2016
Wien

Proceedings

Jens Knoop, Uwe Zdun (Hrsg.)

Software Engineering 2016

23.–26. Februar 2016
Wien, Österreich

Gesellschaft für Informatik e.V. (GI)

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

Volume P-252

ISBN 978-3-88579-646-6
ISSN 1617-5468

Volume Editors
Univ.-Prof. Dr. Jens Knoop
Technische Universität Wien
Argentinierstraße 8, 1040 Wien, Österreich
knoop@complang.tuwien.ac.at

Univ.-Prof. Dr. Uwe Zdun
Universität Wien
Währinger Straße 29, 1090 Wien, Österreich
uwe.zdun@univie.ac.at

printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Die Software Engineering-Konferenz findet 2016 in Wien statt. Angelehnt an das aktuelle Motto der Stadt Wien "SMART CITY WIEN" – so bezeichnen wir unsere Stadt, die den Herausforderungen des 21. Jahrhunderts begegnet", lautet das Motto für diese Konferenz: Software Engineering für Smart Cities.

Software Engineering ist eine praxisorientierte Wissenschaftsdisziplin, deren Ergebnisse in die Praxis der Softwareentwicklung einfließen sollten. Gleichzeitig geben relevante Fragen aus der Praxis immer wieder den Anstoß für innovative Forschungsprojekte. Zum Austausch zwischen den Wissenschaftlern und Praktikern im Bereich des Software Engineering, bietet die Software Engineering 2016 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.

Die Konferenzserie SE ist die deutschsprachige Konferenz zum Thema Software Engineering des Fachbereichs Softwaretechnik der Gesellschaft für Informatik e. V. (GI). Die Organisatoren der SE 2016 von der Technischen Universität Wien und der Universität Wien, sowie die Österreichische Computer Gesellschaft (OCG), laden Sie herzlich nach Wien ein.

Im wissenschaftlichen Programm setzt die SE 2016 das erfolgreiche Format der letzten Jahre fort. Alle Vorträge stellen hochkarätige Forschungsbeiträge vor, die in den vergangenen zwei Jahren auf internationalen Spitzenkonferenzen oder in führenden Fachzeitschriften der Softwaretechnik veröffentlicht wurden.

Das Ziel des wissenschaftlichen Programms ist die Stimulation des wissenschaftlichen Diskurses innerhalb der deutschsprachigen Software Engineering Community sowie die Erhöhung des "Impacts" bereits veröffentlichter Ergebnisse.

Alle Einreichungen wurden durch das Programmkomitee ausgewählt und von mindestens 3 Gutachtern begutachtet. Für jeden akzeptierten Beitrag finden Sie eine Kurzfassung im Umfang von 2 Seiten in diesem Tagungsband, ebenfalls Zusammenfassungen der beiden eingeladenen Hauptvorträge von Prof. Uwe Aßmann, TU Dresden, und Prof. Wilhelm Hasselbring, CAU Kiel.

Weiters finden Sie in diesem Tagungsband Zusammenfassungen der akzeptierten Workshops, sowie des Technologietransferprogramms.

Wir freuen uns, Sie in Wien begrüßen zu dürfen und wünschen Ihnen eine spannende Software Engineering 2016 Tagung!

Wien, im Februar 2016

Jens Knoop, Konferenzvorsitzender

Uwe Zdun, Programmkomiteevorsitzender

Organisationskomitee

Konferenzvorsitzender:	Jens Knoop, TU Wien
Leitung des Programmkomitees:	Uwe Zdun, Universität Wien
Workshop-Vorsitzender:	Wolf Zimmermann, Universität Halle-Wittenberg
Technologietransfer:	Michael Felderer, Universität Innsbruck
	Wilhelm Hasselbring, Universität Kiel

Programmkomitee wissenschaftliches Programm

Uwe Zdun	Universität Wien
Uwe Assmann	Universität Dresden
Gregor Engels	Universität Paderborn
Michael Goedicke	Universität Duisburg-Essen
Wilhelm Hasselbring	Universität Kiel
Maritta Heisel	Universität Duisburg-Essen
Jens Knoop	Technische Universität Wien
Florian Matthes	TU München
Klaus Pohl	Universität Duisburg-Essen
Ralf Reussner	KIT/FZI, Karlsruhe
Dirk Riehle	Friedrich-Alexander-Universität Erlangen-Nürnberg

Workshop-Komitee

Wolf Zimmermann	Martin-Luther-Universität Halle-Wittenberg
Anne Koziolek	Karlsruhe Institute of Technology
Christian Panis	Catena, NL
Sibylle Schupp	Technische Universität Hamburg-Harburg

Inhaltsverzeichnis

Testing

Software Construction 1

Performance Modelling and Analysis 1

Requirements Engineering

Software Construction 2

Performance Modelling and Analysis 2

Empirical Software Engineering 1

Software Construction 3

Empirical Software Engineering 2

Business Process Engineering

Product Lines

Empirical Software Engineering 3

Modelling and Model-Driven Development

Variability and Evolution 1

Software Verification

Variability and Evolution 2

Keynotes

Workshops

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 15

Guiding Random Test Generation with Program Analysis

Lei Ma[1], Cyrille Artho[2], Cheng Zhang[3], Hiroyuki Sato[4], Johannes Gmeiner[5] and Rudolf Ramler[6]

Abstract: Random test generation is effective in creating method sequences for exercising the software under test. However, black-box approaches for random testing are known to suffer from low code coverage and limited defect detection ability. Analyzing the software under test and using the extracted knowledge to guide test generation can help to overcome these limitations. We developed a random test case generator augmented by a combination of six static and dynamic program analysis techniques. Our tool GRT (Guided Random Testing) has been evaluated on real-world software systems as well as Defects4J benchmarks. It outperformed related approaches in terms of code coverage, mutation score and detected faults. The results show a considerable improvement potential of random test generation when combined with advanced analysis techniques.

Keywords: Random testing, program analysis, static and dynamic analysis.

Random approaches for testing object-oriented programs can effectively generate sequences of method calls to execute the objects of the system under test (SUT). The test data for input parameters are either constant values in case of primitive data types or objects retuned by already generated method sequences, which can be used as inputs for further test generation. The generation process incrementally builds more and longer test sequences by randomly selecting methods and reusing previously generated method sequences (that return objects) as input until a time limit is reached.

While being highly automated and easy to use, random testing may suffer from low code coverage and limited defect detection ability when applied to real-world applications. It is considered unlikely that random approaches are able to exercise all "deeper" features of a reasonably-sized program by mere chance. These limitations are due to the adoption of a black-box approach without using application-/implementation-specific knowledge. Mining and leveraging information about the SUT can provide a valuable aid to guide random testing and, thus, to overcome such limitations.

We developed an approach for random test generation, Guided Random Testing (GRT) [Ma15], which has been augmented by an ensemble of six static and dynamic program analysis techniques. They are used to extract and incorporate information on program

[1] University of Tokyo, Japan, malei@satolab.itc.u-tokyo.ac.jp
[2] National Institute of Advanced Industrial Science and Technology (AIST), Japan, c.artho@aist.go.jp
[3] University of Waterloo, Canada, c16zhang@uwaterloo.ca
[4] University of Tokyo, Japan, schuko@satolab.itc.u-tokyo.ac.jp
[5] Software Competence Center Hagenberg (SCCH), Austria, johannes.gmeiner@scch.at
[6] Software Competence Center Hagenberg (SCCH), Austria, rudolf.ramler@scch.at

types, data, and dependencies in the various stages of the test generation process. The overall effectiveness of GRT results not only from applying each of the individual techniques, but also from their combination and orchestration. Program information is extracted by some components at specific steps and passed to others to facilitate their tasks.

Fig. 1 shows the different techniques and how they are interacting. First, the SUT is statically analyzed. *Constant mining* extracts constant values to create a diverse yet application-specific input data set for test generation. The diversity is further increased by applying input fuzzing and by favoring methods that change the state of input objects as a side effect of their execution, which is determined by *Impurity analysis*. Information about dependencies between methods is used by the technique *Detective* for constructing method sequences returning input objects that are not in main object pool. *Elephant brain* manages all the objects stored in the main object pool including exact type information. Coverage information is recorded throughout test generation and is used by *Bloodhound* to select methods not well covered so far. *Orienteering* estimates the execution time of each method sequence to accelerate the overall generation process.

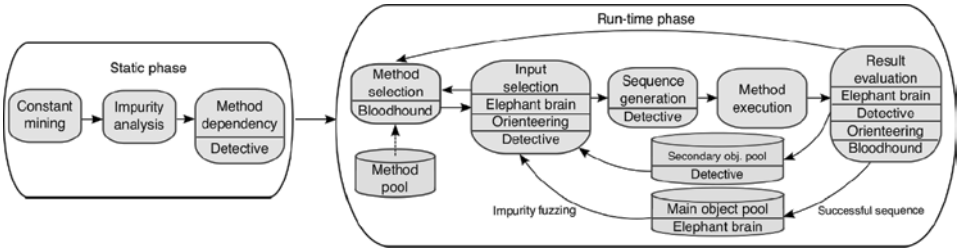


Fig. 1: Static and dynamic program analysis techniques included in GRT.

GRT has been evaluated on 32 real-world projects and, when compared to other tools (Randoop and EvoSuite), outperformed major peer techniques in terms of code coverage (by 13%) and mutation score (by 9 %). On the four studied benchmarks from Defects4J, which contain 224 real faults, GRT also showed better fault detection capability, finding 147 faults (66 %). Furthermore, in an in-depth evaluation on the latest versions of ten popular open source projects, GRT successfully detected over 20 previously unknown defects that were confirmed by the developers.

The results indicate that random testing has not yet reached its limits. There is still a considerable potential for further improving random test generation approaches by incorporating advanced analysis techniques – a path we plan to follow in our future work.

References

[Ma15] Ma, L.; Artho, C.; Zhang, C.; Sato, H.; Gmeiner, J.; Ramler, R.: GRT: Program-Analysis-Guided Random Testing. Proc. 30th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE 2015), Lincoln, Nebraska, USA, November 2015.

Just test what you cannot verify![1]

Mike Czech[2] Marie-Christine Jakobs[3] Heike Wehrheim[4]

Abstract: Software verification is an established method to ensure software safety. Nevertheless, verification still often fails, either because it consumes too much resources, e.g., time or memory, or the technique is not mature enough to verify the property. Often then discarding the *partial verification*, the validation process proceeds with techniques like *testing*.

To enable standard testing to profit from previous, partial verification, we use a summary of the verification effort to simplify the program for subsequent testing. Our techniques use this summary to construct a *residual program* which only contains program paths with unproven assertions. Afterwards, the residual program can be used with standard testing tools.

Our first experiments show that testing profits from the partial verification. The test effort is reduced and combined verification and testing is faster than a complete verification.

Keywords:combination verification and validation, conditional model checking, static analysis, testing, slicing

1 Overview

Although automatic software verification and its tool support evolved in recent years, software verification still fails. The verified property may be beyond the capabilities of a tool or its verification requires too many resources, e.g., time and memory. Thus, verification cannot be applied in an "on-the-fly" context in which validation should be carried out in a small amount of time and probably on a device with restricted resources. To still gain confidence in the software, after a failed verification, further validation techniques like testing are applied which often discard the previous, partial verification results.

Within the Collaborative Research Centre SFB 901 at the University of Paderborn we developed two orthogonal approaches to combine verification and testing [CJW15]. Our idea is to consider the partial verification during testing and only test paths which have not been fully verified. To use standard testing techniques we build a new program for testing, the *residual program*, which contains only the non-verified paths. Both approaches start with a verification tool that keeps track of its (abstract) state space exploration in terms of an abstract reachability graph (ARG). If the verification tool stops with an uncomplete verification, it generates a condition as proposed in conditional model checking [Be12].

[1] This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

[2] Universität Paderborn, Institut für Informatik, Warburger Str. 100, 33098, Paderborn, mczech@mail.upb.de

[3] Universität Paderborn, Institut für Informatik, Warburger Str. 100, 33098, Paderborn, marie.christine.jakobs@upb.de

[4] Universität Paderborn, Institut für Informatik, Warburger Str. 100, 33098, Paderborn, wehrheim@upb.de

This condition is related to the ARG and describes in a graph manner which program paths are proven correct and which remain. Next, our approaches use the condition to construct the residual program. Afterwards, the residual program is tested.

Our first approach computes its residual program via a product combination of the program and condition, excluding paths of the condition which are proven correct. Thus, due to e.g. loop unwindings during verification, the residual program's structure may differ from the original program. It is only a semantical subprogram.

Our second approach constructs a syntactical subprogram which contains all statements that influence the assertions which have not been fully verified. These assertions are all assertions on the unexplored paths in the condition and become the slicing criteria for dependence based slicing. At last, dependence based slicing builds the residual program.

We can easily combine our two approaches. First, we apply the product construction technique to construct an intermediate residual program. Second, the set of all assertions in the intermediate residual program becomes our slicing criterion. Finally, we slice the intermediate residual program to obtain the final residual program for testing.

In our experiments, we used the verification tool CPACHECKER [BK11] for partial verification, Frama-C [Cu12] for slicing and the concolic test tool KLEE [CDE08]. On our small benchmark suite, the combination of verification and testing was mostly faster than complete verification. Additionally, the two slicing based approaches reduced the test effort (number of tests and program size) but none always outperformed the other.

Our proposed combinations of verification and testing demonstrate that testing benefits from previous partial verification.

References

[Be12] Beyer, Dirk; Henzinger, Thomas A.; Keremoglu, M. Erkan; Wendler, Philipp: Conditional Model Checking: A Technique to Pass Information Between Verifiers. In: FSE. FSE '12. ACM, pp. 1–11, 2012.

[BK11] Beyer, Dirk; Keremoglu, M.Erkan: CPAchecker: A Tool for Configurable Software Verification. In (Gopalakrishnan, Ganesh; Qadeer, Shaz, eds): CAV. volume 6806 of LNCS. Springer, pp. 184–190, 2011.

[CDE08] Cadar, Cristian; Dunbar, Daniel; Engler, Dawson: KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs. In: OSDI. OSDI'08. USENIX Association, pp. 209–224, 2008.

[CJW15] Czech, Mike; Jakobs, Marie-Christine; Wehrheim, Heike: Just Test What You Cannot Verify! In (Egyed, Alexander; Schaefer, Ina, eds): FASE, volume 9033 of LNCS, pp. 100–114. Springer Berlin Heidelberg, 2015.

[Cu12] Cuoq, Pascal; Kirchner, Florent; Kosmatov, Nikolai; Prevosto, Virgile; Signoles, Julien; Yakobowski, Boris: Frama-C. In (Eleftherakis, George; Hinchey, Mike; Holcombe, Mike, eds): SEFM. volume 7504 of LNCS. Springer, pp. 233–247, 2012.

Errors Made During Manual Test Case Derivation from UML Activity Diagrams and State Machines: Results of a Controlled Experiment

Michael Felderer[1], Andrea Hermann[2]

1 Overview

This talk presents our recent Information and Software Technology journal article [FH15] on a controlled experiment on manual test case derivation from UML activity diagrams and state machines. Manual test case derivation from behavioral models like UML activity diagrams or state machines is frequently applied in practice. But this kind of manual test case derivation is error-prone and knowing these errors makes it possible to provide guidelines to reduce them. The objective of the study presented in this talk therefore is to examine which errors are possible and actually made when manually deriving test cases from UML activity diagrams or state machines and whether there are differences between these diagram types. We investigate the errors made when deriving test cases manually in a controlled student experiment. The experiment was performed and internally replicated with overall 84 participants divided into three groups at two institutions. As a result of our experiment, we provide a taxonomy of errors made and their frequencies. In addition, our experiment provides evidence that activity diagrams have a higher perceived comprehensibility but also a higher error-proneness than state machines with regard to manual test case derivation. This information enables the development of guidelines for manual test case derivation from UML activity diagrams and state machines which help to make manual test case derivation less error-prone and are also discussed in this talk.

2 References

[FH15] Felderer, M.; Herrmann, A.: Manual test case derivation from UML activity diagrams and state machines: A controlled experiment. Information & Software Technology, 61:1-15, 2015.

[1] University of Innsbruck, Innsbruck, Austria, michael.felderer@uibk.ac.at
[2] 2Herrmann & Ehrlich, Stuttgart, Germany, herrmann@herrmann-ehrlich.de

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 21

Accurate Profiling in the Presence of Dynamic Compilation

Yudi Zheng[1], Lubomír Bulej[2], Walter Binder[3]

Abstract: Many programming languages are implemented on top of a managed runtime system, such as the Java Virtual Machine (JVM) or the .NET CLR, featuring an optimizing dynamic (just-in-time) compiler. Programs written in those languages are first interpreted (or compiled by a baseline compiler), whereas frequently executed methods are later compiled by the optimizing dynamic compiler.

Common feedback-directed optimizations [AHR02] performed by state-of-the-art dynamic compilers, such as the optimizing compiler in the Jikes RVM [Ar00] or Graal [Op], include method inlining and stack allocation of objects based on (partial) escape analysis [Ch99, SWM14], amongst others. Such optimizations result in compiled machine code that does not perform certain operations present at the bytecode level. In the case of inlining, method invocations are removed. In the case of stack allocation, heap allocations are removed and pressure on the garbage collector is reduced.

Many profiling tools are implemented using bytecode instrumentation techniques, inserting profiling code into programs at the bytecode level. However, because dynamic compilation is transparent to the instrumented program, a profiler based on bytecode instrumentation techniques is not aware of the optimizations performed by the dynamic compiler. Prevailing profilers based on bytecode instrumentation suffer from two serious limitations: (1) *over-profiling* of code that is optimized (and in the extreme case completely removed) by the dynamic compiler, and (2) *perturbation* of the compiler optimizations due to the inserted instrumentation code.

We present a novel technique to make profilers implemented with bytecode instrumentation techniques aware of the optimization decisions of the dynamic compiler, and to make the dynamic compiler aware of inserted profiling code. Our technique enables profilers which collect dynamic metrics that (1) correspond to an execution of the base program without profiling (w.r.t. the applied compiler optimizations), and (2) properly reflect the impact of dynamic compiler optimizations.

We implement our approach in a state-of-the-art Java virtual machine and demonstrate its significance with concrete profilers. We quantify the impact of escape analysis on allocation profiling, object lifetime analysis, and the impact of method inlining on callsite profiling. We illustrate how our approach enables new kinds of profilers, such as a profiler for non-inlined callsites, and a testing framework for locating performance bugs in dynamic compiler implementations.

This work was originally presented at OOPSLA'15 [ZBB15], where it received a *Distinguished Paper Award* as well as an endorsement from the Artifact Evaluation Committee for having submitted an easy-to-use, well-documented, consistent, and complete artifact[4]. In the meantime, the work has been integrated into the Graal project.

[1] Università della Svizzera italiana (USI), Faculty of Informatics, Switzerland, yudi.zheng@usi.ch
[2] Università della Svizzera italiana (USI), Faculty of Informatics, Switzerland, lubomir.bulej@usi.ch; Charles University, Faculty of Mathematics and Physics, Czech Republic
[3] Università della Svizzera italiana (USI), Faculty of Informatics, Switzerland, walter.binder@usi.ch
[4] `http://dag.inf.usi.ch/software/prof.acc/`

Acknowledgments

The research presented here has been supported by Oracle (ERO project 1332), by the Swiss National Science Foundation (project 200021_141002), by the European Commission (contract ACP2-GA-2013-605442), and by the Charles University institutional funding (SVV). We especially thank Thomas Würthinger and Lukas Stadler for their support with Graal.

References

[AHR02] Arnold, Matthew; Hind, Michael; Ryder, Barbara G.: Online Feedback-directed Optimization of Java. In: Proc. 17th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. OOPSLA '02. ACM, pp. 111–129, 2002.

[Ar00] Arnold, Matthew; Fink, Stephen; Grove, David; Hind, Michael; Sweeney, Peter F.: Adaptive Optimization in the Jalapeño JVM. In: Proc. 15th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. OOPSLA '00. ACM, pp. 47–65, 2000.

[Ch99] Choi, Jong-Deok; Gupta, Manish; Serrano, Mauricio; Sreedhar, Vugranam C.; Midkiff, Sam: Escape Analysis for Java. In: Proc. 14th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. OOPSLA '99. ACM, pp. 1–19, 1999.

[Op] OpenJDK: , The Graal Compiler Project. `http://openjdk.java.net/projects/graal/`.

[SWM14] Stadler, Lukas; Würthinger, Thomas; Mössenböck, Hanspeter: Partial Escape Analysis and Scalar Replacement for Java. In: Proc. IEEE/ACM International Symposium on Code Generation and Optimization. CGO '14. ACM, pp. 165:165–165:174, 2014.

[ZBB15] Zheng, Yudi; Bulej, Lubomír; Binder, Walter: Accurate Profiling in the Presence of Dynamic Compilation. In: Proc. 30th ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications. OOPSLA '15. ACM, pp. 433–450, 2015.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 23

Transparent Object Proxies for JavaScript

Matthias Keil[1], Omer Farooq[1], Sankha Narayan Guria[2], Andreas Schlegel[1], Manuel Geffken[1], and Peter Thiemann[1]

Abstract:

This work appeared in the conference proceedings of the *European Conference on Object-Oriented Programming, ECOOP 2015*.

One important question in the design of a proxy API is whether a proxy object should inherit the identity of its target. Apparently proxies should have their own identity for security-related applications whereas other applications, in particular contract systems, require transparent proxies that compare equal to their target objects.

In this work we examine the issue with transparency in various use cases for proxies, discuss different approaches to obtain transparency, and propose two designs that require modest modifications in the JavaScript engine and cannot be bypassed by the programmer.

The JavaScript Proxy API embodies a design decision that reveals the presence of proxies in some important use cases. This decision concerns object equality. Proxies are *opaque*, which means that each proxy has its own identity, different from all other (proxy or non-proxy) objects.

Given opaque proxies, an equality test can be used to distinguish a proxy from its target as demonstrated in the following example:

```
1 var target = { /* some object */ };
2 var handler = { /* empty handler */ };
3 var proxy = new Proxy (target, handler);
4 proxy===target; // evaluates to false
```

Even though *target* and *proxy* behave identically, they are not considered equal. Thus, in a program that uses object equality, the introduction of a proxy along one execution path may change the meaning of the program without even invoking an operation on the proxy (which may behave differently from the same operation on the target).

Equality for opaque proxies works well under the assumption that proxies and their targets are never part of the same execution environment. But the assumption that proxies never share their execution environment with their targets is not always appropriate. One prominent use case is the implementation of a contract system.

Two examples for such systems are the contract framework of Racket [FFP14, Chapter 7] and TreatJS for JavaScript [KT15]. Both systems implement contracts on objects with spe-

[1] University of Freiburg, Freiburg, Germany, {keilr,schlegea,geffken,thiemann}@informatik.uni-freiburg.de
[2] Indian Institute of Technology Jodhpur, Jodhpur, India, sankha@iitj.ac.in

cific wrapper objects, Racket's chaperones or impersonators [St12] and JavaScript proxies, respectively. But this may chang the semantics of a program and thus it violates a ground rule for monitoring: a monitor should never interfere with a program conforming to the monitored property.

Our ECOOP paper [Ke15] shows that a significant number of object comparisons would fail when mixing opaque proxies and their target objects, e.g. when gradually adding contracts to a program. As neither the transparent nor the opaque implementation of proxies is appropriate for all use cases, we propose an alternative designs for *transparent proxies* that is better suited for use cases such as certain contract wrappers and access restricting membranes.

We use object capabilities to create proxies in a particular realm and to create an equality function that only reveals proxies for that realm. A new realm constructor returns a new transparency realm represented by an object that consists of a fresh constructor for transparent proxies (named *Constructor*) and an *equals* function revealing proxies of that realm.

```
var realm = TransparentProxy.createRealm();
var proxy == realm.Constructor(target, handler);
proxy===target; // true
realm.equals(proxy, target); // false
```

The proxy *proxy* is transparent with respect to equality unless someone uses the *realm.equals* method. The *realm.equals* method is a capability that represents the right to reveal proxies of that realm. In addition, the realm also contains a constructor for realm-aware weak maps and weak sets.

References

[Bo15] Boyland, John Tang, ed. 29th European Conference on Object-Oriented Programming, ECOOP 2015, July 5-10, 2015, Prague, Czech Republic, volume 37 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[FFP14] Flatt, Matthew; Findler, Robert Bruce; PLT: . The Racket Guide, v.6.0 edition, March 2014. `http://docs.racket-lang.org/guide/index.html`.

[Ke15] Keil, Matthias; Guria, Sankha Narayan; Schlegel, Andreas; Geffken, Manuel; Thiemann, Peter: Transparent Object Proxies in JavaScript. In: (Boyland) [Bo15], pp. 149–173.

[KT15] Keil, Matthias; Thiemann, Peter: TreatJS: Higher-Order Contracts for JavaScripts. In: (Boyland) [Bo15], pp. 28–51.

[St12] Strickland, T. Stephen; Tobin-Hochstadt, Sam; Findler, Robert Bruce; Flatt, Matthew: Chaperones and impersonators: run-time support for reasonable interposition. In (Leavens, Gary T.; Dwyer, Matthew B., eds): Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012, part of SPLASH 2012, Tucson, AZ, USA, October 21-25, 2012. ACM, pp. 943–962, 2012.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 25

Intelligent Code Completion with Bayesian Networks

Sebastian Proksch[1], Johannes Lerch[1] and Mira Mezini[1]

Abstract: Code completion is an integral part of modern Integrated Development Environments (IDEs). Intelligent code completion systems can reduce long lists of type-correct proposals to relevant items. In this work, we replace an existing code completion engine named Best-Matching Neighbor (BMN) by an approach using Bayesian Networks named Pattern-based Bayesian Network (PBN).We use additional context information for more precise recommendations and apply clustering techniques to improve model sizes and to increase speed.

We compare the new approach with the existing algorithm and, in addition to prediction quality, we also evaluate model size and inference speed. Our results show that the additional context information we collect improves prediction quality, and that PBN can obtain comparable prediction quality to BMN, while model size and inference speed scale better with large input sizes.

Keywords: Recommender System, Static Analysis, Machine Learning, Evaluation

1 Motivation

Code completion systems are an integral part of modern Integrated Development Environments (IDEs). They reduce the amount of typing required, thus accelerating coding, and are often used by developers as a quick reference for the Application Programming Interface (API), because they show which fields and methods can be accessed in a certain context.

Traditional code completion systems determine the static type of the variable on which the developer triggers the completion and propose all type-correct methods to the developer. Such a list is often very long with many irrelevant items. More intelligent code completion systems reduce this list to relevant items. They extract a feature vector that describes the current edit location, match this feature vector to released source code found in repositories, and propose relevant methods based on the identified example code.

The prediction quality of intelligent code completion systems mainly depends on the available features, the number of analyzed repositories, and the underlying model that calculates the proposals. To further enhance the results of existing code completion approaches, more repositories need to be analyzed in order to see more examples and more features need to be extracted to have a more precise description of the context. Both results in a rapidly growing size of the available input and not all existing approaches scale well.

Existing completion engines are typically evaluated based on their prediction quality. As the code completion engine is supposed to be used by end users on machines with limited

[1] Technische Universität Darmstadt, Fachbereich Informatik, Fachgebiet Softwaretechnik, Hochschulstr. 10, 64289 Darmstadt, Deutschland, <lastname>@st.informatik.tu-darmstadt.de

resources, it is also necessary to consider memory consumption of the underlying models and computation speed. The three quality dimensions - prediction quality, prediction speed, and model sizes - are not orthogonal and the mutual effect they have on each other must be considered. The hypothesis is that prediction quality is increased by considering more features of the structural context. However, this will presumably increase the model size and negatively affect prediction speed. We need code completion engines that provide a good tradeoff between these quality dimensions or are even configurable along them.

2 Contributions

This paper contributes towards tackling these problems and presents advances to the state of the art in intelligent code completion systems in three ways:

(1) We extended the static analysis of the *best-matching neighbor* approach (BMN) and extracted more context information. We show that this indeed improves prediction quality by up to 3% at the cost of significantly increased model sizes by factor 2 and more.

(2) We introduced a new approach for intelligent code completion called *pattern-based bayesian network* (PBN), a new technique to infer intelligent code completions that enables to reduce model sizes via clustering. We introduced a clustering approach for PBN that enables to trade-off model size for prediction quality.

(3) We extended the state-of-the-art methodology for evaluating code completion systems. We perform comprehensive experiments to investigate the correlation between prediction quality and different model sizes. We show that clustering can decrease the model size by as much as 90% with only minor decrease of prediction quality. We also perform a comprehensive analysis of the effect of input data size on prediction quality, speed and model size. Our experiments show that prediction quality increases with increased input data and that both the model size and prediction speed scales better with the input data size for PBN compared to BMN.

We have released downloadable artifacts to allow replication of our results.[2] The released artifacts includes the dataset used in the paper and the complete source code (i.e., all intelligent code completion engines and the evaluations). We encourage other researches to compare their results based on the same data set.

3 Summary

Our results show that the additional context information we collect improves prediction quality, especially for queries that do not contain method calls. We also show that PBN can obtain comparable prediction quality to BMN, while model size and inference speed scale better with large input sizes.

[2] http://www.st.informatik.tu-darmstadt.de/artifacts/pbn/

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 27

Automated Workload Characterization for I/O Performance Analysis in Virtualized Environments

Axel Busch[1], Qais Noorshams[2], Samuel Kounev[3], Anne Koziolek[4], Ralf Reussner[5], Erich Amrehn[6]

Modern applications, such as mail servers, file servers, or video servers show highly I/O-intensive workload patterns. Their huge data volumes require powerful storage infrastructures. These applicarions are increasingly deployed in virtualized environments due to cost efficiency aspects. Nevertheless, consolidating several applications on one shared infrastructure introduces complex performance implications due to mutual interferences. To consolidate several applications while respecting certain Service Level Agreements necessitates a predicion of these implications up front. Such a prediction, however, requires tailored performance models that in turn require a significant amount of expertise to create the models [Kr12, Kr11]. Moreover, their accuracy depends on the quality of the input parameters which are often unclear how they could be determined [CH11]. We adress this discrepancy in our work. We develop an automated workload characterization approach to extract workload models [Ko09] that are representations of the main aspects of I/O-intensive applications in virtualized environments. We have tailored our approach to enable a non-invasive and lightweight monitoring, yet with a level of abstraction such that the parameters are practically obtainable. To evaluate our approach, we perform a comprehensive evaluation demonstrating its workload modeling performance for common business workloads using two case studies. The case studies demonstrate typical real-worl scenarios, such as consolidation of several workloads on one machine, and workload migration between two systems.

Our approach analysis the *low-level* read and write request that are generated by a *high-level* workload, such as a file server workload. The requests' properties are described by a formalized set of metrics (determined from one of our previous works [NKR13]). These metrics are particularly build for modeling I/O-intensive workloads in virtualized environments. Once extracted, the values are mapped to our reference benchmark, the *Flexible File System Benchmark* (FFSB). FFSB allows to emulate the original low-level workload on the target system.

The metrics set is comprised of six metrics: The average *file size* determining the space of the files physically allocated on the disk, limiting sequential requests. The *file set size* con-

[1] Karlsruhe Institute of Technology, busch@kit.edu
[2] Karlsruhe Institute of Technology, noorshams@kit.edu
[3] University of Wuerzburg, samuel.kounev@uni-wuerzburg.de
[4] Karlsruhe Institute of Technology, koziolek@kit.edu
[5] Karlsruhe Institute of Technology, reussner@kit.edu
[6] IBM Research&Development, amrehn@de.ibm.com

siders the total allocated space that influences the locality of requests, and other strategies, such as data placement and caching strategies. Further, we include the *workload intensity* that determines the running of parallel jobs accessing the disks. The *request mix* determines the ration between read and write requests, while the *average request size* models the average size of each read and write request that is accessed sequentially. Finally, the *disk access pattern* is represented by a heuristic algorithm extracting the ratio of sequential and parallel accesses.

Our approach uses the aforementioned metrics to extract the workload characteristics. For an automated execution, we formalized and implemented our metrics set in the *Storage Performance Analyzer* (SPA). SPA is a tool allowing to extract the workload characteristics automatically using a certain set of metrics.

For our evaluation, we use two state-of-the-art high performant virtualization environments, namely an IBM SYSTEM Z, equipped with a DS8700 storage system, and a SUN FIRE X4440 server system. We use two real-world workloads, namely a mail server and a file server workload. We generated our workloads using *Filebench*, a storage system benchmark that is widely used in the performance modeling community [Kr12, Ah07]. The extracted results are then used in two different scenarios, namely a migration and additionally a consolidation scenario. In the first, we used FFSB with the metric values of the file server workload to estimate its performance on the Sun Fire system. In the second, we emulate both workloads on the Sun Fire system at the same time. Again, it should be mentioned that both workloads are characterized on the IBM system. For the migration scenario we could show a prediction error of 21.59 % for read, and 20.98 % error for the write requests. In case of the consolidation scenario, we demonstrate 12.95 % error for read and 24.52 % for write requests. Both scenarios show the applicability of our approach that benefits in a fast and low-overhead estimation of an I/O-intensive workload that does not rely on complex performance prediction models. The demonstrated accuracy should be sufficient for initial estimations of the workload behaviour.

Literaturverzeichnis

[Ah07] Ahmad, Irfan: Easy and Efficient Disk I/O Workload Characterization in VMware ESX Server. IEEE Computer Society, 2007.

[CH11] Chiang, Ron C.; Huang, H. Howie: TRACON: interference-aware scheduling for data-intensive applications in virtualized environments. SC'11, New York, NY, USA, 2011.

[Ko09] Kounev, Samuel: Wiley Encyclopedia of Computer Science and Engineering - chapter Software Performance Evaluation. Wiley-Interscience and John Wiley & Sons Inc., 2009.

[Kr11] Kraft, Stephan; Casale, Giuliano; Krishnamurthy, Diwakar; Greer, Des; Kilpatrick, Peter: IO Performance Prediction in Consolidated Virtualized Environments. ICPE, 2011.

[Kr12] Kraft, Stefan: Performance Models of Storage Contention in Cloud Environments. Journal of Software and Systems Modeling (SoSyM), 2012.

[NKR13] Noorshams, Qais; Kounev, Samuel; Reussner, Ralf: Experimental Evaluation of the Performance-Influencing Factors of Virtualized Storage Systems. Springer Berlin Heidelberg, 2013.

Performance-Influence Models for Highly Configurable Systems

Norbert Siegmund[1] Alexander Grebhahn[2] Sven Apel[3] Christian Kästner[4]

1 Introduction

The original paper has been published in the proceedings of ESEC/FSE 2015 [SGAK15]. End-users, developers, and administrators are often overwhelmed with the possibilities to *configure* a software system. In most systems today, including databases, Web servers, video encoders, and compilers, hundreds of configuration options can be combined, each potentially with distinct functionality and different effects on quality attributes. The sheer size of the configuration space and complex constraints between configuration options make it difficult to find a configuration that performs as desired, with the consequence that many users stick to default configurations or only try changing an option here or there. This way, the significant optimization potential already present in many of our modern software systems remains untapped. Even domain experts and the developers themselves often do not (fully) understand the performance influences of all configuration options and their combined influence when options interact.

Our goal is to build performance-influence models (and models of other measurable quality attributes, such as energy consumption) that describe how configuration options and their interactions influence the performance of a system (e.g., throughput or execution time of a benchmark). A distinctive feature of our approach is that we consider both binary and numeric options and that we do not solely target prediction accuracy. Performance-influence models are meant to ease *understanding, debugging, and optimization* of highly configurable software systems. For example, a user may identify the best performing configuration from the model and a developer may compare an inferred performance-influence model with her own mental model to check whether the system behaves as expected.

2 Approach

Our approach is to infer a performance-influence model for a given configurable system in a black-box manner, from a series of *measurements* of a set of *sample* configurations using

[1] University of Passau, Germany
[2] University of Passau, Germany
[3] University of Passau, Germany
[4] Carnegie Mellon University, USA

machine learning. That is, we benchmark a given system multiple times in different configurations and learn the influence of individual configuration options and their interactions from the differences between the measurements. Conceptually, a performance-influence model is simply a function from a configuration $c \in \mathscr{C}$ to a performance measure $\Pi : \mathscr{C} \to \mathbb{R}$, where performance can be any measurable property that produces interval-scaled data. All performance-influence models are of the following form:

$$\Pi(c) = \beta_0 + \sum_{i \in \mathscr{O}} \phi_i(c(i)) + \sum_{i..j \in \mathscr{O}} \Phi_{i..j}(c(i)..c(j)) \tag{1}$$

where β_0 represents a minimum, constant base performance shared by all configurations, as determined during learning; $\sum_{i \in \mathscr{O}} \phi_i(c(i))$ represents the sum of the influences of all individual options; $\sum_{i..j \in \mathscr{O}} \Phi_{i..j}(c(i)..c(j))$ is the sum of the influences of all interactions among all options. This structure allows us to easily see the influence of an individual option or an interaction between options from the model.

Learning. We use *stepwise linear regression* to learn the function of a performance-influence model from a sample set of measured configurations. To reduce the dimensionality problem of handling a very large number of options and interactions, we use *feature subset selection* to incrementally learn the model. The key challenge of using linear regression is to identify the relevant terms to be used as independent variables; a term represents the (possibly non-linear) influence of one or multiple configuration options. Conceptually, any combination of options may cause a distinct performance interaction [SKK+12], which would render any learning approach useless, as there is no common pattern. In practice, however, performance behavior is usually more tractable in that only few interactions contribute substantially to the overall performance. In our previous work, we found that relevant interactions do not emerge randomly between configuration options, but form a hierarchy [SvRA13]. Thus, we perform our learning hierarchically and incrementally: Starting with an empty model, our algorithm selects one term in each iteration until improvements of model accuracy become marginal or a threshold for expected accuracy is reached. The term to be added stems from a number of candidate terms. The initial candidates are only the individual option influences, which are then extended by candidates representing interactions between options that have been found already to contribute to performance, and additional functions (e.g., logarithmic or quadratic) representing the influence of numeric options.

Sampling. We divide the configuration space along binary and numeric configuration options and apply structured sampling heuristics to them. For binary sampling, we use heuristics developed in previous work that aim at selecting configurations such that we can learn the influences of individual options and of pair-wise interactions. For sampling numeric options, we use a number of experimental designs, including fractional factorial designs and optimal designs. We found that the Plackett-Burman design provides a sweet spot between measurement effort and accuracy of the learned model. The separately selected configurations for binary and numeric options are combined using the cross product.

Experiments. Our approach is able to build reasonably accurate performance models of configuration spaces of real-world systems, including compilers, multi-grid solvers, and video encoders. In a series of experiments with configurable systems with up to 10^{31}

configurations, we found that few measurements are sufficient to build fairly accurate models (19 % prediction error, on average). The performance-influence models learned by our approach can explain the performance variation between configurations with a few dozen terms describing the influence of individual options and another dozen terms describing interactions. Finally, while accuracy is important, simple models are important, too. Views on a performance-influence model can be used to isolate influences of individual options and their interactions.

References

[SGAK15] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. Performance-Influence Models for Highly Configurable Systems. In *Proc. ESEC/FSE*, pages 284–294. ACM, 2015.

[SKK+12] Norbert Siegmund, Sergiy Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. Predicting Performance via Automated Feature-Interaction Detection. In *Proc. ICSE*, pages 167–177. IEEE, 2012.

[SvRA13] Norbert Siegmund, Alexander von Rhein, and Sven Apel. Family-Based Performance Measurement. In *Proc. GPCE*, pages 95–104. ACM, 2013.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft fr Informatik, Bonn 2016 33

Scaling Size and Parameter Spaces in Variability-aware Software Performance Models

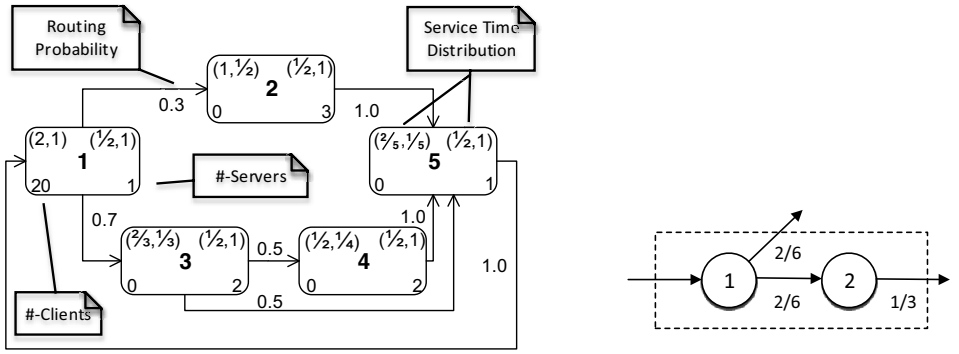
Matthias Kowal[1], Max Tschaikowski[2], Mirco Tribastone[3], Ina Schaefer[4]

Abstract: Model-based software performance engineering often requires the analysis of many instances of a model to find optimizations or to do capacity planning. These performance predictions get increasingly more difficult with larger models due to state space explosion as well as large parameter spaces since each configuration has its own performance model and must be analyzed in isolation (product-based (PB) analysis). We propose an efficient family-based (FB) analysis using UML activity diagrams with performance annotations. The FB analysis enables us to analyze all configurations at once using symbolic computation. Previous work has already shown that a FB analysis is significant faster than its PB counterpart. This work is an extension of our previous research lifting several limitations.

1 Coxian Distributions and PB-Evaluation of PAADs

Performance Annotated Activity Diagrams (PAAD) capture the workflow of a software system and enhance it with performance-related properties [KST14]. An example can be found in Fig. 1a. Each node represents a service center in the software system, e.g. CPUs, web server and so forth, and has the following performance annotations at its corners: vectors for the service time distribution (top left and right values), number of clients at that node during the initial condition (bottom left) and number of servers (bottom right). Edges connect the nodes and are annotated with probabilities denoting the likelihood of a job to take that path. We can construct a continuous-time Markov chain (CTMC), where the length of either vector denotes the actual number of states in the CTMC (or stages of the distribution). The left vector provides the rate of the exponential residence time at each state, while the right vector contains the probability with which a service process moves from one state to the next. The time between entering the first state and exiting from any other state gives us a non-exponential distribution for the service at the specific node. Fig. 1b shows the CTMC for such a Coxian distribution. The services will be exponentially distributed with $2/6+2/6=2/3$ in state 1 and enter state 2 with a probability of $1/2$ encountering an additional delay of $1/3$. Coxian distributions provide better representations of real-world software systems, since they can be seen as a composition of exponential stages and are able to approximate any given general distribution [St09]. In addition, we can now simulate parallelism with multiple servers that are available at a node. Both aspects remove a restriction of our previous work in [KST14]. The calculation of the steady state throughput for such Coxian-distributed multi-server nodes is a non-trivial task that involves solving the system of Ordinary Differential Equations (ODE) given by $R^T T = T$. R is the routing probability matrix and T determines the ODE throughputs. In the PB analysis, we have to solve it for each variant in isolation, which is inefficient. Mean service times as well as number of servers and clients also play role in the calculation of T, but their relation to the ODE system is omitted here.

[1] Technische Universität Braunschweig, Germany
[2] IMT Institute for Advanced Studies Lucca, Italy
[3] IMT Institute for Advanced Studies Lucca, Italy
[4] Technische Universität Braunschweig, Germany



(a) A Performance Annotated Activity Diagram

(b) Coxian CTMC for node 3.

Figure 1: Running Example

2 Variability and FB-Evaluation

A FB analysis is only reasonable if variability is included into the PAADs. Similar to our previous work, we applied the principle of delta modeling (DM) in which deltas can *add*, *remove* or *modify* PAAD elements. Given a specific *core* PAAD, we can generate any variant of the system by applying the respective deltas [Sc10]. The FB analysis relies on the construction of a 150%-model or super-variant. This model is built by merging the core and all deltas into one large model. Each PAAD element that is changed by a delta is represented as a symbol in the 150%-model and not its concrete value, e.g. removing the edge between node 3 and 5 would modify the probability between node 3 and 4 to 1.0 (cf. Fig. 1a) and result in two symbolic parameters in R. Again, we can construct the ODE system, but solve it symbolically this time, which has to be done just once. The steady state throughputs T are now calculated by plugging the concrete values for the desired variant into the parametrized expressions. The FB analysis is faster compared to the PB one and gets even more efficient for larger networks or an increasing number of variants.

References

[KST14] Kowal, Matthias; Schaefer, Ina; Tribastone, Mirco: Family-Based Performance Analysis of Variant-Rich Software Systems. In: FASE 2014. pp. 94–108, 2014.

[Sc10] Schaefer, Ina: Variability Modelling for Model-Driven Development of Software Product Lines. In: VaMoS. pp. 85–92, 2010.

[St09] Stewart, William J.: Probability, Markov Chains, Queues, and Simulation. Princeton University Press, 2009.

Naming the Pain in Requirements Engineering: A Survey Design and German Results

Daniel Méndez Fernández[1] Stefan Wagner[2]

Abstract: This paper summarises the results published in *Information and Software Technology* in January 2015. Although researchers are investigating requirements engineering with a plethora of empirical studies, a broad empirical basis is still missing. To get a foundation about the state of the practice in RE, we propose a distributed family of open and reproducible surveys. The instrument is based on a theory that integrates a set of hypotheses inferred from our experiences and available, isolated studies. We test each hypothesis in our theory and identify further candidates to extend the theory by correlation and Grounded Theory analysis. Our results from Germany reveal, for example, a tendency to improve RE via internally defined qualitative methods rather than relying on normative approaches like CMMI. The survey design proved itself useful and is, at present, now employed in 14 countries in total (see also our website: `www.re-survey.org`). We found that surprisingly many aspects of the status quo and the problems are similar in the surveyed countries. Yet, there are also notable differences. We will report on both the survey design and the detailed results from Germany, and we will give an outlook on the results of the current world-wide replications of the survey.

Keywords: Requirements Engineering, Survey Research, Family of Surveys

1 Introduction

Requirements engineering (RE) is a key to successful development projects as the elicitation, specification and validation of precise and stakeholder-appropriate requirements are critical determinants of software & system quality [Br06]. Although the importance of a high quality RE has been recognised for many years, we can still observe industry struggling in defining and applying a high quality RE [Me12]. The diversity of how RE is performed in various industrial environments, each having its particularities in the domains of application or the software process models used, renders process improvement and, in particular, empirical research difficult.

Our long-term research objective is to establish an open and externally valid set of empirical findings about practical problems and needs in RE that allows us to steer future research in a problem-driven manner. To this end, we conduct a continuously and independently replicated, globally distributed survey on RE that investigates the state of the practice including the status quo, experienced problems as well as related causes and effects. Here, we report the design of the family of surveys on RE and the results obtained from its initial start in Germany (73 completed questionnaires). Our instrument relies on an initial theory obtained from available RE studies and is used to generate hypotheses

[1] Technische Universität München, Garching, Daniel.Mendez@tum.de
[2] University of Stuttgart, stefan.wagner@informatik.uni-stuttgart.de

which we test during the results analysis. The results gathered from open questions are further used to already extend our initial theory using Grounded Theory analysis. Furthermore, we investigate patterns in statistically significant correlations to find further candidate hypotheses for the theory. Finally, we will give an outlook on the results of the current world-wide replications of the survey. We published the full details of the design and the results from Germany in [MFW15].

2 Survey Design and Results from Germany

We can only give two examples of research questions from the survey design and the results from Germany. For example, we had the research question: **How is RE defined, applied, and controlled?** One hypothesis in this area was the mainly, *Requirements are elicited via workshops.* Using the results from Germany, we could corroborate this hypothesis. Another research question was: **Which contemporary problems exist in RE, and what implications do they have?** We found that *incomplete or hidden requirements* was the top-rated problem in RE practice followed by *moving targets* and time boxing.

3 Outlook

At present, we are in the process of finalising the second round of surveys which includes Germany again but has replications all over the world. It has not been a direct replication, but we refined and extended the underlying theory based on the results reported here as well as the discussions with all collaborators. We found that surprisingly many aspects of the status quo and the problems are similar in the surveyed countries. Yet, there are also notable differences. For example, we investigated the differences in RE problems between Brazilian and German countries in more detail [Me15] and found that moving targets pose a bigger problem in German companies while Brazilian companies have more problems with human collaboration.

References

[Br06] Broy, M.: Requirements Engineering as a Key to Holistic Software Quality. In (Levi, A.; Savas, E.; Yenigun, H.; Balcisoy, S.; Saygin, Y., eds): Proceedings of the 21th International Symposium on Computer and Information Sciences (ISCIS 2006). volume 4263. Springer-Verlag Berlin, pp. 24–34, 2006.

[Me12] Mendez Fernandez, D.; Wagner, S.; Lochmann, K.; Baumann, A.; de Carne, H.: Field Study on Requirements Engineering: Investigation of Artefacts, Project Parameters, and Execution Strategies. Information and Software Technology, 54(2):162–178, 2012.

[Me15] Mendez Fernandez, D.; Wagner, S.; Kalinowski, M.; Schekelmann, A.; Tuzcu, A.; Conte, T.; Spinola, R.; Prikladnicki, R.: Naming the Pain in Requirements Engineering: Comparing Practices in Brazil and Germany. IEEE Software, 32(5):16–23, Sept 2015.

[MFW15] Méndez Fernández, D.; Wagner, S.: Naming the Pain in Requirments Enginering: A Design for a global Family of Surveys and First Results from Germany. Information and Software Technology, 57:616–643, 2015.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 37

Supporting the Systematic Assessment of Requirements Traceability - A Quality Model

Patrick Rempel[1] and Patrick Mäder[2]

Abstract: Traceability is an important quality of software requirements and allows to describe and follow their life throughout a development project. The importance of traceable requirements is reflected by the fact that requirements standards, safety regulations, and maturity models explicitly demand for it. In practice, traceability is created and maintained by humans, which make mistakes. In result, existing traces are potentially of dubious quality but serve as the foundation for high impact development decisions. We found in previous studies that practitioners miss clear guidance on how to systematically assess the quality of existing traces. In this paper, we review the elements involved in establishing traceability in a development project and derive a quality model that specifies per element the acceptable state (Traceability Gate) and unacceptable deviations (Traceability Problem) from this state. We describe and formally define how both, the acceptable states and the unacceptable deviations can be detected in order to enable practitioners to systematically assess their project's traceability. We evaluated the proposed model through an expert survey. Participating experts considered the quality model to be complete and attested that its quality criteria are of high relevance. However, experts weight the occurrence of different traceability problems with different criticality. This information is useful for practitioners to quantify the impact of traceability problems and to prioritize the assessment of traceability elements.

Keywords: requirements traceability; traceability quality model; problem classes; assessment

1 Motivation and Challenges

Requirements traceability is a critical element of any rigorous software development process. It provides support for numerous software engineering tasks. However, achieving purposed and trustworthy traceability remains a challenge, which is not yet solved. In industrial practice, traceability is created and maintained by humans who make mistakes [RMK13]. The resulting traceability is often of dubious quality but serves as the foundation for high impact development decisions. Developing safety-critical systems requires the compilation of safety cases arguing that a system is safe for use, which typically involves traceability [Kel99]. In previous studies [RMKC14, MJZC13], we investigated projects that struggled with problems such as a lack of compliance or latent safety risk, which were caused by incomplete or missing traceability data. When reflecting on the results of those previous studies, we realized that no classification of traceability problems was available to systematically assess traceability for structural deficiencies. This lack of a well-defined problem classification makes it difficult for software practitioners to generate an understanding of possible traceability problems and how to recognize them.

[1] Technische Universität Ilmenau, Software Systems Group, Ilmenau, Germany, patrick.rempel@tu-ilmenau.de
[2] Technische Universität Ilmenau, Software Systems Group, Ilmenau, Germany, patrick.maeder@tu-ilmenau.de

2 A Requirements Traceability Quality Model

In this presentation we propose an enumeration of potential traceability quality problems along with an enumeration of possible conformance assessment results. The entire work has been published at [RM15]. A traceability assessment refers to determining the degree of traceability fulfillment and can lead to four possible results. First, the fulfilling set fully conforms with the set of required data. This kind of result is represented in our quality model by the traceability gate elements. Second, the fulfilling set is *incomplete*, because it misses data in order to fully conform with the set of required data. This state is represented in our quality model by the problem category missing traceability. Third, the fulfilling set is *redundant*, because it contains superfluous data, not necessary to conform with the set of required data. This state is represented in our quality model by the problem category superfluous traceability. Fourth, the fulfilling set is *incomplete* and *redundant*, and thus a composite problem category of missing traceability and superfluous traceability.

3 Results and Conclusions

We conducted a survey with 13 traceability experts to evaluate the completeness and usefulness of the proposed model. The participants considered the proposed assessment model to be *complete* and attested the model high practical relevance. Traceability problems related to *completeness* were consistently rated as more important than problems related to *appropriateness*. We assume this pattern is attributed to the differing problem implications. Problems related to the quality attribute appropriateness imply that unnecessary effort was spent. Problems related to the quality attribute completeness imply that traceability based decisions are made on incomplete data. Especially, within the context of safety critical software, this can be a potential threat to the functional safety of a system.

Acknowledgments We are funded by the German Ministry of Education and Research (BMBF) grant 01IS14026B.

References

[Kel99] T. Kelly. *Arguing safety-a systematic approach to managing safety cases*. York, 1999.

[MJZC13] Patrick Mäder, Paul L. Jones, Yi Zhang, and Jane Cleland-Huang. Strategic Traceability for Safety-Critical Projects. *IEEE Software*, 30(3):58–66, May 2013.

[RM15] Patrick Rempel and Patrick Mäder. A quality model for the systematic assessment of requirements traceability. In *Proc. 23rd IEEE International Requirements Engineering Conference*, pages 176–185, 2015.

[RMK13] Patrick Rempel, Patrick Mäder, and Tobias Kuschke. An empirical study on project-specific traceability strategies. In *Proceedings of the 21st IEEE International Requirements Engineering Conference*, pages 195–204. IEEE, 2013.

[RMKC14] Patrick Rempel, Patrick Mäder, Tobias Kuschke, and Jane Cleland-Huang. Mind the gap: assessing the conformance of software traceability to relevant guidelines. In *Proc. 36th International Conference on Software Engineering ICSE*, pages 943–954, 2014.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 39

Supporting Requirements Update during Software Evolution

Eya Ben Charrada[1] Anne Koziolek[2] Martin Glinz[3]

Abstract: Keeping the requirements specification up-to-date is crucial for several maintenance and evolution tasks. Nevertheless, due to time and budget constrains, software maintainers usually apply changes to the code only and leave the requirements unchanged, so they rapidly become obsolete and useless. In this work, we propose an approach for automatically identifying what requirements are likely to be impacted when the code is changed. We use a two-step approach. First, we identify the code changes that are likely to have an impact on requirements based on some heuristics. Second, we trace the relevant changes back to the requirements in order to identify those that are likely to be impacted [**?**]. The output of the tracing is a list of requirements that are sorted according to their likelihood of being impacted. We applied our approach to three case studies and could identify for each case between 70% and 100% of the impacted requirements within a list that includes less than 20% of the total number of requirements in the specification [**?**]. We are currently exploring ways to extend the approach to other types of software artifacts such as tests.

Keywords: Requirements evolution, requirements update, impact analysis, traceability, artifact synchronization

[1] Department of Informatics, University of Zurich, Switzerland, charrada@ifi.uzh.ch
[2] Department of Informatics, Karlsruhe Institute of Technology, Germany, koziolek@kit.edu
[3] Department of Informatics, University of Zurich, Switzerland, glinz@ifi.uzh.ch

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 41

Eine Domänenspezifische Sprache für die technologieübergreifende Bereitstellung von Web Services

Florian Rademacher[1] Martin Peters[1] Sabine Sachweh[1]

Abstract: Web Services realisieren geräteunabhängige Datenschnittstellen zur Kommunikation von Web-Applikationen mit Internet-Clients, wie Smartphone-Apps, und Maschinen im Industrie-4.0-Kontext. In der Mehrzahl werden diese Schnittstellen entweder mit Hilfe des REST-Paradigmas und Standards des World Wide Web (WWW) oder des SOAP-Protokolls und XML-Nachrichten implementiert.
Der Beitrag stellt eine Domänenspezifische Sprache (Domain-Specific Language; DSL) für die effiziente, technologieübergreifende Entwicklung von Web Services vor. Ein Codegenerator überführt die DSL-Angaben in Java-Code, welcher auf einem erweiterbaren Framework basiert, das Unterschiede zwischen verschiedenen Web-Service-Technologien abstrahiert.

Keywords: Domänenspezifische Sprachen, Codegenerierung, Web Services

1 Einführung

Die Gruppe der über das Internet kommunizierenden Geräte wächst in zunehmendem Maße und wird zugleich immer heterogener. Neben Computern und Smartphone-Apps nutzen mittlerweile auch Maschinen und Sensoren im Rahmen einer Industrie 4.0 Technologien für den Datenaustausch über das Internet. Web Services stellen ein etabliertes Mittel für den geräteunabhängigen Datenaustausch dar. Das auf WWW-Standards aufbauende REST-Paradigma und das XML-basierte SOAP-Protokoll sind dabei die am weitesten verbreiteten Web-Service-Technologien [GK13]. Während SOAP in Szenarien mit hohen Anforderungen bspw. an die Übertragungsqualität zum Einsatz kommt, werden REST-Schnittstellen für eine effiziente Kommunikation über das HTTP eingesetzt [PZL08].
Der Beitrag stellt eine DSL für die technologieübergreifende Entwicklung von Web Services vor. Sie basiert auf dem in [RPS15] eingeführten Framework, mit dem Entwickler Geschäftslogik parallel über beliebige Web-Service-Technologien, bspw. für Smartphone-Apps via REST und für Maschinen via SOAP, anbieten können.

2 Spezifikation der Domänenspezifischen Sprache

Abbildung 1 zeigt das semantische UML-Modell der DSL. Es basiert auf den Abstraktionen des Frameworks aus [RPS15] und beschreibt die Sprachkonstrukte als Klassen und ihre Beziehungen als Assoziationen. Pakete kapseln semantische Sprachbereiche. So enthält das *Types*-Paket das Typsystem der Sprache. Es erlaubt die Konstruktion strukturierter

[1] Fachhochschule Dortmund, Fachbereich Informatik, Otto-Hahn-Straße 23, 44227 Dortmund, vorname.nachname@fh-dortmund.de

Datentypen und Listen aus primitiven Basistypen. Das *Services*-Paket definiert Konzepte zur Modellierung von Web Services. Im Kontext der DSL ist ein *Service* ein benanntes Element, welches einen spezifischen *Request* entgegennimmt und eine bestimmte *Response* produziert. Das *Interfaces*-Paket ermöglicht die Assoziation einer technologieneutralen Service-Beschreibung mit Web-Service-Technologien wie REST oder SOAP.

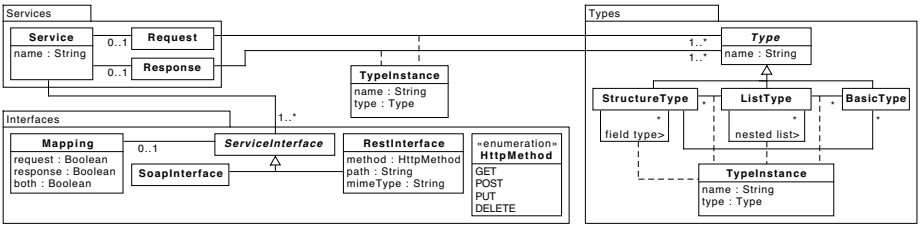


Abb. 1: Semantisches Modell der DSL

Listing 1 enthält Teile der aus dem Modell abgeleiteten DSL-Grammatik. Listing 2 zeigt einen mit der DSL modellierten Web Service, der per REST und SOAP verfügbar und Teil der Fallstudie aus [RPS15] ist. Ein Codegenerator überführt in der DSL vorliegenden

```
Service: 'service' name = ID ':' otoName = 'receives' otoVariables =
  TypeInstances itoName = 'returns' itoVariables = TypeInstances
  interfaces += ServiceInterface (interfaces += ServiceInterface)* ';' ;
ServiceInterface: name = 'interface' type = (RestInterface | SoapInterface)
  (mapping = MappingSpec)? ;
RestInterface: name = 'rest' 'method' method = HttpMethod 'path'
  path = STRING 'handles' mime = MimeSpec ;
SoapInterface: {SoapInterface} name = 'soap' ;
```

List. 1: Auszug der DSL-Grammatik

```
service UpdateParameterValue:
receives long wtsId, String paramName,
         Date timestamp, String value
returns int returnCode
interface rest method put
          path "wts/{wtsId}/{paramName}"
          handles "application/json"
interface soap;
```

List. 2: Beispiel-Service

Code in Framework-basierten Java-Code. Hierbei findet eine Transformation für jede konkrete Klasse aus Abbildung 1 statt. Die Geschäftslogik eines Services muss anschließend in einer Platzhaltermethode implementiert werden. Sie ist dann durch das Framework implizit über alle spezifizierten Web-Service-Technologien verfügbar. In einer erweiterten Fallstudie mit 25 Web Services, die parallel mittels REST und SOAP aufrufbar sein sollten, konnten aus 252 Zeilen DSL-Code 4384 Zeilen Java-Code generiert werden [RPS15]. Die im Folgenden implementierte Geschäftslogik umfasste 789 Zeilen Java-Code. Somit konnten rund 81% des Gesamtsystems automatisch erzeugt werden.

Literaturverzeichnis

[GK13] Gulden, Markus; Kugele, Stefan: A concept for generating simplified RESTful interfaces. In: Proceedings of the 22nd international conference on World Wide Web. International World Wide Web Conferences Steering Committee, S. 1391–1398, 2013.

[PZL08] Pautasso, Cesare; Zimmermann, Olaf; Leymann, Frank: RESTful Web Services vs. "Big"Web Services: Making the Right Architectural Decision. In: Proceedings of the 17th international conference on World Wide Web. ACM, S. 805–814, 2008.

[RPS15] Rademacher, Florian; Peters, Martin; Sachweh, Sabine: Design of a Domain-Specific Language Based on a Technology-Independent Web Service Framework. In: Software Architecture, S. 357–371. Springer, 2015.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 43

Learning how to prevent return-oriented programming efficiently

David Pfaff [1] Sebastian Hack[1] Christian Hammer[1]

1 Extended Abstract

The discovery of recent zero-day exploits against Microsoft Word, Adobe Flash Player and Internet Explorer demonstrate that return-oriented programming (ROP) is the most severe threat to software system security. Microsoft's 2013 Software Vulnerability Exploitation trend report found that 73% of all vulnerabilities are exploited via ROP. The core idea of ROP is to exploit the presence of so-called *gadgets*, small instruction sequences ending in a return instruction. By chaining gadgets together, an attacker is able to build complex exploits. The apparent popularity of ROP is explained by its power to bypass most contemporary exploit mitigation mechanisms, such as data execution prevention (DEP) and address space layout randomization (ASLR). DEP and similar page-protection schemes prevent the execution of injected binary code, but ROP re-uses code already present in the executable memory segments, eliminating the need to inject code. ASLR randomizes the location of most libraries and executables, however, finding code segments left in a few statically known locations is often enough to leverage a ROP attack. Since the inception of ROP by Shacham [Sh07], research on ROP resembles an arms race: emerging defense techniques are continuously circumvented by increasingly subtle attacks [CW14].

In our paper [PHH15], we take a novel, statistical approach on detecting ROP programs. Modern microprocessors spend most of their circuits on machinery that optimizes the execution of programs generated by compilers from "high-level" languages. Among this machinery are caches, translation look-aside buffers, branch predictors, and so on. To assist programmers in detecting performance problems, a modern CPU can record several hundred different kinds of micro-architectural events that occur during program execution (e.g. mispredicted branches, L1 cache misses, etc.). These events are counted by the CPU in special registers, the so-called *hardware performance counters* (HPCs).

In this paper, we claim *and experimentally verify* that the execution of a ROP program triggers such hardware events in a significantly different way than a conventional program that has been generated by a compiler. Essentially, micro-architectural events are a side channel by which a ROP program becomes distinguishable from a normal program at run time. There are several considerations that support this hypothesis: First, ROP programs use only indirect jumps (returns) to control the program flow. Common processor heuristics to detect the target of the return are useless in a ROP program because they do not

[1] CISPA, Saarland University, lastname@cs.uni-saarland.de

follow the call/return policy. Second, ROP gadgets are small and scattered all over the code segment. Thus, there is no spatial locality in the executed code which should be observable in counters relevant to the memory subsystem.

We exploit the deviant micro-architectural behavior of ROP programs by training (using existing ROP exploits and benign programs) a support vector machine (SVM) based on profiles of hardware performance counters. Note, that despite our intuition we did *not* short-list any HPC types for training. We receive a classifier to distinguish ROP from benign programs and use it in a *monitor* kernel module that tracks the evolution of the performance counters and classifies them periodically. If the classifier detects a ROP program, defensive actions, like killing the process, can be taken.

We quantitatively evaluate the performance impact of HadROP on benign program runs using the SPEC2006 benchmark: HadROP incurs a run time overhead of 5% on average and of 8% in the worst case. We also establish the effectiveness and practical applicability of HadROP in several case studies that show that HadROP detects and prevents the execution of a ROP payload of an in-the-wild exploit on Adobe Flash Player, 25 new ROP payloads generated by the ROP-payload generator Q that exploit manually injected vulnerabilities in GNU coreutils, Blind ROP [Bi14] of an nginx web server and multiple recent enhancements [CW14, Da14] that allow ROP to bypass previous hardware-assisted detection schemes. HadROP detects and prevents those attacks in any practical scenario.

References

[Bi14] Bittau, Andrea; Belay, Adam; Mashtizadeh, Ali; Mazieres, David; Boneh, Dan: Hacking blind. In: Proceedings of the 35th IEEE Symposium on Security and Privacy, S&P. 2014.

[CW14] Carlini, Nicholas; Wagner, David: ROP is Still Dangerous: Breaking Modern Defenses. In: 23rd USENIX Security Symposium (USENIX Security 14). USENIX Association, San Diego, CA, pp. 385–399, August 2014.

[Da14] Davi, Lucas; Sadeghi, Ahmad-Reza; Lehmann, Daniel; Monrose, Fabian: Stitching the Gadgets: On the Ineffectiveness of Coarse-Grained Control-Flow Integrity Protection. In: 23rd USENIX Security Symposium (USENIX Security 14). USENIX Association, San Diego, CA, pp. 401–416, August 2014.

[PHH15] Pfaff, David; Hack, Sebastian; Hammer, Christian: Learning How to Prevent Return-Oriented Programming Efficiently. In: Engineering Secure Software and Systems, pp. 68–85. Springer, 2015.

[Sh07] Shacham, Hovav: The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In: Proceedings of the 14th ACM conference on Computer and Communications Security. ACM, pp. 552–561, 2007.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 45

No PAIN, No Gain?
The Utility of PArallel Fault INjections

Stefan Winter[1] Oliver Schwahn[1] Roberto Natella[2] Neeraj Suri[1] Domenico Cotroneo[2]

Abstract: The article reports on interferences between concurrent fault injection test executions.

Keywords: Software fault injection, robustness testing, test interference

Software Fault Injection (SFI) emulates defects in software components to assess the robustness of other software components they interact with. After such defects have been introduced, the software composition is exposed to a workload and the effects of the introduced defect on the component under assessment are monitored. SFI tests entail relatively long execution times for three reasons: (1) They operate on a fully integrated software system, which entails corresponding loading and initialization times. (2) To activate the introduced defects and assess their impact on possibly complex component interactions, the execution of complex workloads is required. (3) After each SFI test execution, the entire software system under test (SUT) needs to be reset to a known fault-free state to prevent residual side effects of injected faults from affecting subsequent tests. Especially the last point entails significant execution time overhead. As the possible effects of an injection cannot be predicted (if they could, no SFI tests were needed), resetting the SUT usually requires a complete termination and re-initialization, sometimes even of its execution environment (e.g., the test machine's file system) if it can be affected by the injected fault.

To improve test throughput, we propose to exploit parallel hardware and execute SFI tests concurrently. While this appears to be a simple and straight-forward solution, it is based on an assumption of non-interference between SFI tests. In a paper [Wi15] that we presented at ICSE this year, we experimentally evaluated this assumption. We executed SFI tests on the Android OS kernel by injecting faults into the SD card driver. To contain the effects of fault activations during these tests, the system was executed in an emulator that was reset after each test. We repeated the tests with varying degrees of concurrency by instantiating varying numbers of emulator instances. To assess, whether concurrency has an effect on the experiment outcome, we compared the result distributions for the varying degrees of concurrency. Our initial results showed significant deviations for higher degrees of concurrency, indicating that an unreflected replication of SUT instances threatens the validity of test results. We identified the SUT instances' competition for shared system resources and the resulting execution latency increases, which directly affected some of

[1] Technische Universität Darmstadt, DEEDS Group, Hochschulstr. 10, 64289 Darmstadt, Germany, {sw | os | suri}@cs.tu-darmstadt.de
[2] Federico II University of Naples, DIETI, via Claudio 21, 80125 Naples, Italy, {roberto.natella | cotroneo}@unina.it

the employed test oracles, as the root cause for the observed deviations. We then devised a pre-test measurement approach to adjust these oracles for the concurrent execution of a given number of SUT instances on a given test machine. Using this approach, we were able to execute up to 44 SFI tests concurrently without any significant test result deviations on a machine with 16 CPU cores and 64 GiB main memory. The highest throughput for this configuration was 157 experiments per hour with 36 concurrent instances, a more than 12-fold throughput increase compared to sequential test execution. For this configuration we also observed the lowest correlation between the degree of concurrency and the test result distribution in a χ^2 test for independence, which indicates that the initially observed result deviations were indeed caused by performance interference of concurrent test executions.

Besides the direct impact of our result on SFI and other robustness testing approaches, where performance sensitive oracles are used to detect so-called hang failures, our result indicates that test parallelization requires careful analysis to obtain valid results *whenever tests rely on execution latencies*. For example, any JUnit tests that use the `timeout` parameter or `Timeout` rule would be similarly affected. An interesting observation from our experiments was that the SUT initialization contributed significantly to the observed test latencies. This is not surprising, as we used heavy-weight isolation measures to make test executions as independent as possible from each other by running them in (almost, as our results show) completely isolated environments. This opens up the possibility for a trade-off: Performance interference decreases with less isolation, which on the other hand increases the risk for other types of test interference [Zh14].

While our pre-test measurement approach proved effective for time-dependent oracle adjustment, it required the execution of around 800 tests for reliable predictions. Our goal is to reduce this calibration overhead and, ideally, devise an analytical model for accurate predictions of *safe* time-dependent oracles and achievable concurrency degrees for a given test type and test machine configuration, that do not even require additional test executions for calibration. To achieve this, we need to better understand the root causes behind latency increases. We hope the related research to also shed some light on the factors that caused throughput to degrade when more than 36 concurrent SUT replica were instantiated in our experiments and to guide hardware and scheduler configuration for better test throughput.

Acknowledgments: This research has been supported in part by DFG GRK 1362, CASED, EC-SPRIDE, EC H2020 #644579, CECRIS FP7 (GA no. 324334), and SVEVIA MIUR (PON02 00485 3487758).

References

[Wi15] Winter, Stefan; Schwahn, Oliver; Natella, Roberto; Suri, Neeraj; Cotroneo, Domenico: No PAIN, No Gain?: The Utility of PArallel Fault INjections. In: Proceedings of the 37th International Conference on Software Engineering - Volume 1. ICSE '15, IEEE Press, Piscataway, NJ, USA, pp. 494–505, 2015.

[Zh14] Zhang, Sai; Jalali, Darioush; Wuttke, Jochen; Muşlu, Kivanç; Lam, Wing; Ernst, Michael D.; Notkin, David: Empirically Revisiting the Test Independence Assumption. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis. ISSTA 2014, ACM, New York, NY, USA, pp. 385–396, 2014.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 47

Analysis of the Trade-offs in Different Modeling Approaches for Performance Prediction of Software Systems

Samuel Kounev[1], Fabian Brosig[2], Philipp Meier[3], Steffen Becker[4], Anne Koziolek[5], Heiko Koziolek[6], and Piotr Rygielski[1]

Abstract: A number of performance modeling approaches for predicting the performance of modern software systems and IT infrastructures exist in the literature. Different approaches differ in their modeling expressiveness and accuracy, on the one hand, and their modeling overhead and costs, on the other hand. Considering a representative set of established approaches, we analyze the semantic gaps between them as well as the trade-offs in using them; we further provide guidelines for selecting the right approach suitable for a given scenario.

Keywords: Modeling, performance prediction, software systems, model transformation

During the last decade, researchers have proposed a number of modeling approaches and respective model-to-model transformations enabling performance prediction of software systems, including their computing, storage and network infrastructures [GMS07]. These transformations map performance-annotated software architecture models into stochastic models solved by analytical means or by simulation. However, so far, a detailed quantitative evaluation of the accuracy and efficiency of different modeling approaches and solution techniques is missing, making it hard to select an adequate transformation for a given context [Ba04].

Approaches based on numerical solvers are known to be fast but often limited in expressiveness to adequately model many realistic situations. Approaches based on simulation are known to be more expressive but often have long execution times leading to high prediction overhead [BHK14]. The intuitively perceived trade-offs between prediction accuracy and solution efficiency in state-of-the-art performance analysis tools are currently not well understood due to the lack of in-depth quantitative evaluations and comparisons. Trade-off decisions between prediction accuracy and time-to-result are important in scenarios where: (a) a large problem space needs to be explored (e.g., scaling of complex cloud applications [Sp15]) or (b) when the prediction results need to be available within a certain time window (e.g., in trigger-based reactive cloud scaling scenarios [HKR13]).

[1] Department of Computer Science, University of Würzburg Am Hubland, 97074 Würzburg. E-mail: {samuel.kounev, piotr.rygielski}@uni-wuerzburg.de
[2] MiNODES GmbH, Friedrichstrae 224, 10969 Berlin, German. E-mail: fabian.brosig@minodes.com
[3] codecentric AG, Elsenheimerstr. 55a, 80687 München, Germany. E-mail: philstyler@googlemail.com
[4] TU Cheminitz, Straße der Nationen 62, 09111 Chemnitz, Germany. E-mail: steffen.becker@tu-chemnitz.de
[5] Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany. E-mail: koziolek@kit.edu
[6] ABB Corp. Research, Wallstadter Str. 59, 68526 Ladenburg, Germany. E-mail: heiko.koziolek@de.abb.com

We provide an in-depth comparison and quantitative evaluation of the trade-offs in different model transformations for performance evaluation of software systems and IT infrastructures. The semantic gaps between typical source model abstractions and the different performance analysis techniques are examined in detail. The accuracy and efficiency of each transformation are evaluated by considering several case studies representing systems of different size and complexity.

The presented results and insights gained from the evaluation help software architects and performance engineers to select the appropriate transformation for a given context, thus significantly improving the usability of model transformations for performance prediction.

We provide an overview of the results of a recent paper published in [Br15], as well as some follow-up work at SIMUTools 2015 [RKTG15] focussing on data center networks [RKZ13]. For networks, similarly to software systems, there exist multiple performance modeling and solution approaches, so similar trade-offs exist in the selection of a suitable approach for a given scenario.

References

[Ba04] Balsamo, Simonetta; Di Marco, Antinisca; Inverardi, Paola; Simeoni, Marta: Model-Based Performance Prediction in Software Development: A Survey. IEEE Trans. on Software Engineering, 30(5), May 2004.

[BHK14] Brosig, Fabian; Huber, Nikolaus; Kounev, Samuel: Architecture-Level Software Performance Abstractions for Online Performance Prediction. Elsevier Science of Computer Programming Journal (SciCo), Vol. 90, Part B:71–92, September 2014.

[Br15] Brosig, Fabian; Meier, Philipp; Becker, Steffen; Koziolek, Anne; Koziolek, Heiko; Kounev, Samuel: Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-based Architectures. IEEE Transactions on Software Engineering (TSE), 41(2):157–175, February 2015.

[GMS07] Grassi, Vincenzo; Mirandola, Raffaela; Sabetta, Antonino: Filling the Gap Between Design and Performance/Reliability Models of Component-based Systems. J. Syst. Softw., 80(4):528–558, April 2007.

[HKR13] Herbst, Nikolas Roman; Kounev, Samuel; Reussner, Ralf: Elasticity in Cloud Computing: What it is, and What it is Not. In: Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013). USENIX, June 2013.

[RKTG15] Rygielski, Piotr; Kounev, Samuel; Tran-Gia, Phuoc: Flexible Performance Prediction of Data Center Networks using Automatically Generated Simulation Models. In: Proceedings of the Eighth EAI International Conference on Simulation Tools and Techniques (SIMUTools 2015). August 2015.

[RKZ13] Rygielski, Piotr; Kounev, Samuel; Zschaler, Steffen: Model-Based Throughput Prediction in Data Center Networks. In: Proceedings of the 2nd IEEE International Workshop on Measurements and Networking (M&N 2013). pp. 167–172, October 2013.

[Sp15] Spinner, Simon; Herbst, Nikolas; Kounev, Samuel; Zhu, Xiaoyun; Lu, Lei; Uysal, Mustafa; Griffith, Rean: Proactive Memory Scaling of Virtualized Applications. In: Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing (IEEE CLOUD 2015). IEEE, pp. 277–284, June 2015.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 49

Feedback Generation for Performance Problems in Introductory Programming Assignments

Sumit Gulwani[1] Ivan Radiček and Florian Zuleger[2]

Abstract: Providing feedback on programming assignments manually is a tedious, error prone, and time-consuming task. In [GRZ14], we motivate and address the problem of generating feedback on performance aspects in introductory programming assignments. We studied a large number of functionally correct student solutions to introductory programming assignments and observed: (1) There are different algorithmic strategies, with varying levels of efficiency, for solving a given problem. These different strategies merit different feedback. (2) The same algorithmic strategy can be implemented in countless different ways, which are not relevant for reporting feedback on the student program.

We propose a light-weight programming language extension that allows a teacher to define an algorithmic strategy by specifying certain key values that should occur during the execution of an implementation. We describe a dynamic analysis based approach to test whether a student's program matches a teacher's specification. Our experimental results illustrate the effectiveness of both our specification language and our dynamic analysis.

Keywords: Education, MOOCs, performance analysis, trace specification, dynamic analysis.

Providing feedback on programming assignments is a very tedious, error-prone, and time-consuming task for a human teacher, even in a standard classroom setting. With the rise of Massive Open Online Courses (MOOCs), which have a much larger number of students, this challenge is even more pressing. Hence, there is a need to introduce automation around this task. Immediate feedback generation through automation can also enable new pedagogical benefits such as allowing resubmission opportunity to students who submit imperfect solutions and providing immediate diagnosis on class performance to a teacher who can then adapt her instruction accordingly.

Recent research around automation of feedback generation for programming problems has focused on guiding students to functionally correct programs either by providing counterexamples (generated using test input generation tools) or generating repairs. However, non-functional aspects of a program, especially performance, are also important. We studied several programming sessions of students who submitted solutions to introductory C# programming problems on the PEX4FUN[3] platform. In such a programming session, a student submits a solution to a specified programming problem and receives a counterexample based feedback upon submitting a functionally incorrect attempt. The student may then inspect the counterexample and submit a revised attempt. This process is repeated

[1] Microsoft Research, USA
[2] TU Wien, Arbeitsbereich Formal Methods in Systems Design, Institut für Informationssysteme 184/4, Favoritenstraße 9–11, 1040 Wien, Austria
[3] http://www.pexforfun.com/

until the student submits a functionally correct attempt or gives up. We studied 24 different problems, and observed that of the 3993 different programming sessions, 3048 led to functionally correct solutions. However, unfortunately, on average around 60% of these functionally correct solutions had (different kinds of) performance problems. In this paper, we present a methodology for semi-automatically generating appropriate performance related feedback for such functionally correct solutions.

From our study, we made two observations that form the basis of our semi-automatic feedback generation methodology. (i) There are different *algorithmic strategies* with varying levels of efficiency, for solving a given problem. Algorithmic strategies capture the global high-level insight of a solution to a programming problem, while also defining key performance characteristics of the solution. Different strategies merit different feedback. (ii) The same algorithmic strategy can be implemented in countless different ways. These differences originate from local low-level implementation choices and are not relevant for reporting feedback on the student program.

In order to provide meaningful feedback to a student it is important to identify what algorithmic strategy was employed by the student program. A profiling based approach that measures running time of a program or use of static bound analysis techniques is not sufficient for our purpose, because different algorithmic strategies that necessitate different feedback may have the same computational complexity. Also, a simple pattern matching based approach is not sufficient because the same algorithmic strategy can have syntactically different implementations.

Our key insight is that the algorithmic strategy employed by a program can be identified by observing the values computed during the execution of the program. We allow the teacher to specify an algorithmic strategy by simply annotating (at the source code level) certain key values computed by a sample program (that implements the corresponding algorithm strategy) using a new language construct, called *observe*. Fortunately, the number of different algorithmic strategies for introductory programming problems is often small (at most 7 per problem in our experiments). These can be easily enumerated by the teacher in an iterative process by examining any student program that does not match any existing algorithmic strategy.

We propose a novel dynamic analysis that decides whether the student program (also referred to as an *implementation*) matches an algorithm strategy specified by the teacher in the form of an annotated program (also referred to as a *specification*). Our dynamic analysis executes a student's implementation and the teacher's specification to check whether the key values computed by the specification also occur in the corresponding traces generated from the implementation.

References

[GRZ14] Gulwani, Sumit; Radicek, Ivan; Zuleger, Florian: Feedback generation for performance problems in introductory programming assignments. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014. pp. 41–51, 2014.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 51

Integrated Performance Simulation of Business Processes and Information Systems

Robert Heinrich[1] Philipp Merkle[1] Jörg Henss[2] Barbara Paech[3]

Abstract: Business processes (BPs) and information systems (ISs) mutually affect each other in non-trivial ways. Quality issues may arise from missing alignment. Although simulation is a powerful approach to predict the performance of BPs and ISs, current approaches lack their integration in simulation. We propose a holistic performance prediction approach to adequately reflect the mutual impact between BPs and ISs in simulation. Applying the approach and tooling in a real-life case study showed its feasibility and practicability.

Keywords: Business Process, Information System, Alignment, Performance

1 Mutual Quality Impact between Business Processes and Information Systems

Business process (BP) designs and enterprise information system (IS) designs are often not well aligned. Missing alignment may result in quality issues at run-time, such as large process execution time or overloaded IS resources. The complex interrelations between BPs and ISs are neither adequately researched so far nor sufficiently considered in development or operation. Especially interrelations between quality aspects (such as performance, reliability, security, or maintainability) concerned with BP designers and those concerned with IS developers are not well understood. Frequently, a direct mapping of metrics is difficult as the representation of a certain quality aspect may differ in the BP and IS domain.

Engineering methods for aligning one domain to the quality objectives of another are missing. One major reason for insufficient quality engineering is that current approaches lack an integrated consideration of quality aspects among several domains. Frequently, BPs and ISs are not well aligned, meaning that BPs are designed without taking IS impact into account and vice versa [He14]. Neglecting the mutual impact between BPs and ISs leads to serious issues, e.g. unsatisfied requirements, unreliable decisions, deceleration and rework.

Simulation is a promising approach to predict performance of both, BP and IS designs. Based on prediction results, design alternatives may be compared and verified against requirements. Thus, BP and IS designs can be aligned to improve performance. Yet, BP simulation and IS simulation are not adequately integrated in current simulation approaches. This results in limited prediction accuracy due to neglected interrelations between the BP and the IS in simulation.

[1] Karlsruhe Institute of Technology, {robert.heinrich, philipp.merkle}@kit.edu
[2] FZI Forschungszentrum Informatik, henss@fzi.de
[3] Heidelberg University, paech@informatik.uni-heidelberg.de

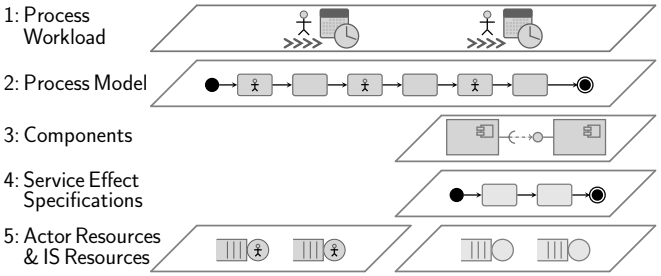


Abb. 1: Overview of the IntBIIS Simulation Layers [He15]

2 Integrated Business IT Impact Simulation (IntBIIS)

The holistic approach IntBIIS [He15] combines performance prediction on software architecture level and business process level to adequately reflect the mutual impact between BPs and ISs in simulation. Based on a quality reference model [He14] for BPs, IntBIIS models and analyzes the mutual performance impact between BPs and ISs building upon the Palladio approach [BKR09]. While Palladio provides adequate means for analyzing IS architectures, IntBIIS extends Palladio by modeling constructs and simulation behavior to analyze BPs and their organizational environment. In this way, the alignment of BP designs and IS designs can be supported by comparing the predicted performance impact of design alternatives and verifying them against requirements.

Fig. 1 illustrates IntBIIS where elements with a stickman symbol indicate layers and elements introduced as a result of our work. The remaining layers and elements are taken from the Palladio reference simulator. A run of the integrated simulation starts at the topmost layer with simulating time-variant workloads. The workloads trigger the traversing of an action chain of actor steps and system steps specified in the BP model (layer 2). For actor steps a suitable actor resource is requested (layer 5, left) to process the step according to a predefined scheduling policy. For system steps resource demands are not issued directly, but emerge as the system request propagates through software components (layer 3), their service effect specifications (layer 4), down to hardware resources (layer 5, right).

We evaluated the feasibility and practicability of our approach and tooling by modeling and simulating a BP and involved ISs in a real-life case study. Comparing our simulation output to reference values measured in reality and prediction results of another BP simulation tool indicated that IntBIIS yields accurate simulation results. In experiments we examined the scalability of IntBIIS and showed its ability to handle long and complex simulation runs.

References

[BKR09] Becker, Steffen; Koziolek, Heiko; Reussner, Ralf: The Palladio component model for model-driven performance prediction. Journal of Systems and Software, 82:3–22, 2009.

[He14] Heinrich, Robert: Aligning Business Processes and Information Systems: New Approaches to Continuous Quality Engineering. Springer, 2014. ISBN: 978-3-658-06517-1.

[He15] Heinrich, Robert; Merkle, Philipp; Henss, Jörg; Paech, Barbara: Integrating Business Process Simulation and Information System Simulation for Performance Prediction. Intl. Journal on Software & Systems Modeling, 2015. DOI: 10.1007/s10270-015-0457-1.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 53

From Aristotle to Ringelmann: A large-scale analysis of team productivity and coordination in Open Source Software projects

Ingo Scholtes[1], Pavlin Mavrodiev[2], Frank Schweitzer[3]

Abstract: The productivity of software development teams, i.e., how their size relates to their output, is an important question for project management. Most studies suggest that teams become less productive as they grow larger, a phenomenon paraphrased as *Brooks' law* in software engineering and as *Ringelmann effect* in social psychology. Conversely, a recent study suggests that the productivity of teams in OSS projects *increases* as they grow larger. Attributing it to synergetic effects, this was linked to the Aristotelian quote that "the whole is more than the sum of its parts". Using data on 58 OSS projects with 580,000 commits by 30,000 developers, we perform a large-scale analysis of productivity in development teams. We confirm the negative relation previously found by software engineering research, providing quantitative evidence for the Ringelmann effect. Taking a network perspective on developer-code associations, we investigate mechanism behind this effect and show that the magnitude of the productivity decrease is related to the growth dynamics of coordination networks.

Most of today's software projects are so complex that they cannot be developed by a single person, instead requiring large teams of collaborating developers. This necessity of large teams raises a simple, yet important question: How productive is a team of developers compared to a single developer? Or, in other words: How much time do n developers need to finish a project compared to the time taken by a single developer? This question is of significant importance not only for project management but also for the development of cost estimation models for software engineering processes. One may naively assume that the productivity of individual team members is *additive*, i.e., that, compared to the time taken by a single developer, n developers will speed up the development time by a factor of n. However, this misses out two important factors that can give rise to a non-additive scaling of productivity. First, the collaboration of developers in a team can give rise to *synergy effects*, which result in the team being *more productive* than one would expect from adding up individual productivities of its members. Under this assumption, the average output per team member can be *increased* by adding developers to the team, a fact that has recently been related to Aristotle's quote that *"the whole is more than the sum of its parts"* [SMG14]. A second, contrary factor that influences the productivity of developer teams is the communication and coordination overhead which is likely to increase as teams grow larger. In particular, this can lead to situations where the average output per team member *decreases* as the size of the team is increased. Studies showing that growing team sizes negatively affect productivity can be traced back to early studies of Maximilian Ringelmann [Ri13]. In the context of software engineering, it can be related

[1] Chair of Systems Design, ETH Zürich, CH-8092 Zürich, Switzerland, ischoltes@ethz.ch
[2] Chair of Systems Design, ETH Zürich, CH-8092 Zürich, Switzerland, ischoltes@ethz.ch
[3] Chair of Systems Design, ETH Zürich, CH-8092 Zürich, Switzerland, ischoltes@ethz.ch

to "Brook's" law of software project management, which states that *"adding manpower to a late software project makes it later"* [Jr75].

Using a data set covering the history of 58 Open Source Software (OSS) projects hosted on the social coding platform GITHUB, in [SMS15] we quantitatively address the question how the size of a software development team is related to their productivity. Based on a time-slice analysis of more than 580,000 commit events over a period of more than 14 years, we analyse the output of projects in terms of code and study how their time-varying productivity relates to the number of active software developers. Using the *distribution of inter-commit times*, we first identify reasonable time windows for the definition of team size and the analysis of commit activities in OSS projects. We measure the contributions of individual commits based on a microscopic, textual analysis of commit contents. Our analysis confirms the intuition that the actual contribution of commits exhibits a large variation, thus requiring an analysis of commit contents rather than the mere number of commits. We define a measure for the contribution of developers which is based on the *Levenshtein edit distance* [Le66] between consecutive versions of source code files. Using this fine-grained measure, we quantitatively show that in all of the studied OSS projects the average productivity of developers decreases as the team size increases, thus providing quantitative evidence for the Ringelmann effect. Finally, we take a network perspective on the association between developers and the source code files they have edited. Aiming at a file-based and language-independent first-order approximation for coordination structures, we analyse the growth dynamics of co-editing networks constructed from repository data. For all projects in our data set, we observe a *super-linear* growth of co-editing networks, which can be seen as one potential mechanism behind the observed Ringelmann effect. We argue that both our results as well as our methodology are useful to refine and calibrate existing software development cost models based on empirical data from software development repositories.

References

[Jr75] Jr., Frederick P. Brooks: The Mythical Man-Month. Addison-Wesley, 1975.

[Le66] Levenshtein, Vladimir I: Binary codes capable of correcting deletions, insertions and reversals. In: Soviet physics doklady. volume 10, p. 707, 1966.

[Ri13] Ringelmann, Maximilan: Recherches sur les moteurs animes: Travail de l'homme. Annales de l'Institut National Agronomique, 12(1):1–40, 1913.

[SMG14] Sornette, Didier; Maillart, Thomas; Ghezzi, Giacomo: How Much Is the Whole Really More than the Sum of Its Parts? 1 + 1 = 2.5: Superlinear Productivity in Collective Group Actions. PLoS ONE, 9(8):e103023, 08 2014.

[SMS15] Scholtes, Ingo; Mavrodiev, Pavlin; Schweitzer, Frank: From Aristotle to Ringelmann: A large-scale analysis of team productivity and coordination in Open Source Software projects. Empirical Software Engineering, 2015. accepted for publication on September 23 2015, to appear.

Software Process Improvement: Where Is the Evidence?

Marco Kuhrmann[1], Claudia Konopka[2], Peter Nellemann[1], Philipp Diebold[3] and Jürgen Münch[4]

Abstract: Software process improvement (SPI) is around for decades: frameworks are proposed, success factors are studied, and experiences have been reported. However, the sheer mass of concepts, approaches, and standards published over the years overwhelms practitioners as well as researchers. What is out there? Are there new emerging approaches? What are open issues? Still, we struggle to answer the question for what is the current state of SPI and related research? We present initial results from a systematic mapping study to shed light on the field of SPI and to draw conclusions for future research directions. An analysis of 635 publications draws a big picture of SPI-related research of the past 25 years. Our study shows a high number of solution proposals, experience reports, and secondary studies, but only few theories. In particular, standard SPI models are analyzed and evaluated for applicability, especially from the perspective of SPI in small-to-medium-sized companies, which leads to new specialized frameworks. Furthermore, we find a growing interest in success factors to aid companies in conducting SPI.

This summary refers to the paper *Software Process Improvement: Where Is the Evidence?* [Ku15]. This paper was published as full research paper in the *ICSSP'2015* proceedings.

Keywords: software process, software process improvement, systematic mapping study

1 Introduction

Software process improvement (SPI) aims to improve software processes and comprises a variety of tasks, such as scoping, assessment, design and realization, and continuous improvement. Several SPI models compete for the companies' favor, success factors to support SPI implementation at the large and the small scale are studied, and numerous publications report on experiences in academia and practice. SPI is considered an important topic. However, SPI is a diverse field: On the one hand, a number of standards is available, e.g., ISO/IEC 15504 or CMMI but, on the other hand, these standards are criticized often [St07]. In a nutshell, the different facets of SPI and the corresponding research provide a huge body of knowledge on SPI.

Problem. The field of SPI evolved for decades and provides a vast amount of publications addressing a huge variety of topics. Still, we see new method proposals, research on success factors, and experience reports. However, missing is a big picture

[1] University of Southern Denmark, Campusvej 55, 5230 Odense, Denmark, {kuhrmann,pnel}@mmmi.sdu.dk
[2] 4Soft GmbH, Mittererstr. 3 80336 Munich, Germany, claudia.konopka@4soft.de
[3] Fraunhofer IESE, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany, philipp.diebold@iese.fhg.de
[4] University of Helsinki, Department of Computer Science, Gustaf Hällströmin katu 2b, FI-00014 Helsinki, Finnland, juergen.muench@cs.helsinki.fi

that illustrates where SPI gained a certain level of saturation and where are still hot topics and unresolved issues calling for more investigation.

Objective, Method, and Contribution. To better understand the state of the art in SPI, we aim to analyze the whole publication flora to draw a big picture on SPI. As research method, we opted for a combination of the well-known *Systematic Literature Review* and *Mapping Study* instruments. We contribute initial findings from a comprehensive literature study in which we analyze 635 papers from 25 years of SPI-related research.

2 Results

In total, in our study, we obtained 635 papers on SPI published between 1989 and 2013. Most papers (≈2/3) were categorized as solution proposal (n=244) or philosophical paper (n=214). However, the result set also contains a number of evaluation research (n=102) and experience papers (n=70) showing the field of SPI still moving. The classification shows that lessons learned (n=290, 46%) and frameworks (n=235, 37%) make the majority of the contributions. Other categories are barely represented, e.g., tools (n=36), models (n=24), and theories (n=12). Most of the solution proposals focus on frameworks (n=167), i.e., 26% of all papers propose a new SPI framework. The largest share of the philosophical papers is devoted to lessons learned (n=155, i.e., 24%). Yet, the result set also points to some new trends, e.g., SPI in the context of agile software development and in the context of small-to-medium-sized companies.

3 Conclusion

The field of SPI suffers from missing evidence: Proposed solutions are barely evaluated for their feasibility, studies comparing and analyzing proposed solutions for their advantages and disadvantages are missing, and testable theories are—if at all—in the construction phase awaiting confirmation. Furthermore, our study reveals some trends in SPI-related research: We found growing interest over the recent years in SPI for SME's and adopting agile principles for SPI. Also, we found an increasing number of secondary studies of which some already started to collect, structure, and generalize knowledge.

References

[Ku15] Kuhrmann, K.; Konopka, C.; Nellemann, P.; Diebold, P.; Münch, J.: Software Process Improvement: Where is the Evidence? Proc. of Int. Conf. on Software and System Process, ACM, New York, NY, pp. 107-116, 2015.

[St07] Staples, M.; Niazi, M.; Jeffery, R.; Abrahams, A.; Byatt, P.; Murphy, R.: An exploratory study of why organizations do not adopt CMMI. Journal of Systems and Software, 80(6):883–895, 2007.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 57

Cost-Effective Evolution of Research Prototypes into End-User Tools: The MACH case study

Harald Störrle[1]

Abstract: Much of Software Engineering research hinges on an implementation as a proof-of-concept. The resulting tools are often quite prototypical, to the degree of having little practical benefit. The Model Analyzer/Checker (MACH) is a case study in turning a set of research prototypes for analyzing UML models into a tool that can be used in research and teaching, by a broad audience. We document how the requirements and constraints of an academic environment influence design decisions in software tool development. We argue that our approach while perhaps unconventional, serves its purpose with a remarkable cost-benefit ratio. This paper is a summary of [St15].

Background For the purposes of research and publication, there is little value in polishing the usability, stability, portability, extensibility, or performance of a tool, as there is only ever one user of the tool. However, this excludes independent replication, benchmarking, human-factors studies (e.g., observing real users actually using the tool), tool usage in teaching settings, or commercial dissemination. Turning a research prototype into a "proper" tool, however, implies substantial engineering effort with little contribution to the research being conducted. Delegating the task to a student as a thesis project may not be possible, or may prove not successful. All too often, the researcher ends up abandoning the further dissemination of a strand of research for lack of resources.

Objective In this particular case, the author's research work focuses on advanced operations on UML models that are beyond the scope of existing modeling tools. Over the years, many small exploratory prototypes have been created, each of which requires a high degree of expertise to use, and presents only a negligible contribution to a modeler. Thus, the objective of MACH was to create a single integrated tool from a set of prototypes to realize synergies, encapsulate it with a user interface to make it accessible to a wide range of users, and, most of all, do all this with as little cost as possible.

Method We focus on academic stakeholders, thus justifying the assumptions that (1) MACH users have some understanding of models and the underlying concepts, (2) they have sufficient motivation to use the tool even if its user interface is less polished than commercial end user software.

Driven by the main rationale of rapid prototyping, most of the prototypes mentioned above have been implemented using the PROLOG programming language. Long-term usage of the resulting code, usage by third parties, or long-term-evolution of the code base was not considered at the time of creation. Thus, creating MACH tried to achieve three goals: (1) Combine most or all of the existing prototypes into a single tool; (2) make the major functions available to students and colleagues; but (3) strictly limit the effort in creating

[1] Department of Applied Mathematics and Computer Science,Technical University of Denmark,hsto@dtu.dk

the tool to the bare minimum. We mapped these goals into ten requirements, designed a solution, and evaluated the tool repeatedly in various classes taught by the author.

Result MACH is implemented in PROLOG and provides a command-line user interface, both of which are relatively exotic choices, today. MACH can be obtained online at `www.compute.dtu.dk/~hsto`. There is an online demonstration of MACH 0.93 available via the SHARE platform at [St14], including samples and a manual. No installation is required. MACH uses a set of utility functions and a common file format glued together the following advanced analysis and checking procedures on (UML) models.

- **Clone Detection** important task in the quality assurance of models, where we often find duplicate model fragments, arising through thoughtless "copy-paste modeling".
- **Model Version Control** is as effective as the weakest link in the chain, typically the way differences are presented to human modelers. We have proposed a novel approach and tested it in controlled experiments, but lacked in vivo user studies to provide more reliable evidence.
- **Model size and similarity metrics** are necessary to assess the size and nature of a model. It is straightforward to use the frequency distribution of model element types, and visualize it as a histogram.

In addition to the features described above, MACH provides a number of supportive functions that are essential for doing practical work, such as navigating the directory tree, loading models, handling aliases, command history, file name completion, a help system, and so on. Usage experience so far suggests that our approach serves its purpose.

Conclusions By sticking to an "exotic" high-level language like PROLOG, offering only a simple text interface with ASCII-graphics, and abandoning the idea of a tight, high-productivity integration into an existing tool framework like Eclipse RCP, we can achieve the core goals of deploying research results quickly to a broader audience, while keeping the required effort at the absolute minimum. The approach may be unconventional, but it offers a unique cost-benefit ratio.

We believe this is a viable path for other researchers faced with similar challenges. We hope that our experience will help others make their research prototypes available to larger audiences with reasonable effort, for the benefit of the entire scientific community.

References

[St14] Störrle, Harald: , MACH 0.93. online, 2014. `http://is.ieis.tue.nl/staff/pvgorp/share/?page=ConfigureNewSession&vdi=XP-TUe_XP_SWIPL.vdi`.

[St15] Störrle, Harald: Cost-Effective Evolution of Research Prototypes into End-User Tools: The MACH case study. Science of Computer Programming, 2015. in print.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 59

Getting to Know You: Towards a Capability Model for Java

Ben Hermann[1] Michael Reif[2] Michael Eichberg[3] and Mira Mezini[4]

Abstract: Developing software from reusable libraries lets developers face a security dilemma: Either be efficient and reuse libraries as they are or inspect them, know about their resource usage, but possibly miss deadlines as reviews are a time consuming process. In this paper, we propose a novel capability inference mechanism for libraries written in Java. It uses a coarse-grained capability model for system resources that can be presented to developers. We found that the capability inference agrees by 86.81% on expectations towards capabilities that can be derived from project documentation. Moreover, our approach can find capabilities that cannot be discovered using project documentation. It is thus a helpful tool for developers mitigating the aforementioned dilemma.

1 Summary

The efficiency of software development largely depends on an ecosystem of reuse [Bo99, Gr93]. Numerous software libraries are available that solve various problems ranging from numerical computations to user interface creation. The safe use of these libraries is an exigence for the development of software that meets critical time-to-market constraints.

However, when including software libraries into their products software developers entrust the code in these libraries with the same security context as the application itself regardless of the need for this excessive endorsement. For instance, a system that makes use of a library of numerical functions also enables the library to use the filesystem or make network connections although the library does not need these capabilities. If the library contains malicious code it could make use of them. In commonly used languages like Java no effective mechanism to limit or isolate software libraries from the application code exists. So developers face a dilemma: Either trust the component and finish the project in time or be secure, review the library's source code and possibly miss deadlines.

We propose to consider this excessive assignment of authority as a violation of the *Principle of Least Privilege [SS75]*. The principle states that every program should operate under the least set of privilege necessary to complete its job. In order to alleviate the described dilemma, we introduce an effective mechanism in this paper to detect the actual permission need of software libraries written in Java.

Drawing inspiration from Android, we construct a capability model for Java. It includes basic, coarse-grained capabilities such as the authority to access the filesystem or to open a

[1] Technische Universität Darmstadt, Fachbereich Informatik Fachgebiet Softwaretechnik, Hochschulstraße 10, 64289 Darmstadt, hermann@cs.tu-darmstadt.de
[2] reif@cs.tu-darmstadt.de
[3] eichberg@cs.tu-darmstadt.de
[4] mezini@cs.tu-darmstadt.de

network socket. As Java programs by themselves cannot communicate with the operating system directly, any interaction with those capabilities has to happen through the use of the Java Native Interface (JNI). By tracking the calls backwards through the call graph, we produce a capability set for every method of the Java Class Library (JCL) and by the same mechanism towards methods of a library. We can thus effectively infer the necessary capabilities of a library using our approach. We can also infer the subset of these capabilities used by an application, as it may not use every functionality supplied by the library.

As the precision of our approach is directly depending on the precision of the algorithm used to calculate the call graph of the library, we took several measures to compute a reasonably precise call graph while not compromising the scalability of the algorithm too severely. We evaluated our approach by comparing our results against expectations derived from API documentation. We found that for 70 projects from the Qualitas Corpus [Te10], that we evaluated against, actual results exceeded expectations and produce a far more accurate footprint of the projects capability usage. Thereby, our approach helps developers to quickly make informed decisions on library reuse without the need for manual inspection of source code or documentation.

In our pursuit to mitigate the software developer's dilemma w.r.t. library reuse, we thus contribute the following in our paper:

- an algorithm to propagate capability labels backwards through a call graph,
- a labeling of native methods with their necessary capabilities to bootstrap the process,
- a collection of efficient analysis steps to aid the precision of common call-graph algorithms,
- an evaluation of the algorithm against extracted capability expectations from documentation.

We provide the implementation and all related data of our approach here:
`http://www.thewhitespace.de/projects/peaks/capmodel.html`

References

[Bo99] Boehm, Barry W: Managing Software Productivity and Reuse. IEEE Computer, 32(9):111–113, 1999.

[Gr93] Griss, Martin L: Software Reuse: From Library to Factory. IBM Systems Journal, 32(4):548–566, 1993.

[SS75] Saltzer, J.H.; Schroeder, M.D.: The protection of information in computer systems. Proceedings of the IEEE, 63(9):1278–1308, Sept 1975.

[Te10] Tempero, Ewan; Anslow, Craig; Dietrich, Jens; Han, Ted; Li, Jing; Lumpe, Markus; Melton, Hayden; Noble, James: Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In: 2010 Asia Pacific Software Engineering Conference (APSEC2010). pp. 336–345, December 2010.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 61

Copy-Paste Redeemed [1]

Krishna Narasimhan[2] Christoph Reichenbach[3]

Abstract: Software evolves continuously. As software evolves, it's code bases require implementations of new features. These new functionalities are sometimes mere extensions of existing functionalities with minor changes. A commonly used method of extending an existing feature into a similar new feature is to copy the existing feature and modify it. This method of extending feature is called "Copy-paste-modify". Another method of achieving the same goal of extending existing feature into similar feature is abstracting the multiple similar features into one common feature with appropriate selectors that enable choosing between the features. The advantages of the "Copy-paste-modify" technique range from speed of development to reduced possibility of breaking existing feature. The advantages of abstraction vary from user preference to have abstracted code to long term maintenance benefits. In our paper, we describe an informal poll and discuss related work to confirm our beliefs about the advantages of each method of extending features. We observe a potential compromise while developers extend features which are near-clones of existing features. We propose to address this dilemma by coming up with a novel approach that can semi-automatically abstract near-clone features and evaluate our approach by building a prototype in C++ and abstracting near-clone methods in popular open source repositories.

Keywords: Refactoring, Software clones, Static analysis, Software evolution, Abstraction

1 Introduction

Programmers frequently employ copy-paste-modify as a method of implementing extensions to features. Although copy-paste-modify yields quicker results with minimal damage to existing code, it results in bloated code space with redundant code. This is a headache for maintenance as readability is reduced and bug fixing is tedious is a bug in the initial near-clone is propagated to the copy pasted extensions. On the abstraction, provides maintenance friendly code occupying less code space. But, manual abstraction is hard. We propose to resolve this discrepancy with a novel approach that can abstract features from near-clones, thereby allowing developers to quickly extend features by employing copy-paste as a method of extending features and invoking a refactoring which will provide the best possible abstraction.

2 Informal Poll

We conducted an informal poll with five programmers of varying experience with C++ programming ranging from 2 months to 10 years in order to determine which method of

[1] A summary of the publication by the same name in ASE 2015

[2] Goethe Universitït, Informatik, Robert Mayer Strasse 10, 60486 Frankfurt, krishna.nm86@gmail.com
[3] Goethe Universitït, Informatik, Robert Mayer Strasse 10, 60486 Frankfurt, reichenbach@em.uni-frankfurt.de

extending features was easier to develop and which method was preferred for use. For the initial study, we collected 5 near-clone function pairs from popular open source repositories, removed one of the functions and asked the programmers to implement the remaining feature using copy-paste(for one group) and abstraction(for another group). We measured the time taken and observed that **Users find copy paste quicker**. We followed the initial study with a survey on the user preference in the same issue and found out that **Users prefer abstracted versions of code to maintain and use**.

3 Merging Algorithm

The merging algorithm takes as input abstract syntax trees of function definitions and

```
void function1(){
  b(c,d);
  y = f1;
  x(z);
}
```

```
void function2(){
  b(c,e);
  y = f2;
  x(z);
}
```

```
void function3(){
  b2();
  n();
  y = f3;
  x(z);
}
```

```
void fnMerged(int functionId, int fValue, int bParam)
{
  if(functionId == 1 || functionId == 2)
  {
    b(c, bParam);
  }
  if(functionId == 3)
  {
    b2();
    n();
  }
  y = fValue;
  x(z);
}
```

returns a merged function definition. The algorithm identifies the merge points. In the example, the merge points are the two If conditional branches and the position of 'bParam'. After identifying the merge points, the algorithm arrives at the best code transformation pattern to perform the merge. In our example, there are two resolution patterns, one is the extra parameter and the other an if conditional branch. There are many other possibilities depending on the type of nodes in the merge point.

4 Experiments

We evaluated our approach by building a prototype in C++ and using the prototype to merge existing near-clones in popular GitHub open source repositories, including Google's Protobuf and Oracle's Nodedb. Majority of our abstractions were merged into production code, thereby validating the quality of our abstractions.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 63

Hidden Truths in Dead Software Paths

Michael Eichberg[1] Ben Hermann[2] Mira Mezini[3] and Leonid Glanz[4]

Abstract: Approaches and techniques for statically finding a multitude of issues in source code have been developed in the past. A core property of these approaches is that they are usually targeted towards finding only a very specific kind of issue and that the effort to develop such an analysis is significant. This strictly limits the number of kinds of issues that can be detected.

In this paper, we discuss a generic approach – based on the detection of infeasible paths in code – that can discover a wide range of code smells ranging from useless code that hinders comprehension to real bugs. The issues are identified by computing the difference between the control-flow graph that contains all technically possible edges and the corresponding graph recorded while performing a more precise analysis using abstract interpretation.

The approach was evaluated using the Java Development Kit as well as the Qualitas Corpus (a collection of over 100 Java Applications) and enabled us to find thousands of issues.

1 Overview

Since the 1970s many approaches have been developed that use static analyses to identify a multitude of different types of issues in source code [Co06, CA01, GYF06]. The techniques used by these approaches range from pattern matching [Co06] to using formal methods [Co09] and vary widely w.r.t. their precision and scalability. But, they have in common that each one only targets a very specific kind of issues. Those tools (e.g., FindBugs [Co06]) that can identify issues across a wide(r) range of issues are typically just suits of relatively independent analyses. In all cases, the issues that can be found are limited to those that are identified by some tool developer.

We present a generic approach that detects control- and data-flow dependent issues in Java Bytecode without targeting any specific kind of issues per se. The approach applies abstract interpretation based techniques to analyze the code and while doing so records the paths that are taken. Afterwards, the analysis compares the recorded paths with the set of all paths that could be taken according to a naïve control-flow analysis that does not consider any data-flows. The paths computed by the latter analysis, but not found in the former graph, are then reported along with a justification why they were not taken.

The rationale underlying this approach is that many issues such as null dereferences or array index out of bounds accesses lead to executions that leave infeasible paths behind.

[1] Technische Universität Darmstadt, Fachbereich Informatik Fachgebiet Softwaretechnik, Hochschulstraße 10, 64289 Darmstadt, eichberg@cs.tu-darmstadt.de
[2] hermann@cs.tu-darmstadt.de
[3] mezini@cs.tu-darmstadt.de
[4] glanz@cs.tu-darmstadt.de

Hence, the hypothesis underlying the approach is threefold. First, in well-written code every path between an instruction and all it's direct successors is eventually taken, and, second, a path that will never be taken indicates an issue. Third, a large class of relevant issues manifests itself sooner or later in infeasible paths.

Though we opted for analyzing the code as precisely as possible, we deliberately limited the scope of the analysis to make it scalable. We start with each method of a project and then perform a context-sensitive analysis with a very small maximum call chain size. This makes the analysis unsound – i.e. we may miss certain issues – but it enables us to use it for large industrial sized libraries and applications.

To validate our approach we analyzed the Java Development Kit (JDK 1.8.0_25) and also the applications of the Qualitas Corpus [Te10]. The issues that we found range from seemingly benign issues to serious bugs that will lead to exceptions at runtime or to dead features. However, even at first sight benign issues, such as unnecessary checks that test what is already guaranteed, can have, e.g., an impact in code reviews such code generally hinders comprehension.[5]

2 Conclusion

The proposed approach is based on the idea that infeasible paths in software are a good indication of code issues and that a large class of relevant issues manifest themself sooner or later in infeasible paths. The implementation relies on a new static analysis technique that exploits abstract interpretation and is parametrized over abstract domains as well as the depth of call chains to follow inter-procedurally. This enables us to make informed reasonable trade-offs between scalability and soundness. The validity of the claims is evaluated by doing a case study of industrial size software; the issues revealed during the case study constitute themselves a valuable contribution of the paper and are publicly available.

References

[CA01] Cyrille, A.; Armin, B.: Applying Static Analysis to Large-Scale, Multi-Threaded Java Programs. In: Proceedings of ASWEC '01. IEEE Computer Society, 2001.

[Co06] Cole, B.; Hakim, D.; Hovemeyer, D.; Lazarus, R.; Pugh, W.; Stephens, K.: Improving Your Software Using Static Analysis to Find Bugs. In: Companion to OOPSLA '06. ACM, 2006.

[Co09] Cousot, P.; Cousot, R.; Feret, J.; Mauborgne, L.; Miné, A.; Rival, X.: Why Does Astrée Scale Up? Form. Methods Syst. Des., 35(3):229–264, December 2009.

[GYF06] Geay, E.; Yahav, E.; Fink, S.: Continuous Code-quality Assurance with SAFE. In: Proceedings of PEPM '06. ACM, 2006.

[Te10] Tempero, E.; Anslow, E.; Dietrich, J.; Han, T.; Li, J.; Lumpe, M.; Melton, H.; Noble, J.: Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In: APSEC2010. 2010.

[5] The tool and the data set are available for download at `www.opal-project.de/tools/bugpicker`.

Effekte modellbasierter Test- und Analyseverfahren in Unternehmen: Ergebnisse einer großangelegten empirischen Evaluation mittels industrieller Fallstudien [1]

Michael Kläs[2], Thomas Bauer[3], Andreas Dereani[4], Thomas Söderqvist[5], Philipp Helle[6]

Abstract: Bei der Erstellung eingebetteter Systeme spielen neben modellbasierten Softwareentwicklungsverfahren zunehmend auch modellbasierte Prüfverfahren eine Rolle. Entsprechende Qualitätssicherungstechniken und -werkzeuge verheißend hierbei Kosteneinsparungen und höhere Produktqualität. Leider existieren derzeit nur wenige empirische Daten, die diesbezügliche Aussagen auch in der Praxis belegen. Der Beitrag stellt daher eine aktuelle unternehmensübergreifende Studie vor, die den Einfluss modellbasierter Qualitätssicherung im industriellen Kontext auf eine Reihe von Verbesserungsziele untersucht. Die zugrundeliegenden Daten stammen aus 13 Fallstudien, die im Rahmen eines europäischen Forschungsprojekts bei Unternehmen aus dem Verkehrssektor durchgeführt wurden. Die aggregierten Ergebnisse deuten dabei auf deutliche Kosteneinsparungspotentiale hin, so können die Verifikations- und Validationskosten im Durchschnitt um 29% bis 34% gesenkt werden. Ebenso können Fehlerbehebungskosten im Mittel um 22% bis 32% reduziert werden. Vergleichsweise gering sind hingegen die beobachteten Verbesserungen bezüglich Testüberdeckung und Fehlerzahlen, wie auch bezüglich Produkteinführungszeiten.

Keywords: Empirical study, embedded software quality assurance, multiple case study, GQM+Strategies, quantitative technology evaluation, model-based testing, internal baselines.

1 Einleitung und Forschungsfragen

Der Beitrag basiert auf der Veröffentlichung "*A Large-Scale Technology Evaluation Study: Effects of Model-based Analysis and Testing*" von Michael Kläs, Thomas Bauer, Andreas Dereani, Thomas Söderqvist und Philipp Helle auf der 37. International Conference on Software Engineering (ICSE) 2015 [Kla15].

In dieser wurden die folgenden drei Forschungsfragen untersucht:

[1] Teile der hier vorgestellten Arbeiten und Forschungsergebnisse wurden im Rahmen des ARTEMIS Projekts MBAT (no. 269335), sowie der Projekte ARAMiS (01IS11035) und SPES (01IS12005E) gefördert.

[2] Fraunhofer Institute for Experimental Software Engineering (IESE), Data Engineering, Fraunhofer-Platz 1, 67663 Kaiserslautern, michael.klaes@iese.fraunhofer.de
[3] Fraunhofer Institute for Experimental Software Engineering (IESE), Embedded System Quality Assurance, Fraunhofer-Platz 1, 67663 Kaiserslautern, thomas.bauer@iese.fraunhofer.de
[4] Daimler AG, Sindelfingen, Germany, andreas.dereani@daimler.com
[5] Volvo Group Trucks Technology, Advanced Technology & Research, Gothenburg, Sweden, thomas.soderqvist@volvo.com
[6] Airbus Group Innovations, Hamburg, Germany, philipp.helle@airbus.com

- Welche Ziele werden mit einer Einführung von (integrierten) modellbasierten Analyse- und Testtechnologien (MBAT) bei eingebetteten System in den Domänen Schienen-, Automobil-, und Luftverkehr verfolgt und mit welchen Strategien sollen diese erreicht werden?
- Wie können angestrebte Verbesserungen quantifiziert und im Rahmen einer großen, mehrere Einzelfallstudien umfassenden Evaluation beurteilt werden?
- Welche Verbesserungen können durch MBAT Technologien erreicht werden?

2 Vorgehen und Ergebnisse

In der Studie wurden zu Beginn wichtige Verbesserungsziele und Strategien im Umfeld der (integrierten) Nutzung modelbasierter Analysen und Testverfahren basierend auf einer Dokumentenanalyse und einer Umfrage bei den potentiellen Fallstudiengebern erhoben und konsolidiert. Dabei kam eine vereinfachte Variante der GQM$^+$Strategies Methode [Bas14] zum Einsatz [KBT13]. Das Ergebnis waren GQM$^+$Strategies-Graphen für fünf Hauptziele mit insgesamt 23 Unterzielen sowie entsprechenden Strategien. Die konsolidierten Ziele wurden anschließend mit abstrakten Maßen quantifiziert und durch fallstudienspezifische Messpläne individuell operationalisiert. Entsprechende Daten wurden anschließend in den verschiedenen Fallstudien erhoben. Um den Vertraulichkeitsanforderungen der Fallstudiengeber nachzukommen, wurde dabei das Konzept der internen Baselines [KBT13] genutzt bei dem keine absoluten Zahlenwerte sondern relative Verbesserungen kommuniziert und analysiert werden.

Die aggregierten Ergebnisse zeigen in den Zielbereichen mit Kostenbezug deutliche Einsparungen bezüglich Fehler- wie auch Qualitätssicherungskosten auf. Hinsichtlich Markeinführungszeiten und Produktqualität waren eher geringe Effekte zu verzeichnen. Eine erhoffte Reduktion der Gesamtbetriebskosten der Entwicklungsumgebung konnte zumindest zum Studienzeitpunkt noch nicht beobachtet werden. Die detaillierten Ergebnisse mit Unsicherheitsintervallen sind in der Publikation [Kla15] zu finden.

Literaturverzeichnis

[Kla15] Kläs, M.; Bauer, T.; Dereani, A.; Söderqvist, T.; Helle, P.: A Large-Scale Technology Evaluation Study: Effects of Model-based Analysis and Testing. In: Proc. 37th IEEE International Conference on Software Engineering (ICSE), Florence, S. 119-128, 2015.

[KBT13] Kläs, M.; Bauer, T.; Tiberi, U.: Beyond Herding Cats: Aligning Quantitative Technology Evaluation in Large-Scale Research Projects. In: Proceedings of 14th International Conference on Product-Focused Software Development and Process Improvement (PROFES), Paphos, Cyprus, Springer, S. 80-92, 2013.

[Bas14] Basili, V.; Trendowicz, A.; Kowalczyk, M.; Heidrich, J.; Seaman, C.; Münch, J.; Rombach, D.: Aligning Organizations through Measurement - The GQM+Strategies Approach. Springer-Verlag, 2014.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 67

Empirical Software Metrics for Benchmarking of Verification Tools

Yulia Demyanova Thomas Pani Helmut Veith und Florian Zuleger[1]

Abstract: In recent work [De15, PVZ15, DVZ13], we study empirical metrics for software (SW) source code, which can predict the performance of verification tools on specific types of SW. Our metrics comprise variable usage patterns, loop patterns, as well as indicators of control-flow complexity and are extracted by simple data-flow analyses. We demonstrate that our metrics are powerful enough to devise a machine-learning based portfolio solver for SW verification. We show that this portfolio solver would be the (hypothetical) overall winner of both the 2014 and 2015 International Competition on Software Verification (SV-COMP). This gives strong empirical evidence for the predictive power of our metrics and demonstrates the viability of portfolio solvers for SW verification.

Keywords: Software verification, software metrics, portfolio solver, machine learning.

A modern verification tool needs to pick and choose how to combine a multitude of methods from the fields of model checking, static analysis, shape analysis, SAT solving, SMT solving, abstract interpretation, termination analysis, pointer analysis etc. The trade-offs are based on both technical and pragmatic aspects: many tools are either optimized for specific application areas (e.g. device drivers), or towards the in-depth development of a technique for a restricted program model (e.g. termination for integer programs).

In [De15] we demonstrate that the results of the annual *International Competition on Software Verification* (SV-COMP) [Be15] can be explained by intuitive metrics on the source code. In fact, the metrics are strong enough to enable us to construct a *portfolio solver* which would (hypothetically) win SV-COMP 2014 and 2015. Here, a portfolio solver is a SW verification tool that uses heuristic preprocessing to select one of the existing tools.

As an approach to SW verification, portfolio solving brings interesting advantages: (1) a portfolio solver optimally uses available resources, (2) it can avoid incorrect results of partially unsound tools, (3) machine learning in combination with portfolio solving allows us to select between multiple versions of the same tool with different runtime parameters, (4) the portfolio solver gives good insight into the state-of-the-art in SW verification.

To choose the SW metrics, we consider the zoo of techniques discussed above along with their target domains, our intuition as programmers, as well as the tool developer reports in their competition contributions. The obtained metrics are naturally understood in three dimensions: program variables, program loops, and control flow.

In [De15, PVZ15, DVZ13] we describe metrics which correspond to these dimensions, and are based on simple data-flow analyses. Our algorithm for the portfolio is based on

[1] TU Wien, Arbeitsbereich Formal Methods in Systems Design, Institut für Informationssysteme 184/4, Favoritenstraße 9–11, 1040 Wien, Austria

machine learning using *support vector machines* (SVMs) [CV95] over the metrics defined above. Figure 1 depicts our experimental results on SV-COMP'15: Our tool TP is the overall winner and outperforms all other tools.

	blast	cas-cade	cbmc	cpa-che-cker	pre-dator-hp	smack	ulti-mate-kojak	ulcseq	TP
Overall	737 4546	806 5146	684 11936	**2228** **6288**	389 96	**1542** **8727**	1215 7979	273 12563	**2511** **6260**
Medals	1/0/0	0/0/0	1/1/1	2/1/5	1/0/1	2/1/1	0/2/0	0/0/0	1/6/1

Fig. 1: Experimental results for the eight best competition participants in SV-COMP'15 *Overall*, plus our portfolio TP, given as arithmetic mean of 10 experiments on randomly selected 40% subsets chosen for testing. The first row shows the *Overall* SV-COMP score and beneath it the runtime in minutes. We highlight the *Overall* gold, silver, and bronze medal in dark gray, light gray and white+bold font, respectively. The second row shows the number of gold/silver/bronze medals won in individual categories.

While portfolio solvers are important, we also think that the SW metrics we define in this work are interesting in their own right. Our results show that categories in SV-COMP have characteristic metrics. Thus, the metrics can be used to 1) characterize benchmarks not publicly available, 2) understand large benchmarks without manual inspection, 3) understand presence of language constructs in benchmarks.

Summarizing, our work makes the following contributions:

- We define software metrics along the three dimensions – program variables, program loops and control flow – in order to capture the difficulty of program analysis tasks.
- We develop a machine-learning based portfolio solver for software verification that learns the best-performing tool from a training set.
- We experimentally demonstrate the predictive power of our software metrics in conjunction with our portfolio solver on the software verification competitions SV-COMP'14 and SV-COMP'15.

References

[Be15] Beyer, Dirk: Software Verification and Verifiable Witnesses - (Report on SV-COMP 2015). In: TACAS. pp. 401–416, 2015.

[CV95] Cortes, Corinna; Vapnik, Vladimir: Support-vector networks. Machine learning, 20(3):273–297, 1995.

[De15] Demyanova, Yulia; Pani, Thomas; Veith, Helmut; Zuleger, Florian: Empirical Software Metrics for Benchmarking of Verification Tools. In: CAV. pp. 561–579, 2015.

[DVZ13] Demyanova, Yulia; Veith, Helmut; Zuleger, Florian: On the Concept of Variable Roles and its Use in Software Analysis. In: FMCAD. pp. 226–229, 2013.

[PVZ15] Pani, Thomas; Veith, Helmut; Zuleger, Florian: Loop Patterns in C Programs. ECEASST, 72, 2015.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 69

An Empirical Study on Program Comprehension with Reactive Programming

Guido Salvaneschi, Sven Amann, Sebastian Proksch, and Mira Mezini[1]

Abstract:

Starting from the first investigations with strictly functional languages, reactive programming has been proposed as *the programming paradigm* for reactive applications. The advantages of designs based on this style over designs based on the Observer design pattern have been studied for a long time. Over the years, researchers have enriched reactive languages with more powerful abstractions, embedded these abstractions into mainstream languages – including object-oriented languages – and applied reactive programming to several domains, like GUIs, animations, Web applications, robotics, and sensor networks. However, an important assumption behind this line of research – that, beside other advantages, reactive programming makes a wide class of otherwise cumbersome applications more comprehensible – has never been evaluated. In this paper, we present the design and the results of the first empirical study that evaluates the effect of reactive programming on comprehensibility compared to the *traditional* object-oriented style with the Observer design pattern. Results confirm the conjecture that comprehensibility is enhanced by reactive programming. In the experiment, the reactive programming group significantly outperforms the other group.

Keywords: Reactive Programming, Controlled Experiment, Program Comprehension

Reactive applications are a wide class of software that needs to respond to internal or external stimuli with a proper action. Examples of such applications include user-interactive software, like GUIs and Web applications, graphical animations, data acquisition from sensors, and distributed event-based systems.

Over the last few years, reactive programming (RP) has gained the attention of researchers and practitioners for the potential to express otherwise complex reactive behavior in intuitive and declarative way. RP has been firstly introduced in Haskell. Influenced by these approaches, implementations of RP have been proposed in several widespread languages, including Java, Javascript and Scala. Recently, concepts inspired by RP have been applied to production frameworks like Microsoft Reactive Extensions (Rx), which received great attention after the Netflix success story. Finally, a lot of attention in the front-end developers community is revealed by the increasing number of libraries that implement RP principles, among others React.js, Bacon.js, Knockout, Meteor, and Reactive.coffee.

The relevance of RP comes from the well-known complexity of reactive applications, which are hard to develop and understand, because of the mixed combination of data and control flow. The Observer design pattern is widely used for such applications. It has the advantage of decoupling observers from observables. But, when it comes to program readability, it does not make things easier, because of dynamic registration, side effects in call-

[1] Technische Universität Darmstadt, Fachbereich Informatik, Fachgebiet Softwaretechnik, Hochschulstr. 10, 64289 Darmstadt, Deutschland, <lastname>@st.informatik.tu-darmstadt.de

backs, and inversion of control. In contrast, RP supports a design based on data flows and time-changing values: the programmer states which relations should be enforced among the variables that compose a reactive program and the RP runtime takes care of performing all the required updates. Dependencies are defined explicitly instead of being hidden in the control flow. Combination can be guided by types as opposed to callbacks that return void. Contrarily to the Observer pattern, control is not inverted and less boilerplate is required, since collecting dependencies and performing the updates is automatized by the framework. Based on these arguments, it has been argued that RP greatly improves over the traditional Observer pattern used in OO programming both from the *software design* perspective as well as from the perspective of facilitating the comprehensibility of the software.

Yet, little empirical evidence has been provided in favor of the claimed advantages of RP – especially enhancement of comprehensibility. Despite the intuition about its potential, the reactive style is not *obviously* more comprehensible than the Observer design pattern. For example, in the Flapjax paper [Me09] a Javascript application based on Observer is compared against a functionally equivalent RP version. The authors argument that the RP version is much easier to comprehend. However, the reader is warned that: *"Obviously, the Flapjax code may not appear any 'easier' to a first-time reader"*. Doubting, at this point, is legitimate: does RP really make reactive applications easier to read? Also, it is unclear how much expertise is required to find a RP program "easier" – if ever.

To fill the gap, this paper provides the first empirical evaluation of the impact of RP on program comprehension compared to the traditional technique based on the Observer design pattern. The experiment, based on the REScala language [SHM14], involves 38 subjects that where divided into an RP group and an OO group. They were shown a reactive application and their understanding of the reactive functionalities was measured. To the best of our knowledge, such a study has never been conducted before. Results show that (1) RP increases correctness of program comprehension, (2) comprehending programs in the RP style does not require more time than comprehending their OO equivalent, and (3) in contrast to OO where score results are correlated to programming skills, with RP (advanced) programming skills are not needed to understand reactive applications. The last result suggests that RP lowers the entrance barrier required to understand reactive applications.

References

[Me09] Meyerovich, Leo A.; Guha, Arjun; Baskin, Jacob; Cooper, Gregory H.; Greenberg, Michael; Bromfield, Aleks; Krishnamurthi, Shriram: Flapjax: A Programming Language for Ajax Applications. In: Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications. OOPSLA '09, ACM, New York, NY, USA, pp. 1–20, 2009.

[SHM14] Salvaneschi, Guido; Hintz, Gerold; Mezini, Mira: REScala: Bridging Between Object-oriented and Functional Style in Reactive Applications. In: Proceedings of the 13th International Conference on Modularity. MODULARITY '14, ACM, New York, NY, USA, pp. 25–36, 2014.

Supporting Process Model Validation through Natural Language Generation

Henrik Leopold[1] Jan Mendling[2] Artem Polyvyanyy[3]

The design and development of process-aware information systems is often supported by specifying requirements as business process models. Although this approach is generally accepted as an effective strategy, it remains a fundamental challenge to adequately validate these models given the diverging skill set of domain experts and system analysts. As domain experts often do not feel confident in judging the correctness and completeness of process models that system analysts create, the validation often has to regress to a discourse using natural language. In order to support such a discourse appropriately, so-called verbalization techniques have been defined for different types of conceptual models. However, there is currently no sophisticated technique available that is capable of generating natural-looking text from process models. The reason why a proper process model verbalization technique is still missing might be a result of the difficulty to meet this challenge. A process model verbalization technique has to serialize the non-sequential structure of a process model into sequential, yet execution-order preserving, text. In addition, it must be capable of analyzing the short and grammatically varying labels of process model elements and of annotating them with their semantic components like action or business object. Furthermore, the verbalization technique needs to handle optionality of certain pieces of information. In the paper [LMP14], we address this research gap and propose a technique for generating natural language texts from business process models.

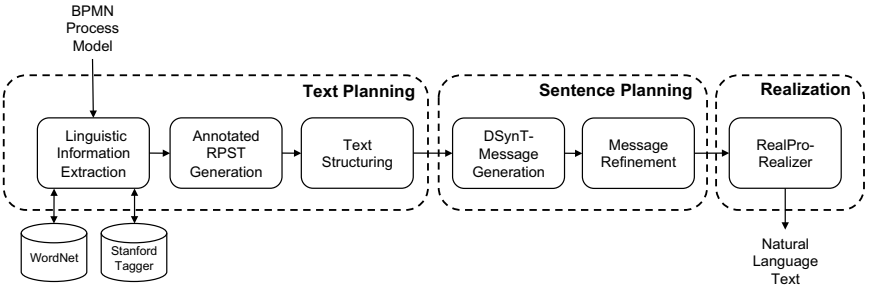


Abb. 1: Architecture of our Natural Language Generation System

The architecture of our text generation technique is building on the traditional pipeline concept from natural language generation systems. The basic rationale of the technique is to utilize the existing information from the model to generate a text. In order to derive

[1] VU University Amsterdam, De Boelelaan 1081, 1081HV Amsterdam, The Netherlands, h.leopold@vu.nl
[2] WU Vienna, Welthandelsplatz 1, 1020 Vienna, Austria, jan.mendling@wu.ac.at
[3] Queensland University of Technology, Brisbane, QLD 4001, Australia

a sequence of sentences, we linearize the model via the creation of a tree structure. In particular, the text generation technique comprises six components (see Figure 1):

1. *Linguistic Information Extraction*: Extraction of linguistic components from the process model element labels.
2. *Annotated RPST Generation*: Linearization of process model through the generation of a tree structure. In addition, each node is annotated with the linguistic information from the previous step.
3. *Text Structuring*: Application of text structuring techniques, such as the insertion of paragraphs and bullet points, based on the computed tree structure.
4. *DSynT-Message Generation*: Generation of an intermediate linguistic message structure for each node of the tree. This component represents the core of the generation technique.
5. *Message Refinement*: Refinement of the generated messages through aggregation or the introduction of referring expressions and discourse markers.
6. *RealPro-Realizer*: Transformation of intermediate message structures to grammatically correct sentences.

To demonstrate the capability of the proposed technique for generating natural language texts from process models, we conducted a two-step evaluation. First, we applied our technique to real-world process models and investigated how the generated texts compare to textual descriptions created by humans. Second, we studied in how far humans are capable of making sense of the generated texts. To this end, we asked humans to transform the generated texts back into process models. The first evaluation step showed that the generated texts convey the model semantics in a more compact and also syntactically less complex manner. Due to the design of the technique, the generated texts are closer to the model and describe the model content and control flow explicitly. The second evaluation step demonstrated that the generated texts are very informative and can successfully be interpreted by humans.

Based on our findings, we conclude that the proposed text generation technique has the potential to facilitate the validation discourse between domain experts and process analysts. First, the generated texts support domain experts in understanding the details of process models even if they are not familiar with process modeling. Second, the text generation may also train domain experts in reading and interpreting process models. As long as text and model are presented together, readers can see and learn about the connection between model and text. Thus, their overall familiarity with process models can be expected to increase in the long term.

References

[LMP14] Leopold, Henrik; Mendling, Jan; Polyvyanyy, Artem: Supporting process model validation through natural language generation. Software Engineering, IEEE Transactions on, 40(8):818–840, 2014.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 73

Kognitive Belastung als lokales Komplexitätsmaß in Geschäftsprozessmodellen

Kathrin Figl[1], Ralf Laue[2]

Abstract: In unserem Beitrag [FL15] untersuchten wir die Verständlichkeit von Geschäftsprozessmodellen. Wir ließen Probanden Aufgaben zu Prozessmodellen lösen, in denen durch logisches Schlussfolgern Fragen zum Modell zu beantworten waren. Neben aus der Literatur bekannten Einflussfaktoren (z.B. Modellierungserfahrung) untersuchten wir, wie sich Metriken, die auf dem Begriff der kognitiven Belastung (cognitive load) aufbauen, auf die korrekte Beantwortung von Fragen auswirken.

Unsere Ergebnisse erlauben eine neue Sicht auf Komplexitätsmetriken für Geschäftsprozessmodelle: Eine bisher kaum beachtete Variable wurde als relevanter Einflussfaktor für die Modellverständlichkeit erkannt – die Interaktivität zwischen Modellelementen. Diese kann durch die Zahl der Modellelemente, die zur Beantwortung einer Frage gemeinsam (in Verbindung zueinander) betrachtet werden müssen, operationalisiert werden.

1 Komplexitätsmetriken für Geschäftsprozessmodelle

In den letzten Jahren erschien eine Vielzahl von Veröffentlichungen, die sich mit Komplexitätsmetriken für Geschäftsprozessmodelle befassten (siehe etwa [RM11]). Eine solche Metrik soll eine Aussage über die Schwierigkeit, das Geschäftsprozessmodell zu verstehen, treffen.

Aktuell verfügbare Metriken erlauben zwar eine Aussage darüber, *wie schwer* ein Modell zu verstehen ist, nicht jedoch darüber *was* genau in einem Prozessmodell zu Verständnisproblemen beitragen kann. In unserem Artikel [FL15] versuchen wir, das Verständnis von Modell-Fragmenten aus lokaler Sicht zu bewerten um schwierige Modellfragmente identifizieren zu können.

Hierzu legten wir 155 Probanden Fragen zu vier Geschäftsprozessmodellen vor, die sie durch logisches Schlussfolgern beantworten sollten. Typische Fragentypen waren etwa „Können A und B gleichzeitig ausgeführt werden?“ oder „Muss in jedem Prozessdurchlauf mindestens einmal C ausgeführt werden?“. Es ist festzustellen, dass zur Beantwortung einer solchen Frage in der Regel nur ein Ausschnitt aus dem gesamten Modell betrachtet werden muss. Somit wird Komplexität - abhängig von der zu lösenden Aufgabe - zu einer *lokalen* Eigenschaft im Modell.

Um diese zu bewerten, gingen wir von den in der Psychologie gewonnenen Erkenntnissen zur kognitiven Belastung aus. Sweller [Sw94] führt den Begriff der Interaktivität zwischen

[1] Wirtschaftsuniversität Wien, Institut für Wirtschaftsinformatik und Neue Medien, kathrin.figl@wu.ac.at
[2] Westsächsische Hochschule Zwickau, Fachgruppe Informatik,ralf.laue@fh-zwickau.de

Elementen ein. Dieser beschreibt, ob Elemente *gemeinsam* wahrgenommen und verstanden werden müssen, um ein bestimmtes Problem zu lösen. Eine hohe Interaktivität zwischen Elementen führt zu einer hohen kognitiven Belastung, da die relevanten Elemente gleichzeitig im Arbeitsgedächtnis gehalten werden müssen.

Als Maß für die kognitive Belastung bei der Beantwortung einer Frage zu einem Geschäftsprozessmodell bestimmten wir die Zahl der Konzepte, die zur Beantwortung einer Frage im Arbeitsgedächtnis gehalten werden müssen. Hierzu nutzten wir eine kanonische Zerlegung eines Geschäftsprozessmodells in Modellfragmente, wobei jedes dieser Modellfragmente einem Prozessablauf-Konzept (also etwa „parallele Ausführung“ oder „Wiederholung“) entspricht. Auf diese Weise führten wir eine lokale Metrik ein, die die Schwierigkeit der Beantwortung einer bestimmten Frage zum Geschäftsprozessmodell misst.

In der Auswertung unseres Experiments zeigte sich ein signifikanter Zusammenhang zwischen dieser Metrik und der Zahl korrekt beantworteter Fragen. Weiterhin bestätigte sich das aus anderen Arbeiten bekannte Ergebnis, dass manche Konzepte schwerer zu verstehen sind als andere. Insbesondere Fragen, zu deren Beantwortung Wiederholungen (Schleifen) im Geschäftsprozessmodellen zu analysieren waren, wurden häufiger falsch beantwortet. Da sowohl Schleifen als auch Prozesspfadverzweigungen mit dem Symbol „XOR“ modelliert werden, ergibt sich folgende Überlegung: Komplexitätsmetriken, die nur die Zahl der XOR-Elemente oder die Zahl ihrer Ausgangskanten berücksichtigen (etwa genutzt in [Sa10]), könnten an Vorhersagekraft gewinnen, wenn sie die zusätzliche Information miteinbeziehen würden, ob mit einem XOR-Symbol eine Wiederholung oder eine Fallunterscheidung modelliert wird.

Da die Interaktivität zwischen Modellelementen einen messbaren Einfluss auf die richtige Beantwortung von Fragen ausübt, sollte diese Variable auch in zukünftigen Experimenten, welche die Verständlichkeit von Modellen messen, als Kontrollvariable beachtet werden.

Literaturverzeichnis

[FL15] Figl, Kathrin; Laue, Ralf: Influence factors for local comprehensibility of process models. International Journal of Human-Computer Studies, 82:96 – 110, 2015.

[RM11] Reijers, Hajo A.; Mendling, Jan: A Study Into the Factors That Influence the Understandability of Business Process Models. IEEE Transactions on Systems, Man, and Cybernetics, Part A, 41(3):449–462, 2011.

[Sa10] Sanchez-Gonzalez, Laura; García, Félix; Mendling, Jan; Ruiz, Francisco: Quality Assessment of Business Process Models Based on Thresholds. In: On the Move to Meaningful Internet Systems, Jgg. 6426 in LNCS, S. 78–95. 2010.

[Sw94] Sweller, John: Cognitive load theory, learning difficulty, and instructional design. Learning and Instruction, 4(4):295 – 312, 1994.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 75

Automatic Detection and Resolution of Lexical Ambiguity in Process Models (Extended Abstract)

Fabian Pittke[1], Henrik Leopold[2], and Jan Mendling[3]

Process models play an important role in various system-related management activities including requirements elicitation, domain analysis, software design as well as documentation of databases, business processes, and software systems. However, it has been found that the correct and meaningful usage of process models appears to be a challenge in practical settings requiring the usage of automatic model analysis techniques. Up until now, such automatic quality assurance is mainly available for checking formal properties of process models. For instance, there is a rich set of analysis techniques to check control-flow-related properties of process models, such as soundness. There are only a few techniques available for checking guidelines on text labels with regard to terminological ambiguity. Moreover, the terminology problem is even more serious in process models since the process model gives an abstract view of the business process and provides only limited context to detect and resolve ambiguity issues. It is thus the goal of [PLM15] to address the need for automatic techniques as well as to define detection and resolution technique for textual ambiguities that improve the terminological quality of a process models and repositories thereof.

For that purpose, our approach addresses three important issues, i.e. the manual effort, the missing focus on process model text fragments, and the focus on single models. First, the *required manual effort*, refers to the extensive amount of manual work that is required to detect and resolve ambiguities in process models. The human effort can be tremendous since organizations tend to maintain several hundreds or even thousands of process models. Second, the *missing focus on process model text fragments* refers to the fact that many approaches of ambiguity detection and resolution are tailored to deal with sentences and phrases taken from a grammatically and syntactically correct natural language text. However, the elements of process models contain only short textual fragments that do not exhibit a complete or a correct sentence structure impeding the direct application of such approaches. Third, the *focus on single models* relates to the observation that available techniques consider only single models or smaller units thereof. Hence, these techniques address ambiguities within a single document or process model. However, since we assume a repository of several process models, the correction of ambiguities on document level might introduce an inconsistency in another document or model.

Our approach introduces the notion of semantic vectors that represents all the possible meanings of a term in the context of a process model. A semantic vector interprets the

[1] WU Vienna, Welthandelsplatz 1, 1020 Vienna, Austria, fabian.pittke@wu.ac.at
[2] VU University Amsterdam, De Boelelaan 1081, 1081HV Amsterdam, The Netherlands, h.leopold@vu.nl
[3] WU Vienna, Welthandelsplatz 1, 1020 Vienna, Austria, jan.mendling@wu.ac.at

vector dimensions as word meanings and quantifies the occurrence of a particular word meaning with a score that is non-zero. The higher the score, the more prevailing the respective meaning in the process model. Furthermore, our approach operationalizes lexical ambiguity by defining necessary and sufficient ambiguity conditions. While the necessary ambiguity conditions focus on the basic characteristics of lexical ambiguities, the sufficient ambiguity conditions explicitly include the usage context of the respective term. The rationale is motivated by the fact that only because a term *might* be ambiguous, this does not necessarily mean that it actually is. A word is ambiguous only, if the context of the word is not sufficient to infer the correct meaning.

This logic has been used to define ambiguity detection and resolution techniques. Accordingly, the detection technique makes use of the previous conceptualization and combines it with the lexical database BabelNet and its integrated word sense disambiguation method to instantiate the semantic vectors and to evaluate the ambiguity conditions. Furthermore, the approach groups such process models, in which a specific term is used with a similar intention, i.e. models with semantic vectors close to each other. The resolution techniques employs different strategies based on semantic relations that suggest alternative terms for replacement. These strategies are, for instance, based on the hypernym and hyponym relations between terms or the specificity of alternative terms. Again, the resolution strategies make use of the lexical database BabelNet, which provides a rich knowledge base of possible word meanings and semantic relations between them.

These techniques have been evaluated by using three process model collections from practice varying in size, domain, and degree of standardization. In particular, the performance of the detection technique was evaluated by conducting an extensive user experiment. The experiment involved six native English speakers who provided their interpretation of a term in a given model. The performance of the resolution technique has been assessed by quantifying the degree of ambiguity and comparing it before and after applying the resolution strategies to the test collections. The evaluation with the English native speakers illustrates that the detection technique uncovers a relevant share of ambiguous terms within the test collections. Moreover, the introduced metrics highlight the positive effect of the resolution approach, which has lead to a significant reduction of ambiguity.

The results of this research underline the importance of terminological consistency of process models and other conceptual models that are affected by terminological inconsistencies, such as goal models, use case models, or feature models. The results also provide support for the revision of such inconsistent models and repositories and for sustaining the consistency of language and terminology over a longer period of time. Moreover, the techniques make an important contribution to existing quality assurance techniques and represent an important step towards the automated quality assurance of process models.

References

[PLM15] Pittke, Fabian; Leopold, Henrik; Mendling, Jan: Automatic Detection and Resolution of Lexical Ambiguity in Process Models. IEEE Transactions on Software Engineering, 41(6):526–544, 2015.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 77

Design and Evaluation of a Customizable Multi-Domain Reference Architecture on top of Product Lines of Self-Driving Heavy Vehicles – An Industrial Case Study

Jan Schroeder[1] Daniela Holzner[1] Christian Berger[1] Carl-Johan Hoel[2] Leo Laine[2] Anders Magnusson[2]

Abstract: Self-driving vehicles are of high interest for academia and industry at the moment. Particularly, in the transportation domain they exhibit a huge potential to increase companies' competitiveness by automating delivery tasks or construction work. This industrial case study reports on the process of developing and evaluating a multi-domain reference architecture concerned with commercial transport mission planning, execution, and tracking for self-driving vehicles. Therefore, internal and external stakeholders as well as development documents were consulted. The resulting reference architecture is evaluated based on its underlying non-functional requirements ensuring early confirmation of compliance with stakeholder needs. A concrete variant of the architecture was also deployed on a Volvo FMX truck and practically evaluated in an exemplary construction site setting.
This paper summarizes our work Schroeder et al. [Sc15] published at 2015 ICSE.

Keywords: self-driving vehicles, reference architecture, design, evaluation, variability, case study

1 Goal and Research Questions

Preliminary research revealed that so far no industrial case studies showed the successful elicitation, integration, and validation of functional requirements (FR) and non-functional requirements (NFR) for a multi-domain reference architecture supporting transport mission planning, execution, and tracking. Consequently, the main goal of this study to design and evaluate such a reference architecture. This goal was divided into the following two research questions:

1. What are FRs, NFRs, and design patterns for a multi-domain reference architecture that supports transport mission planning, execution, and tracking?
2. How can the resulting reference architecture be evaluated regarding how well it accords with the elicited requirements?

[1] Chalmers | University of Gothenburg, Department of Computer Science and Engineering, SE-41296 Göteborg, Sweden, jan.schroder@gu.se, holzner@student.chalmers.se, christian.berger@gu.se
[2] Volvo Group Trucks Technology, SE-41296 Göteborg, Sweden, {carl-johan.hoel,leo.laine,anders.magnusson.3}@volvo.com

2 Resulting Process, Architecture Design, and Evaluation

The process starts with an extensive data collection during stakeholder interviews, document reviews, and literature reviews. This resulted in unprioritized FRs from customers and domains, and unprioritized NFRs on the self-driving heavy vehicle systems concerned with transport mission planning, execution, and tracking. Prioritization in multiple stakeholder workshops led to concrete functionalities expressed as variations and commonalities necessary for the reference architecture design. The design was expressed in component diagrams and feature models enabling automatic product variant generation. The prioritized NFRs matching the domain needs were incorporated as architectural design patterns.
The resulting design was evaluated using three methods based on scenarios and mathematical models. Evaluation was performed at two distinct stages throughout the process, assessing adaptability, changeability, and stability. The first evaluation was done in an early phase of the architectural design and the second one at the end. The evaluations finalize the architecture development process and ensure that stakeholders' expectations towards the architecture design are fulfilled. Finally a practical evaluation was performed on a test track using a concrete variant of the reference architecture deployed on a Volvo FMX truck.

3 Contributions and Conclusions

The reported industrial case study contributes with a complete process for developing and evaluating reference architectures in the domain of self-driving transport vehicles. It furthermore lists results for each step including functional and non-functional requirements, and resulting design patterns for a reference architecture for this domain. The resulting design of the reference architecture is described in form of component diagrams and feature models. Finally, architectural evaluation is reported for assessing changeability, adaptability, stability, and interoperability.
The resulting reference architecture expresses stakeholder requirements as well domain needs in both functional and non-functional terms. It proved itself being useful in theory and applicable in practice. The development process and the resulting reference architecture can be considered as role model for comparable industrial settings.
More detailed information on this study can be found in [Sc15].

References

[Sc15] Schroeder, Jan; Holzner, Daniela; Berger, Christian; Hoel, Carl-Johan; Laine, Leo; Magnusson, Anders: Design and Evaluation of a Customizable Multi-domain Reference Architecture on Top of Product Lines of Self-driving Heavy Vehicles: An Industrial Case Study. In: Proceedings of the 37th International Conference on Software Engineering - Volume 2. ICSE '15, IEEE Press, Piscataway, NJ, USA, pp. 189–198, 2015.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 79

A Primer on Counterexample Guided Abstraction Refinement of Product-Line Behavioural Models

Maxime Cordy[1], Bruno Dawagne[2], Patrick Heymans[3], Axel Legay[4], Martin Leucker[5] and Pierre-Yves Schobbens[6]

Abstract: The model-checking problem for Software Products Lines (SPLs) is harder than for single systems: variability constitutes a new source of complexity that exacerbates the state-explosion problem. Abstraction techniques have successfully alleviated state explosion in single-system models. However, they need to be adapted to SPLs, to take into account the set of variants that produce a counterexample. In this paper, we recall the main ideas of a paper published elsewhere that applies CEGAR (Counterexample-Guided Abstraction Refinement) and desings new forms of abstraction specifically for SPLs. Experiments are carried out to evaluate the efficiency of our new abstractions. The results show that our abstractions, combined with an appropriate refinement strategy, hold the potential to achieve large reductions in verification time, although they sometimes perform worse.

Keywords: Software Product Lines, Model Checking, CEGAR, Abstraction

Summary[7]

Software Product Lines (SPLs) are families of similar software systems developed together. SPL engineering aims to facilitate the development of the members of a family (called products or variants) by identifying upfront their commonalities and differences. Variability in SPLs is commonly represented in terms of features, i.e., units of difference between products that appear natural to stakeholders. The emergence and the increasing popularity of SPLs have raised the need for SPL-specific quality assurance techniques. Indeed, engineers have to provide solid evidence that all the products they build satisfy their intended requirements. Moreover, in case of failure, they should identify which features, or combinations of features, are responsible for the errors in order to facilitate repair.

Model checking is an automated technique to verify a behavioural model of a system against a property expressed in temporal logic. It relies on an exhaustive exploration of the model in search for counterexamples, i.e., executions that violate the property to verify. Due to its exhaustiveness, model checking is costly in time and memory. When applied to real systems with a typically huge state space, model checking faces a combinatorial blow-up called state explosion. The model-checking problem is even harder for SPLs: in this case, the model checker must either prove the absence of errors or find a counterexample for each variant that can produce a violation. Given that the worst-case number of products of an SPL is exponential in the number of features, variability dramatically exacerbates state explosion. As a consequence, it is not feasible to apply single-system model

[1] University of Namur, Belgium, mcr@info.fundp.ac.be [2] University of Namur, Belgium, bdawagne@student.fundp.ac.be [3] University of Namur, Belgium, phe@info.fundp.ac.be [4] INRIA Rennes, France, axel.legay@inria.fr [5] University of Lübeck, Germany, leucker@isp.uni-luebeck.de [6] University of Namur, Belgium, pys@info.fundp.ac.be
[7] This paper summarizes the paper published in [Co14]. References and further details can be found there.

checking to the thousands of variants that can compose real-world SPLs. In recent years, many variability-aware techniques have been designed to address the SPL model checking problem. These techniques keep track of variability information contained in an SPL behavioural model to associate each execution path to the exact set of variants able to produce it. By doing so, they are able to identify the set of products that violate a given property, and to report a counterexample of violation for each of them. Moreover, being aware of variability allows them to check behaviour common to several products only once. One of the most effective answers to state explosion is model abstraction, which creates more concise and therefore easier to verify models of the system, typically by merging similar states. This reduced size often comes at the cost of inaccuracies in the models: A reported counterexample can therefore be spurious, that is, it exists within the abstract model but the not in the real, concrete model. In this case, the abstraction must be refined to eliminate this false positive. Common methods to achieve this refinement make use of the spurious counterexample itself. They give rise to Counterexample Guided Abstraction Refinement (CEGAR), i.e. abstraction techniques that iteratively refine an abstract model until either they find a real counterexample or they can prove the absence of violation.

In spite of their success in single-system model checking, abstraction techniques for SPLs have received little attention. In [Co14], this gap is filled by proposing SPL-specific abstraction procedures based on CEGAR. Applying CEGAR to SPLs is more tedious because a counterexample can be real for some products and spurious for others. This observation leads to two refinement strategies: one refines the model as soon as it finds a spurious counterexample, whereas the other performs the spuriousity check and the refinement after the discovery of all counterexamples. As for the abstraction of the model, we distinguish between (1) state abstraction that only merge states as in single-model abstraction, (2) feature abstraction that modifies only the variability information contained in the model, and (3) mixed abstraction that combines the previous two types. This latter type is the most complicated to implement, as spuriousness can originate from the merging of states, the abstraction of features, or both. In [Co14], the correctness of the approach is proven on the basis of mathematical relations such as simulation relations. Morover, both abstractions and their combination were implemented in ProVeLines, an SPL model checker previously developed by some of the authors. Experiments were carried out to evaluate the efficiency of different combinations of refinement strategies and abstractions. The results tend to show that state abstraction brings performance gains most of the time, whereas feature abstraction generally results in small losses of performance but achieve huge decreases of verification time in some cases. Preliminary experiments on mixed abstraction tend to show that its performance is comparable to that of state abstraction, although slightly worse on average. Other abstractions of this kind could, however, be designed as part of future work and yield better results

References

[Co14] Cordy, Maxime; Dawagne, Bruno; Heymans, Patrick; Legay, Axel; Leucker, Martin; Schobbens, Pierre-Yves: Counterexample Guided Abstraction Refinement of Product-Line Behavioural Models. In: FSE'14. ACM, Hong Kong, China, November 2014.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 81

On Facilitating Reuse in Multi-goal Test-Suite Generation for Software Product Lines[1][2]

Malte Lochau[3], Johannes Bürdek[3], Stefan Bauregger[3], Andreas Holzer[4], Alexander von Rhein[5], Sven Apel[5], Dirk Beyer[5]

Abstract: Software testing is still the most established and scalable quality-assurance technique in practice today. However, generating effective test suites remains computationally expensive, consisting of repetitive reachability analyses for multiple test goals according to a coverage criterion. This situation is even worse when it comes to testing of entire software product lines (SPL). An SPL consists of a family of similar program variants, thus testing an SPL requires a sufficient coverage of all derivable program variants. Instead of considering every product variant one-by-one, family-based approaches are variability-aware analysis techniques in that they systematically explore similarities among the different variants. Based on this principle, we propose a novel approach for automated product-line test-suite generation incorporating extensive reuse of reachability information among test cases derived for different test goals and/or program variants. The developed tool implementation is built on top of CPA/TIGER which is based on CPACHECKER. We further present experimental evaluation results, revealing a considerable increase in efficiency compared to existing test-case generation techniques.

Software product line (SPL) engineering aims at developing families of similar, yet well-distinguished software products built upon a common core platform. The commonality and variability among the family members (product variants) of an SPL are specified as *features*. In this regard, a feature corresponds to user-configurable product characteristics within the problem domain, as well as implementation artifacts being automatically composeable into implementation variants. The resulting extensive *reuse* of common feature artifacts among product variants facilitates development efficiency as well as product quality compared to one-by-one variant development. However, for SPLs to become fully accepted in practice, software-quality assurance techniques have to become *variability-aware*, too, in order to benefit from SPL reuse principles. In practice, systematic software testing constitutes the most elaborated and wide-spread assurance technique, being directly applicable to software systems at any level of abstraction. In addition, testing enables a controllable trade-off between effectiveness and efficiency. In particular, white-box test generation consists of (automatically) deriving input vectors for a program under test with respect to predefined *test goals*. The derivation of sufficiently large *test suites* is, therefore, guided by test selection metrics, e.g., structural coverage criteria like *basic block coverage* and *condition coverage* [Be13]. These criteria impose multiple test goals, thus requiring *sets* of test input vectors for their complete coverage [Be13]. In case of mission-/safety-

[1] This is a summary of a full article on this topic that appeared in Proc. FASE 2015 [Bü15].

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

[3] TU Darmstadt, Germany
[4] TU Wien, Austria
[5] University of Passau, Germany

critical systems, it is imperative, or even enforced by industrial standards to guarantee a particular degree of code coverage for every delivered product. Technically, automated test input generation requires expensive reachability analyses of the program state space. Symbolic model checking is promising approach for fully automated white-box test generation using counterexamples as test inputs [Be04]. Nevertheless, concerning large sets of complex test goals, scalability issues still obstruct efficient test case generation when being performed for every test goal in separate. This problem becomes even worse while generating test inputs for covering entire product line implementations. To avoid a variant-by-variant (re-)generation of test cases potentially leading to many redundant generation runs, an SPL test-suite generation approach must enhance existing techniques.

In [Bü15], we presented a novel technique for efficient white-box test-suite generation for multi-goal test coverage of product-line implementations. The approach systematically exploits reuse potentials among reachability analysis results by means of similarity among test cases (1) derived for different test goals [Be13], and/or (2) derived for different product variants [Ci11]. The combination of both techniques allows for an incremental, coverage-driven exploration of the state space of entire product lines under test implemented in C enriched with feature parameters. We implemented an SPL test-suite generator for arbitrary coverage criteria on top of the symbolic software model checker CPACHECKER [Be13]. We evaluated our technique considering sample SPL implementations of varying size. Our experiments revealed the applicability of the tool to real-world SPL implementations, as well as a remarkable gain in efficiency obtained from the reuse of reachability analysis results. compared to test suite generation approaches without systematic reuse. As a future work, we plan to improve reuse capabilities by applying multi-property model-checking techniques of CPACHECKER which allows for reachability analyses of multiple test goals in a single run.

Literaturverzeichnis

[Be04] Beyer, Dirk; Chlipala, Adam J.; Henzinger, Thomas A.; Jhala, Ranjit; Majumdar, Rupak: Generating Tests from Counterexamples. In: ICSE. pp. 326–335, 2004.

[Be13] Beyer, Dirk; Holzer, Andreas; Tautschnig, Michael; Veith, Helmut: Information Reuse for Multi-goal Reachability Analyses. In: ESOP, pp. 472–491. Springer, 2013.

[Bü15] Bürdek, Johannes; Lochau, Malte; Bauregger, Stefan; Holzer, Andreas; von Rhein, Alexander; Apel, Sven; Beyer, Dirk: Facilitating Reuse in Multi-Goal Test-Suite Generation for Software Product Lines. In: FASE. Springer, 2015.

[Ci11] Cichos, Harald; Oster, Sebastian; Lochau, Malte; Schürr, Andy: Model-based Coverage-Driven Test Suite Generation for Software Product Lines. In: MoDELS. Springer, pp. 425–439, 2011.

How Reviewers Think About Internal and External Validity in Empirical Software Engineering

Janet Siegmund* Norbert Siegmund† Sven Apel‡

Abstract: Empirical methods have grown common in software engineering, but there is no consensus on how to apply them properly. Is practical relevance key? Do in-ternally valid studies have any value? Should we replicate more to address the trade-off between internal and external validity? We asked the key players of software-engineering research, but they do not agree on answers to these questions.

The original paper has been published at the International Conference on Software Engineering 2015 [SSA15]. Empirical research in software engineering came a long way. From being received as a niche science, the awareness of its importance has increased. In 2005, empirical studies were found in about 2% of papers of major venues and conferences, while in recent years, almost all papers of ICSE, ESEC/FSE, and EMSE reported some kind of empirical evaluation, as we found in a literature review. Thus, the amount of empirically investigated claims has increased considerably.

With the rising awareness and usage of empirical studies, the question of where to go with empirical software-engineering research is also emerging. New programming languages, techniques, and paradigms, new tool support to improve debugging and testing, new visualizations to present information emerge almost daily, and claims regarding their merits need to be evaluated—otherwise, they remain claims. But, how should new approaches be evaluated? Do we want observations that we can fully explain, but with a limited generalizability, or do we want results that are applicable to a variety of circumstances, but where we cannot reliably explain underlying factors and relationships? In other words, do researchers focus on internal validity and control every aspect of the experiment setting, so that differences in the outcome can only be caused by the newly introduced technique? Or, do they focus on external validity and observe their technique in the wild, showing a real-world effect, but without knowing which factors actually caused the observed difference?

This tradeoff between internal and external validity is inherent in empirical research. Due to the options' different objectives, we cannot choose both. Deciding for one of these options is not easy, and existing guidelines are too general to assist in making this decision.

With our work, we want to raise the awareness of this problem: *How should we address the tradeoff between internal or external validity?* In the end, every time we are planning an experiment, we must ask ourselves: Do we ask the right questions? Do we want pure,

*University of Passau, siegmunj@fim.uni-passau.de
†University of Passau, siegmunn@fim.uni-passau.de
‡University of Passau, apel@uni-passau.de

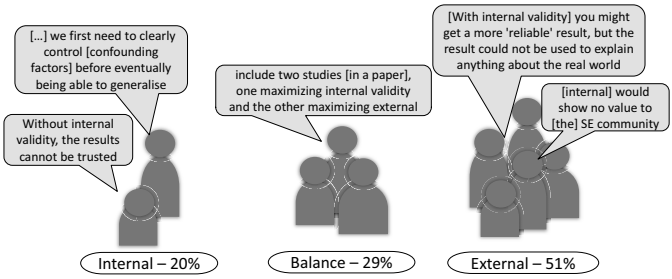


Fig. 1: Preferences for internal vs. external validity among program-committee and editorial-board members.

ground research, or applied research with immediate practical relevance? Is there even a way to design studies such that we can answer both kinds of questions at the same time, or is there no way around replications (i.e., exactly repeated studies or studies that deviate from the original study design only in a few, well-selected factors) in software-engineering research?

To understand how the key players of software-engineering research would address this problem, we conducted a survey among the program-committee members of the major software-engineering venues of the recent years [SSA15]. In essence, we found that there is no agreement and that the opinions of the key players differ considerably (illustrated in Fig. 1). Even worse, we also found a profound lack of awareness regarding the tradeoff between internal and external validity, such that one reviewer would reject a paper that maximizes internal validity, because it "[w]ould show no value at all to SE community". When we asked about replication, many program-committee members admitted that we need more replication in software-engineering research, but also indicated that replications have a difficult stand. One reviewer even states that replications are "a good example of hunting for publications just for the sake of publishing. Come on."

If the key players cannot agree on how to address the tradeoff between internal and external validity (or even do not see this tradeoff), and admit that replication—a well-established technique in other disciplines—would have almost no success in software-engineering research, how should we move forward? In the original paper, we shed light on this question, give insights on the participants' responses, and make suggestions on how we can address the tradeoff between internal and external validity.

References

[SSA15] Janet Siegmund, Norbert Siegmund, and Sven Apel. Views on Internal and External Validity in Empirical Software Engineering. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 9–19. IEEE CS, 2015.

Understanding the Influence of User Participation and Involvement on System Success – a Systematic Mapping Study

Ulrike Abelein[1], Barbara Paech[1]

Abstract: *Context:* User participation and involvement in software development are considered to be essential for a successful software system. We think it is important to analyze user participation and involvement in software engineering comprehensively to encourage further research in this area. *Objectives:* We investigate the evidence on effects of user participation and involvement on system success and we explore which methods are available in literature. *Methods:* A systematic mapping study was conducted. The systematic search yielded 3,698 hits, from which we identified 289 unique papers. *Results:* Based on the empirical evidence of the surveys and meta-studies, we developed a meta-analysis of structural equation models. The analysis of the proposed solutions from the method papers revealed a wide variety of user participation and involvement.

Keywords: User Participation, User Involvement, Software Development, Systematic Mapping Study, Literature Review, Meta-Analysis

In the paper [AP15] we describe a systematic mapping study that examines the influence of User Participation and Involvement (UPI) on system success. We followed the guidelines of [KCh07]. The objective of the study was twofold. First, we wanted to figure out if an increase of UPI increases system success. Second, we wanted to identify the characteristics of methods that increase UPI within software development. To validate the effect of UPI, we used meta-analytical techniques. We extracted the researched aspects, correlation data, variation, and number of participants for validation from 86 studies. The most important finding is that the vast majority of the derived correlations showed a positive effect, thus we can conclude that aspects of the development process and human aspects have a positive effect on system success. In addition, we found that most studies with negative correlations were published more than 10 years ago. These results increase the confidence that UPI is beneficial to system success, which is an important finding for other researchers that develop methods to increase UPI in software development. Nevertheless, the large variation of correlations shows the complexity of measuring and studying UPI. Another important contribution of this paper is the developed classification of the aspects of UPI. This classification can support researchers interested in studying the aspects of UPI.

From the 36 methods papers, an important finding is that all software development

[1] Institute of Computer Science, University of Heidelberg, Im Neuenheimer Feld 326, 69120 Heidelberg, Germany, {abelein, paech}@informatik.uni-heidelberg.de

activities are influenced by the methods, but only few methods focus on the design and implementation activity. This insight can support other researchers in the identification of existing research gaps for methods that aim to increase UPI. In addition, an important contribution of this paper is the structured overview of practices with method examples. The overview shows that practices derived from the solutions have a wide variety in all software activities. The overview is particularly helpful for practitioners, who want to use existing practices and methods to increase UPI in software development. In addition, it can also be valuable to other researchers to understand the state-of-the-art research of UPI methods in software development. The comparison between aspects researched by the surveys and the targeted aspects from the methods reveals that methods for user participation and involvement target similar categories as the surveys. In addition, they target mostly the success factor system quality, which differs from the survey papers that mostly research user satisfaction. The analysis of the validation context revealed that most methods were validated in a public environment.

Overall, we conclude that the systematic mapping study shows a positive correlation of various aspects of UPI on system success. However, there is still no common conceptual model to measure and validate this effect. We identified a broad variety of methods to increase UPI in software development, but they have been validated mostly in smaller projects and in the public sector areas. We therefore suggest to further research and develop new methods for other contexts. Especially in large-scale information technology projects, UPI is not a common practice [A02]. As a follow up to this paper we developed the UDC-LSI method to enhance user-developer communication in large-scale IT [A15]. The analysis of aspects did indicate only little focus on organizational factors or system attributes. However, when we consider large information technology projects within big companies, these projects are heavily influenced by factors such as the complexity of the system and the managerial culture of the organization. Thus, we emphasized those aspects in our UDC-LSI method. In addition, our method also focuses on the software design and implementation activity, as the study reveals that only few methods focus on UPI in these activities, even though within these activities a lot of important decisions are made.

References

[A15] Abelein U.: User-Developer Communication in Large-Scale IT projects. Südwestdeutscher Verlag für Hochschulschriften, 2015.

[AP15] Abelein, U.; Paech, B.: Understanding the Influence of User Participation and INvolvement on System Success: a Systematic Mapping Study, Empirical Softwrae Engineering, vol. 20, p. 28-81, 2015.

[A02] Alleman G.: Agile project management methods for ERP. In: Wells D, Williams L (eds) Extreme Programming and Agile Methods, pp 70–88. Springer Verlag, 2002.

[KCh07] Kitchenham B.; Charters S.: Guidelines for performing systematic literature reviews in oftware engineering, doi: 10.1.1.117.471, 2007

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 87

Hierarchical Software Landscape Visualization

Florian Fittkau[1], Alexander Krause[2], Wilhelm Hasselbring[2]

Abstract: An efficient and effective way to comprehend large software landscapes is required. The current state of the art often visualizes software landscapes via flat graph-based representations of nodes, applications, and their communication. In our ExplorViz visualization, we introduce hierarchical abstractions aiming at solving typical system comprehension tasks fast and accurately for large software landscapes. To evaluate our hierarchical approach, we conduct a controlled experiment comparing our hierarchical landscape visualization to a flat, state-of-the-art visualization. In addition, we thoroughly analyze the strategies employed by the participants and provide a package containing all our experimental data to facilitate the verifiability, reproducibility, and further extensibility of our results. We observed a statistically significant increase in task correctness of the hierarchical visualization group compared to the flat visualization group in our experiment. The time spent on the system comprehension tasks did not show any significant differences. The results backup our claim that our hierarchical concept enhances the current state of the art in landscape visualization for better software system comprehension.

While program comprehension has been researched extensively, system comprehension has received much less attention. From a historical point of view, program comprehension became important when programs reached more than a few hundreds lines of code. Today's IT infrastructures in enterprises often consist of several hundreds of applications forming large software landscapes [FRH15].

Our ExplorViz approach [FWWH13] provides live visualization for large software landscapes introducing three hierarchical abstractions [FRH15]. Life visualization with ExplorViz is scalable [FH15] and elastic in cloud environments [vHRGH09].

We present a controlled experiment to compare a flat, state-of-the-art landscape visualization to our hierarchical visualization in the context of system comprehension [FKH15c]. Additional features of ExplorViz include trace visualizations [FFHW15], architecture conformance checks [FSH14], and a landscape control center [FvHH14] with performance anomaly detection [EvHWH11, MRvHH09]. New perspectives on employing virtual reality [FKH15b] and physical models [FKH15a] are further explored. Beneath evaluating if a hierarchical visualization provides benefits, we conducted this experiment to get input for improving our ExplorViz tool.[3]

[1] PPI AG, Wall 55, 24103 Kiel, Germany, http://www.ppi.de
[2] Kiel University, Software Engineering Group, 24118 Kiel, http://se.informatik.uni-kiel.de/
[3] `http://www.explorviz.net`

References

[EvHWH11] Jens Ehlers, André van Hoorn, Jan Waller, and Wilhelm Hasselbring. Self-Adaptive Software System Monitoring for Performance Anomaly Localization. In *Proceedings of the 8th IEEE/ACM International Conference on Autonomic Computing (ICAC 2011)*, pages 197–200. ACM, June 2011.

[FFHW15] Florian Fittkau, Santje Finke, Wilhelm Hasselbring, and Jan Waller. Comparing Trace Visualizations for Program Comprehension through Controlled Experiments. In *Proceedings of the 23rd IEEE International Conference on Program Comprehension (ICPC 2015)*, pages 266–276. IEEE, May 2015.

[FH15] Florian Fittkau and Wilhelm Hasselbring. Elastic Application-Level Monitoring for Large Software Landscapes in the Cloud. In Schahram Dustdar, Frank Leymann, and Massimo Villari, editors, *Service Oriented and Cloud Computing*, volume 9306 of *Lecture Notes in Computer Science*, pages 80–94. Springer-Verlag, September 2015.

[FKH15a] Florian Fittkau, Erik Koppenhagen, and Wilhelm Hasselbring. Research Perspective on Supporting Software Engineering via Physical 3D Models. In *Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*, pages 125–129. IEEE, September 2015.

[FKH15b] Florian Fittkau, Alexander Krause, and Wilhelm Hasselbring. Exploring Software Cities in Virtual Reality. In *Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*, pages 130–134. IEEE, September 2015.

[FKH15c] Florian Fittkau, Alexander Krause, and Wilhelm Hasselbring. Hierarchical Software Landscape Visualization for System Comprehension: A Controlled Experiment. In *Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*, pages 36–45. IEEE, September 2015.

[FRH15] Florian Fittkau, Sascha Roth, and Wilhelm Hasselbring. ExplorViz: Visual Runtime Behavior Analysis of Enterprise Application Landscapes. In *Proceedings of the 23rd European Conference on Information Systems (ECIS 2015)*, pages 1–13. AIS, 2015.

[FSH14] Florian Fittkau, Phil Stelzer, and Wilhelm Hasselbring. Live Visualization of Large Software Landscapes for Ensuring Architecture Conformance. In *Proceedings of the 2014 European Conference on Software Architecture Workshops (ECSAW 2014)*, pages 28:1–28:4. ACM, August 2014.

[FvHH14] Florian Fittkau, André van Hoorn, and Wilhelm Hasselbring. Towards a Dependability Control Center for Large Software Landscapes. In *Proceedings of the 10th European Dependable Computing Conference (EDCC 2014)*. IEEE, May 2014.

[FWWH13] Florian Fittkau, Jan Waller, Christian Wulf, and Wilhelm Hasselbring. Live Trace Visualization for Comprehending Large Software Landscapes: The ExplorViz Approach. In *Proceedings of the 1st International Working Conference on Software Visualization (VISSOFT 2013)*, pages 1–4, September 2013.

[MRvHH09] Nina S. Marwede, Matthias Rohr, André van Hoorn, and Wilhelm Hasselbring. Automatic Failure Diagnosis in Distributed Large-Scale Software Systems based on Timing Behavior Anomaly Correlation. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR'09)*, pages 47–57. IEEE, 2009.

[vHRGH09] André van Hoorn, Matthias Rohr, Imran Asad Gul, and Wilhelm Hasselbring. An Adaptation Framework Enabling Resource-efficient Operation of Software Systems. In *Proceedings of the Warm Up Workshop (WUP 2009) for ICSE 2010*. ACM, 2009.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 89

Extraktion von Frame Conditions aus Operation Contracts

Philipp Niemann[1], Frank Hilken[1], Martin Gogolla[1], Robert Wille[2]

1 Einleitung und Hintergrund

Um die steigende Komplexität heutiger Softwaresysteme mit modellbasierten Ansätzen zu beherrschen, hat sich die *Unified Modeling Language* (UML) zusammen mit der *Object Constraint Language* (OCL) zu einem de-facto Standard herausgebildet. UML/OCL ermöglicht die Beschreibung von Anforderungen an Verhalten und Struktur komplexer Systeme ohne dabei konkrete Implementierungsdetails zu verlangen. In den vergangenen Jahren wurden viele Verfahren zur Validation und Verifikation von UML/OCL-Modellen vorgestellt.

In dieser Arbeit wird die Beschreibung des Verhaltens mit Hilfe von Klassendiagrammen und dazugehöriger *Operation Contracts* (gegeben in OCL in Form von Vor- und Nachbedingungen) betrachtet [Me92]. Vor- und Nachbedingungen beschreiben die Funktionalität einer Operation auf deklarative Weise. Sie schränken Systemzustände ein, in denen eine Operation ausgeführt werden darf, und beschreiben Eigenschaften, die der resultierende Systemzustand erfüllen muss. Für Verifikationsverfahren ist aber außerdem von Bedeutung, welche Modellelemente zusätzlich zu diesen Bedingungen verändert bzw. eben nicht verändert werden dürfen.

Die Ermittlung eines konkreten Verhaltens einer Operation aus den gegebenen Operation Contracts wird in der Literatur als *Frame Problem* bezeichnet [BMR95]. Zur Lösung des Problems, wurden verschiedene Beschreibungsmittel vorgeschlagen, welche die Vor- und Nachbedingungen um so genannte *Frame Conditions* erweitern [Ko13, BKW09]. Dabei handelt es sich um zusätzliche Bedingungen, die Modellelemente angeben, welche bei der Ausführung einer Operation geändert werden dürfen. Durch die Angabe von Vor- und Nachbedingungen sowie Frame Conditions steht eine vollständige Beschreibung der Funktionalität einer Operation auf Modellebene zur Verfügung.

[1] Department of Computer Science, University of Bremen, 28359 Bremen, Germany, {pniemann,fhilken,gogolla}@informatik.uni-bremen.de
[2] Insitute for Integrated Circuits, Johannes Kepler University Linz, 4040 Linz, Austria, robert.wille@jku.at
Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany

2 Betrachtetes Problem und Beitrag

Die Erzeugung der Frame Conditions stellt allerdings eine zusätzliche Herausforderung dar, die bisher überwiegend manuell bewältigt werden muss – verbunden mit den entsprechenden Kosten. In dieser Arbeit wird untersucht, inwiefern sich Frame Conditions automatisch aus den bereits gegebenen Vor- und Nachbedingungen extrahieren lassen. Dazu wird eine Methodik vorgestellt, die Elemente (Objektinstanzen, Attribute und Rollen) eines gegebenen Modells automatisch einer der folgenden Kategorien zuordnet:

- Das Element ist *variabel*, d.h. kann während der Operationsausführung verändert werden.
- Das Element ist *unveränderlich*, d.h. kann während der Operationsausführung nicht verändert werden.
- Das Element lässt sich nicht eindeutig zuordnen und muss manuell klassifiziert werden.

Ziel dieser Kategorisierung ist, den/die Entwerfer/-in bei der Erzeugung von Frame Conditions zu unterstützen. Fallstudien zeigen, dass in der Tat ein Großteil der Modellelemente automatisch klassifiziert und damit die notwendigen Frame Conditions automatisch erzeugt werden können. Entwerfer/-innen müssen anschließend nur für einen recht kleinen Teil von nicht eindeutig zuordbaren Modellelementen manuell entsprechende Bedingungen ergänzen. Dadurch wird die Produktivität der modellbasierten Verifikation deutlich gesteigert.

3 Weitere Quellen

Die generelle Idee des Problems sowie der Lösungsidee wird in [Ni15a] skizziert. Eine detaillierte Beschreibung des Lösungsansatzes sowie eine Evaluation des Ansatzes findet sich in [Ni15b].

Literaturverzeichnis

[BKW09] Brucker, Achim D.; Krieger, Matthias P.; Wolff, Burkhart: Extending OCL with Null-References. In: MoDELS. S. 261–275, 2009.

[BMR95] Borgida, Alexander; Mylopoulos, John; Reiter, Raymond: On the Frame Problem in Procedure Specifications. IEEE Trans. Software Eng., 21(10):785–798, 1995.

[Ko13] Kosiuczenko, Piotr: Specification of invariability in OCL - Specifying invariable system parts and views. Software and System Modeling, 12(2):415–434, 2013.

[Me92] Meyer, Bertrand: Applying Design by Contract. IEEE Computer, 25(10):40–51, 1992.

[Ni15a] Niemann, Philipp; Hilken, Frank; Gogolla, Martin; Wille, Robert: Assisted generation of frame conditions for formal models. In: DATE. S. 309–312, 2015.

[Ni15b] Niemann, Philipp; Hilken, Frank; Gogolla, Martin; Wille, Robert: Extracting frame conditions from operation contracts. In: MoDELS. 2015.

Model-based Security Verification for Evolving Systems

Jan Jürjens[1,2], Sven Wenzel[2], Daniel Poggenpohl[2], Martín Ochoa[3]

Abstract: Security certification of complex systems requires a high amount of effort. As a particular challenge, today's systems are increasingly long-living and subject to continuous change. After each change of some part of the system, the whole system needs to be re-certified from scratch (since security properties are not in general modular), which is usually far too much effort. We present a tool-supported approach for security certification that minimizes the amount of effort necessary in the case of re-certification after change. It is based on an approach for model-based development of secure software which makes use of the security extension UMLsec of the Unified Modeling Language (UML). It allows the user to integrate security requirements such as secure information flow and audit security into a system design model, it supported by a security verification tool chain, and has been applied to a number of industrial applications.

Keywords: Secure Software Engineering, Model-based Software Development, Security Verification, Software Evolution.

1 Introduction

Security certification of complex systems requires a high amount of effort. Model-based development is a widely accepted methodology where software or parts of it is generated from models. In order to ensure quality properties such as consistency of security requirements the models are often verified prior code generation.

As a particular challenge, today's systems are increasingly long-living and subject to continuous change. After each change of some part of the system, the whole system needs to be re-certified from scratch (since security properties are not in general modular), which is usually far too much effort. Also, if several alternative evolutions of a model are possible, each alternative has to be modeled and verified in order to find the best model for further development and code generation.

We present a tool-supported approach for security certification that minimizes the amount of effort necessary in the case of re-certification after change. It is based on an approach for model-based development of secure software which makes use of the security extension UMLsec of the Unified Modeling Language (UML) [Jur05]. It allows the user to integrate security requirements such as secure information flow and audit security [Jur01] into a system design model and has been applied to a number of industrial applications such as an electronic purse system.

[1] Institute for Software Technology, University of Koblenz-Landau, Koblenz, Germany. http://jan.jurjens.de
[2] Fraunhofer Institute for Software and Systems Engineering ISST, Dortmund, Germany
[3] Singapore University of Technology and Design, Singapore

The approach presented is based on results that determine under which conditions change preserves security properties (for example in the context of structuring techniques such as refinement or architectural principles such as modularization). The approach supports an automated difference-based security analysis, at the level of design models as well as the implementation code (using static security verification [DGJN11] or run-time verification). It has been applied e.g. to cryptographic protocols, distributed security infrastructures, and identity management systems, and there are empirical results comparing it to classical techniques for security certification. In the outlook, we briefly present current research directions, such as applying the approach to the security certification of cloud-based systems.

We present a verification strategy to analyze whether a software evolution preserves a given security property. This is presented on the basis of the UML profile UMLchange which can be used for specifying potential evolutions of a given model simultaneously. UMLchange makes our approach independent from specific modeling tools. We also present an extensible tool that reads the annotations of EMF-based UML2 models and computes a delta model containing all possible evolution paths of the given model. The evolution paths can be verified wrt. security properties, and for each successfully verified path a new model version is generated automatically.

References

[DGJN11] F. Dupressoir, A. D. Gordon, J. Jürjens, D. A. Naumann: Guiding a General-Purpose C Verifier to Prove Cryptographic Protocols. In: 24th IEEE Computer Security Foundations Symposium (CSF), pp. 3-17, 2011.

[HGJF06] S. H. Houmb, G. Georg, J. Jürjens, R. B. France: An Integrated Security Verification and Security Solution Design Trade-off Analysis Approach. In: H. Mouratidis (editors): Integrating Security and Software Engineering: Advances and Future Vision, Idea Group, pp. 190-219, 2006. Invited chapter.

[IMJ11] S. Islam, H. Mouratidis, J. Jürjens: A Framework to Support Alignment of Secure Software Engineering with Legal Regulations. In: Journal of Software and Systems Modeling (SoSyM), Springer-Verlag, vol. 10, no. 3, pp. 369-394, 2011.

[Jur01] J. Jürjens: Modelling Audit Security for Smart-Cart Payment Schemes with UML-SEC. IFIP TC11 Sixteenth Annual Working Conference on Information Security (IFIP/Sec'01), Kluwer 2001, pp. 93-108

[Jur05] J. Jürjens: Secure Systems Development with UML, Springer, 2005

[PWP+07] D. Petriu, M. Woodside, D. Petriu, Jing Xu, T. Israr, G. Georg, R. France, J. Bieman, S. H. Houmb, J. Jürjens: Performance Analysis of Security Aspects in UML Models. In: Sixth Int. Works. on Software and Performance (WOSP'07), pp. 91-102, ACM, 2007.

[WWJO14] S. Wenzel, D. Warzecha, J. Jürjens, M. Ochoa: UMLchange - Specifying Model Changes to Support Security Verification of Potential Evolution. In: Journal of Computer Standards & Interfaces, vol. 36, pp. 776-791, 2014. Special Issue on

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 93

A DSL-Based Approach for Event-Based Monitoring of Systems of Systems

Michael Vierhauser[1], Rick Rabiser[1], Paul Grünbacher[2], Alexander Egyed[2]

Abstract: Complex software-intensive systems such as systems of systems (SoS) need to be monitored at runtime to detect deviations from their requirements. In our earlier work [Vi15a] – summarized in this paper – we described our experiences of developing and applying an SoS monitoring approach based on a Domain-specific Language (DSL) in the domain of industrial automation software. More specifically, we have been developing a constraint DSL for industrial end users as well as an incremental constraint checker for event-based monitoring. Our evaluation demonstrates the expressiveness of our DSL and the scalability of the checker in an industrial scenario.

Keywords: Runtime Monitoring, Systems of Systems, Constraint DSL.

1 Summary

Many software-intensive systems today are systems of systems comprising heterogeneous and independently developed yet interrelated elements. As the full behavior of SoS emerges during operation only, system testing is not sufficient to determine compliance with requirements. Instead, the behavior of the systems and their interactions need to be continuously monitored and checked during operation to detect and analyze deviations from the expected behavior. Checks include the occurrence and order of runtime events (temporal behavior), the interaction of systems (structural behavior), or properties of runtime data (data checks).

Despite the wide variety of existing runtime monitoring approaches, most of these only support particular technologies or certain types of constraints and checks, impeding their application to SoS. In earlier work [Vi15a] we have described our experiences of extending an existing incremental consistency checker for design models [Vi10] to support event-based runtime monitoring of SoS [Vi15b]. Our work is motivated by an industrial case of monitoring a metallurgical plant automation system, an example of an SoS.

More specifically, we have been developing a domain-specific constraint language aiming at industrial end users, who often lack deep programming skills, to ease the definition of various types of constraints. Our DSL-based approach (cf. Fig. 1) allows incrementally checking constraints at runtime. This ensures that violations of requirements

[1] Johannes Kepler Unversität Linz, Christian Doppler Labor MEVSS, Altenberger Str. 69, 4040 Linz, michael.vierhauser@jku.at
[2] Johannes Kepler Unversität Linz, Institut für Software Systems Engineering, Altenberger Str. 69, 4040 Linz, paul.gruenbacher@jku.at

can be reported instantly to users monitoring an SoS. The approach further supports the definition and deployment of constraints at runtime, i.e., constraints can be added or modified without stopping the checker or the monitored systems.

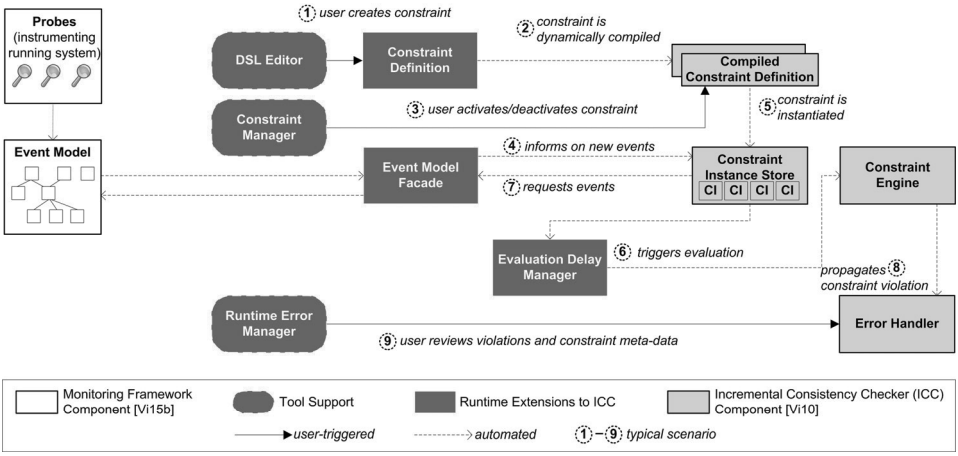


Fig. 1: Our DSL-based approach to define and check constraints at runtime [Vi15a].

In [Vi15a] we demonstrated the expressiveness of the DSL using real constraints from an industrial case. We further evaluated the scalability of our checker in an industrial monitoring scenario. Our experiences suggest designing an extensible constraint DSL in an iterative manner and keeping it as simple as possible. We also suggest keeping the mapping of the DSL to the checking engine flexible to gain independence of underlying checking technologies. Industrial scenarios demonstrate the need to add new or modify existing constraints even while the system and the monitoring infrastructure are running, e.g., if investigating newly emerging and therefore unforeseen issues.

References

[Vi15a] Vierhauser, M., Rabiser, R., Grünbacher, P., Egyed, A.: Developing a DSL-Based Approach for Event-Based Monitoring of Systems of Systems: Experiences and Lessons Learned. Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering, Lincoln, Nebraska, USA, ACM, 2015.

[Vi10] Vierhauser, M., Grünbacher, P., Egyed, A., Rabiser, R., Heider, W.: Flexible and Scalable Consistency Checking on Product Line Variability Models. Proc. of the 25th IEEE/ACM Int'l Conf. on Automated Software Engineering, Antwerp, Belgium, ACM, 2010, pp. 63-72.

[Vi15b] Vierhauser, M., Rabiser, R., Grünbacher, P., Seyerlehner, K., Wallner, S., Zeisel, H.: ReMinds: A Flexible Runtime Monitoring Framework for Systems of Systems. Journal of Systems and Software, 2015 (doi: 10.1016/j.jss.2015.07.008).

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 95

A Vision for Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants

Stefan Fischer[1] Lukas Linsbauer[2] Roberto E. Lopez-Herrejon[3] Alexander Egyed[4]

Many companies build a portfolio of similar product variants, each tailored to different customer needs. The number of such product variants varies widely. We have observed anything between a handful and 1000+ variants and correspondingly large code sizes. The key characteristic of these product variants is that they share a high degree of common functionality (i.e. features) but still differ. Currently the state of the art investigates this problem in form of Software Product Lines (SPLs), which are single, configurable systems from which all desired product variants can be derived. The drawback of SPLs is that they require considerable upfront investments because all possible product variants need to be pre-engineered into SPLs. If it is not possible to predict all these product variants SPL approaches are problematic. And even if all product variants were known a-priori, not all companies could afford building them due to the associated high cost.

In practice we rarely observed full-blown SPLs. Instead, we found that companies often resorted to an ad-hoc practice of clone-and-own, where a new product variant is created by modifying existing variants that closely match the new variant's needs while copying and pasting from other variants as needed. Clone-and-own has three informal steps:

1. *extraction:* locating reusable artifacts (e.g. code) in the existing variants and,
2. *composition:* copying/merging those artifacts that closest match the desired needs into a new product variant, and
3. *completion:* adapting the new product variant to account for needs that did not exist thus far in any existing variant (i.e. new requirements) or could not be extracted.

This paper provides a vision for an approach called ECCO for supporting software engineers in applying clone-and-own [Fi14, Li15, Fi15]. ECCO stands for Extraction and Composition for Clone-and-Own and it allows software engineers to incrementally develop software portfolios, one product at a time, while supporting the reuse of already available product variants. To accomplish this we automated parts of the clone-and-own process, the *extraction* and the *composition*. The *completion* step is still manual, but our approach guides the software engineer with hints.

[1] Johannes Kepler University, Institute for Software Systems Engineering, Linz Austria, stefan.fischer@jku.at
[2] Johannes Kepler University, Institute for Software Systems Engineering, Linz Austria, lukas.linsbauer@jku.at
[3] Johannes Kepler University, Institute for Software Systems Engineering, Linz Austria, roberto.lopez@jku.at
[4] Johannes Kepler University, Institute for Software Systems Engineering, Linz Austria, alexander.egyed@jku.at

The *extraction* step locates artifacts that implement a feature based on commonalities and differences in product variants. The main assumption is, that if two product variants have features in common, then the artifacts they have in common trace to these features. Moreover, the *extraction* is able to deal with feature interactions, which refer to the fact that the implementation of a feature may change depending on the presence or absence of other features.

The *composition* step is the reverse operation of the *extraction*. It merges extracted fragments based on selected features by software engineers who intend to build a new product variant. The fragments are selected not only based on the selected features but the *composition* also takes feature interactions into account. Moreover, references between artifacts and the order of artifacts are considered. The result of the *composition* is a product consisting of the selected features, as far as the *extraction* was able to distinguish them. For features that were not adequately extracted, ECCO provides a set of hints to guide the software engineer during the *completion* step.

In the *completion* step the software engineers finalize a new product variant by adding features/interactions that did not exist thus far. The engineers use the hints provided by ECCO to find missing or surplus features/interactions in the product, i.e. features/interactions that never occurred in any previous product or that could not be separated from others. When two artifacts are merged that never existed together before then ECCO provides suggestions of the orderings of these artifacts which the engineers can choose from. After a product is completed, it can be fed back into the *extraction*, which will refine the knowledge.

ECCO is thus an incrementally evolving SPL that does not require major upfront investments but still facilitates the reuse of already existing product variants. Software engineers do not have to change their development practices. They can continue to develop single product variants the way they are used to but get automated support in doing so. Our approach assumes that product variants exhibit similar code structures (i.e. a common architecture) which is a valid assumption based on our experiences thus far (SPLs also make this assumption). Please find a detailed empirical evaluation in [Fi14].

Acknowledgment The research reported has been supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH, and the Austrian Science Fund (FWF) project P25289 and P25513.

Literaturverzeichnis

[Fi14] Fischer, Stefan; Linsbauer, Lukas; Lopez-Herrejon, Roberto Erick; Egyed, Alexander: Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants. In: ICSME. S. 391–400, 2014.

[Fi15] Fischer, Stefan; Linsbauer, Lukas; Lopez-Herrejon, Roberto E.; Egyed, Alexander: The ECCO Tool: Extraction and Composition for Clone-and-Own. In: ICSE. S. 665–668, 2015.

[Li15] Linsbauer, Lukas; Fischer, Stefan; Lopez-Herrejon, Roberto E.; Egyed, Alexander: Using Traceability for Incremental Construction and Evolution of Software Product Portfolios. In: SST. S. 57–60, 2015.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 97

Morpheus: Variability-Aware Refactoring in the Wild

Jörg Liebig[1] Sven Apel[2] Andreas Janker[3] Florian Garbe[4] and Sebastian Oster[5]

Abstract: Today, many software systems are configurable with conditional compilation. Just like any software system, configurable systems need to be refactored during their evolution. The inherent variability of configurable systems induces an additional dimension of complexity that is not addressed properly by current academic and industrial refactoring engines. Even simple refactorings, such as RENAME IDENTIFIER, are not handled well by existing refactoring engines and may introduce errors in some variants of the configurable system to be refactored. To improve the state of the art, we propose a variability-aware refactoring approach that relies on a *canonical variability representation* and *variability-aware analysis*. The goal is to preserve the behavior of all variants of the configurable system, without compromising general applicability and scalability. To demonstrate practicality, we developed MORPHEUS, a sound variability-aware refactoring engine for C code with preprocessor directives. We applied MORPHEUS to three substantial real-world systems (*Busybox*, *OpenSSL*, and *SQLite*) showing that variability-aware refactoring is practical (i.e., scalable, sound, and complete) in the presence of conditional compilation.

Keywords: configurable systems, refactoring, preprocessor

For more than 40 years software developers implement configurable software systems in the programming language C using conditional compilation with the C preprocessor CPP. Using preprocessor directives, such as `#ifdef`s, developers write optional and alternative code fragments, which form the basis to tailor a configurable system to different application scenarios and use cases. Many developers are familiar with `#ifdef`s, and practically every software system written in C is configurable with conditional compilation [Li10]. To evolve and maintain software systems, software developers usually rely on refactoring engines as part of integrated development environments, such as ECLIPSE. A refactoring engine is a tool, which (semi-)automatically applies code restructings with the goal to improve the internal code structure while preserving the external program behavior [Me02]. However, evolving and maintaining configurable systems is challenging, as the behavior not only of a single system, but of multiple system variants have to be considered, something which is not adressed properly in current refactoring engines.

To assess the state-of-the-art of refactoring engines for configurable systems, we conducted an empirical study to classify strategies of how existing refactoring engines (industrial, open-source, and academic) handle refactorings for configurable systems [Li15]. Overall, we found that there are five different strategies: code restructurings using standard editor facilities (*find/replace*), applying refactorings to single variants only (*single variant*), applying refactorings to multiple variants in isolation (*variant-based*), support for source code with a

[1] Method Park, joerg.liebig@methodpark.de
[2] University of Passau, apel@fim.uni-passau.de
[3] University of Passau, janker@fim.uni-passau.de
[4] University of Passau, fgarbe@fim.uni-passau.de
[5] Method Park, sebastian.oster@methodpark.de

limited set of `#ifdef` usage patterns in configurable code (*limited patterns*), and involving heuristics to reasons about code restructurings in the presence of preprocessor directives (*heuristics*). All strategies suffer from certain limitations that hinder their applicability in practice: they are error-prone (*find/replace*), are incomplete (*single variant* and *limited patterns*), do not scale (*variant-based*), or are unsound (*heuristics*). For example, even a simple refactoring, such as EXTRACT FUNCTION, does not work properly in the popular development environment ECLIPSE in the presence of `#ifdefs`. In ECLIPSE, which applies a *single-variant* strategy, we observed a different program behavior after code restructurings.

As existing strategies and refactoring engines have serious shortcommings, which hinder their application in practice, we developed our own strategy, called *variability-aware refactoring* [Li15]. The idea is to use variability-aware data-structures and algorithms [Wa14, Li13, Kä11] for this task, i.e., data-structures and algorithms, which incorporate variability in the form of `#ifdefs` directly during code restructurings. To assess the applicability of variability-aware refactoring, we implemented variability-aware versions of three common refactorings (RENAME IDENTIFIER, EXTRACT FUNCTION, and INLINE FUNCTION) as part of our own refactoring engine: MORPHEUS. In experiments with three, real-world case studies (*Busybox*, *OpenSSL*, and *SQLite*), we could show that variability-aware refactoring scales well to practical configurable systems, while preserving the behavior of all variants. The average time for applying a single refactoring is in the order of milliseconds. With variability-aware refactoring, we close a gap in tool-support for configurable software systems and show that, for the first time, a scalable, sound, and complete refactoring engine for C (including preprocessor directives) is possible.

References

[Kä11] Kästner, C.; Giarrusso, P.; Rendel, T.; Erdweg, S.; Ostermann, K.; Berger, T.: Variability-Aware Parsing in the Presence of Lexical Macros and Conditional Compilation. In: Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA). ACM, pp. 805–824, 2011.

[Li10] Liebig, J.; Apel, S.; Lengauer, C.; Kästner, C.; Schulze, M.: An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines. In: Proceedings of the International Conference on Software Engineering (ICSE). ACM, pp. 105–114, 2010.

[Li13] Liebig, J.; von Rhein, A.; Kästner, C.; Apel, S.; Dörre, J.; Lengauer, C.: Scalable Analysis of Variable Software. In: Proceedings of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE). ACM, pp. 81–91, 2013.

[Li15] Liebig, J.; Janker, A.; Garbe, F.; Apel, S.; Lengauer, C.: Morpheus: Variability-Aware Refactoring in the Wild. In: Proceedings of the International Conference on Software Engineering (ICSE). ACM, pp. 380–391, 2015.

[Me02] Mens, T.: A State-of-the-Art Survey on Software Merging. IEEE Transactions on Software Engineering, 28(5):449–462, 2002.

[Wa14] Walkingshaw, E.; Kästner, C.; Erwig, M.; Apel, S.; Bodden, E.: Variational Data Structures: Exploring Tradeoffs in Computing with Variability. In: Proceedings of the International Symposium on New Ideas in Programming and Reflections on Software (Onward!). ACM, pp. 213–226, 2014.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 99

Model-Driven Development of Platform-Independent Mobile Applications Supporting Role-based App Variability*

Steffen Vaupel[1], Gabriele Taentzer[1], Renï Gerlach[2], Michael Guckert[2]

Abstract: The use of mobile applications has become an indispensable part of human interaction and especially of urban life. This will lead to rapidly increasing numbers of applications and users that make the development of mobile applications to one of the most promising fields in software engineering. Due to short time-to-market, differing platforms and fast emerging technologies, mobile application development faces typical challenges where model-driven development (MDD) can help. We present a modeling language and an infrastructure for the model-driven development of native apps in Android and iOS. Our approach allows flexible app development on different abstraction levels: compact modeling of standard app elements such as standard data management and increasingly detailed modeling of individual elements to cover specific behavior. Moreover, a kind of variability modeling is supported such that apps variants for a range of user roles can be developed. Several apps including a mobile learning app, a conference app, and a museum guide with augmented reality functionality demonstrate the usefulness of our approach.

Keywords: mobile application, model-driven software development, variability

1 Introduction

An infrastructure for model-driven development (MDD) has a high potential for accelerating the development of software applications. While just modeling the application-specific data structures, processes and layouts, runnable software systems can be generated. Hence, MDD does not concentrate on technical details but lifts software development to a higher abstraction level. The heart and soul of MDD is the domain-specific modeling language. It comes along with a tool environment consisting of textual or visual model editors and appropriate code generators for the desired target platforms (as, e.g., Android and iOS). For the development of our MDD infrastructure, we have chosen an agile *bottom-up* process [VSRT15], starting with a domain analysis and feature identification of mobile applications, template extraction from re-implemented prototypes, and iterative language extension.

2 Domain and approach

Mobile apps are developed for diverse purposes – from mere entertainment to serious business applications. While focusing mainly on data-oriented business apps, our approach allows to enrich them by entertainment and educational elements, or sensor and external hardware access. A particular case is using the built-in camera to recognize objects and

*This work was partially funded by LOEWE HA (State Offensive for the Development of Scientific and Economic Excellence) project no. 355/12-45: PIMAR – Platform Independent Mobile Augmented Reality.

[1] Philipps-Universitït Marburg, Hans Meerwein Straïe 1, 35032 Marburg, Germany, {svaupel, taentzer}@informatik.uni-marburg.de

[2] KITE - Kompetenzzentrum fïr Informationstechnologie, Technische Hochschule Mittelhessen, Wilhelm-Leuschner-Straïe 13, 61169 Friedberg, Germany, {rene.gerlach, michael.guckert}@mnd.thm.de

augment the live view with different types of virtual objects, which is called augmented reality (AR) [GMG+15]. This feature is useful for industry applications as well as for apps in education and tourism sectors.

Although there are already approaches to model-driven development of mobile apps such as MD^2 [HMK13], our contribution differs considerably in design and purpose of the language. Our approach focuses on data-driven apps with role-based variants (Figure 1). The entire approach has three user roles: *app developers* who create the application, *providing users* who may configure the application, and finally *end users* of the app.

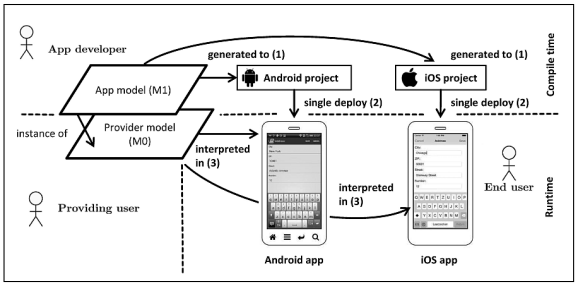


Fig. 1: Cross-platform generation of role-based apps

3 Modeling language, MDD infrastructure, and case studies

The general approach to the modeling language is component-based. An *app model* consists of a *data model* defining the underlying class structure, a *GUI model* containing the definition of pages and style settings for the graphical user interface, and a *process model* which defines the behavioral facilities of an app in the form of processes and tasks. *Provider models* are instances of app models.

The Eclipse-based MDD infrastructure provides a visual model editor (including validation rules) and contains two code generators. In addition to the work presented in [VTH+14] and [PIMAR], the MDD infrastructure has been evaluated at five differently focused case studies showing the applicability and usefulness of the approach. Team members and students have created a *conference app* for the MoDELS'14 conference with conference organizers and participants as user roles, a *learning app* with teachers and learners as user roles, a *museum guide* including AR-functionality with museum providers and visitors as user roles, a control app for power sockets (*SmartPlug*), and a *TV-Reminder*.

References

[GMG+15] Guckert, Michael; Malerczyk, Cornelius; Gerlach, René; Taentzer, Gabriele; Vaupel, Steffen; Fatum, Michael: Plattformunabhängige Entwicklung mobiler Anwendungen mit Augmented Reality-Funktionalität. Anwendungen und Konzepte der Wirtschaftsinformatik, (3):5, 2015.

[HMK13] Heitkötter, Henning; Majchrzak, Tim A.; Kuchen, Herbert: Cross-Platform Model-Driven Development of Mobile Applications with MD^2. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, pp. 526 – 533, 2013.

[PIMAR] PIMAR: Platform-Inpendent Development of Mobile Apps with Augmented Reality. http://www.uni-marburg.de/fb12/swt/forschung/software/pimar/, 2015.

[VTH+14] Vaupel, Steffen; Taentzer, Gabriele; Harries, Jan Peer; Stroh, Raphael; Gerlach, René; Guckert, Michael: Model-driven development of mobile applications allowing role-driven variants. In: Model-Driven Engineering Languages and Systems, pp. 1 – 17, LNCS 8767. Springer, 2014.

[VSRT15] Vaupel, Steffen; Strüber, Daniel; Rieger, Felix; Taentzer, Gabriele: Agile bottom-up development of domain-specific IDEs for model-driven development. In: Proceedings of FlexMDE 2015: Workshop on Flexible Model-Driven Engineering, pp. 12 – 21, Vol. 1470. CEUR-WS.org, 2015.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 101

A Simple and Scalable Static Analysis for Bound Analysis and Amortized Complexity Analysis

Moritz Sinn, Florian Zuleger and Helmut Veith (TU Wien) [1]

Automatic methods for computing bounds on the resource consumption of programs are an active area of research. In [SZV14] we present the first scalable *bound analysis* for imperative programs that achieves *amortized complexity analysis*. Our techniques can be applied for deriving upper bounds on how often loops can be iterated as well as on how often a single or several control locations can be visited in terms of the program input.

The majority of earlier work on bound analysis has focused on mathematically intriguing frameworks for bound analysis. These analyses commonly employ general purpose reasoners such as abstract interpreters, software model checkers or computer algebra tools and therefore rely on elaborate heuristics to work in practice. Our work [SZV14] takes an orthogonal approach that complements previous research. We propose a bound analysis based on a simple abstract program model, namely *lossy vector addition systems with states*. We present a static analysis with four well-defined analysis phases that are executed one after each other: program abstraction, control-flow abstraction, generation of a lexicographic ranking function and bound computation.

```
    void main(uint n) {
      int a = n, b = 0;
l1:   while (a > 0) {
         a--; b++;
l2:      while (b > 0) {
            b--;
l3:         for (int i = n-1; i > 0; i--)
               if (a > 0 && ?) {
l4:               a--; b++;
    }  }  }  }
```

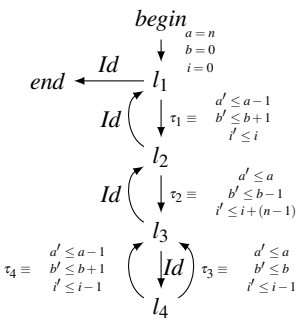


Figure 1: Our running example, '?' denotes non-determinism (arising from a condition not modeled in the analysis). On the right we state the lossy VASS obtained by abstraction, *Id* denotes $a' \leq a, b' \leq b, i' \leq i$.

The example presented in Figure 1 (encountered during our experiments) is challenging for an automated bound analysis: (C1) There are loops whose loop counter is modified by an inner loop: the innermost loop modifies the counter variables a and b of the two outer loops. Thus, the inner loop *cannot be ignored* (i.e., cannot be sliced away) during the analysis of the two outer loops. (C2) The middle loop with loop counter b requires a *path-sensitive* analysis to establish the linear loop bound n: it is not enough to consider how

[1] Supported by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through grants PROSEED and ICT12-059.

often the innermost loop can be executed (at most n^2 times) but rather how often the if-branch of the innermost loop (on which b is actually incremented) can be executed (at most n times). (C3) Current bound analysis techniques cannot model *increments* and instead approximate increments by *resets*, e.g., approximate the increment of b by an assignment to a value between 0 and n (using the fact that n is an upper bound of b)! Because of this overapproximation existing bound analysis techniques fail to compute the linear loop bound n for the middle loop. We illustrate the main steps of our analysis:

1. Program Abstraction: First, our analysis abstracts the program to the VASS depicted in Figure 1. We are using *parameterized* VASSs, where we allow increments that are symbolic but constant throughout the program (such as $n-1$). We extract lossy VASSs from C programs using simple invariant generation and symbolic execution techniques (see [SZV14]).

2. Control Flow Abstraction: In [SZV14] we propose a new abstraction for bound analysis, which we call *control flow abstraction* (CA). CA abstracts the VASS from Figure 1 into a transition system with four transitions: $\rho_1 \equiv a' \leq a-1 \wedge b' \leq b+1 \wedge i' \leq i$, $\rho_2 \equiv a' \leq a \wedge b' \leq b-1 \wedge i' \leq i+(n-1)$, $\rho_3 \equiv a' \leq a \wedge b' \leq b \wedge i' \leq i-1$, $\rho_4 \equiv a' \leq a-1 \wedge b' \leq b+1 \wedge i' \leq i-1$.
CA effectively merges loops at different control locations into a single loop creating one transition for every cyclic path of every loop (without unwinding inner loops). This significantly simplifies the design of the later analysis phases.

3. Ranking Function Generation: Our ranking function generation (discussed in [SZV14]) finds an *order* on the transitions resulting from CA such that there is a variable for every transition, which decreases on that transition and does not increase on the transitions that are lower in the order. This results in the lexicographic ranking function $l = \langle a, a, b, i\rangle$ for the transitions $\rho_1, \rho_4, \rho_2, \rho_3$ in that order. Our soundness theorem (see [SZV14]) guarantees that l proves the termination of Figure 1.

4. Bound Analysis: Our bound analysis (discussed in [SZV14]) computes a bound for every transition ρ by adding for every other transition τ how often τ increases the variable of ρ and by how much. In this way, our bound analysis computes the bound n for ρ_2, because ρ_2 can be incremented by ρ_1 and ρ_4, but this can only happen n times, due to the initial value n of a. Further, our bound analysis computes the bound $n*(n-1)$ for ρ_3 from the fact that only ρ_2 can increase the counter i by $n-1$ and that ρ_2 has the already computed transition-bound n. Our soundness result (see [SZV14]) guarantees that the bound n obtained for ρ_2 is indeed a bound on how often the middle loop of Figure 1 can be executed.

Our bound analysis solves the challenges (C1)-(C3): CA allows us to analyze all loops at once (C1) creating one transition for every loop path (C2). The abstract model of lossy VASS is precise enough to model counter increments, which is a key requirement for achieving amortized complexity analysis (C3).

References

[SZV14] Sinn, Moritz; Zuleger, Florian; Veith, Helmut: A simple and scalable static analysis for bound analysis and amortized complexity analysis. In: CAV. Springer, pp. 745–761, 2014.

GR(1) Synthesis for LTL Specification Patterns

Shahar Maoz[1] and Jan Oliver Ringert[1]

This abstract reports on [MR15a], where we investigated support for linear temporal logic (LTL) specification patterns in General Reactivity of Rank 1 (GR(1)) synthesis.

Reactive synthesis is an automated procedure to obtain a correct-by-construction reactive system from its temporal logic specification [PR89]. Rather than manually constructing a system and using model checking to verify its compliance with its specification, synthesis offers an approach where a correct implementation of the system is automatically obtained for a given specification, if such an implementation exists. In the case of reactive synthesis, an implementation is typically given as an automaton that accepts input from the environment (e.g., from sensors) and produces the system's output (e.g., on actuators).

Two main challenges in bringing reactive synthesis to software engineering practice are its very high worst-case complexity – for LTL it is double exponential in the length of the formula, and the difficulty of writing declarative specifications using basic LTL operators.

To address the first challenge, Piterman et al. [PPS06, Bl12] have suggested the GR(1) fragment of LTL, which has an efficient polynomial time symbolic synthesis algorithm. GR(1) is a strict assume/guarantee subset of LTL, comprised of constraints for initial states, safety propositions over the current and successor state, and justice constraints (i.e., assertions about what should hold infinitely often). GR(1) synthesis has been used in various application domains and contexts, including robotics [KFP09, MR15b], and scenario-based specifications [MS12], to name a few.

To address the second challenge, Dwyer et al. [DAC99] have identified 55 LTL specification patterns, which are common in industrial specifications and make writing specifications easier. An example pattern is p `occurs between` q `and` r where p, q, and r are parameters of the pattern, which can be instantiated with non-temporal propositions. This pattern is numbered P09 and its semantics expressed in LTL according to [DAC99] is `G (q & !r -> (!r W (p & !r)))`.

In [MR15a] we show that almost all LTL specification patterns identified by Dwyer et al. can be used as assumptions and guarantees in the GR(1) fragment of LTL. Specifically, we present an automated, sound and complete translation of the patterns to the GR(1) form, which effectively results in an efficient reactive synthesis procedure for any specification that is written using the patterns.

Technically, the translation starts from the LTL formula of the pattern, translates it to a minimal deterministic Büchi automaton (DBW), if one exists, and then translates the

[1] School of Computer Science, Tel Aviv University, Israel

automaton to a GR(1) assumption or guarantee formula, while possibly adding auxiliary variables to the GR(1) synthesis problem. If no DBW exists, the pattern is not supported.

Critical to the usefulness of our approach is that the costly translation of LTL to DBW is done only once for every pattern. In fact, we have already done it and saved the result as a set of templates inside our synthesis tool. This works because patterns are instantiated only with propositions (not with nested temporal operators). We further show that patterns can even be instantiated with past LTL formulas, but not with nested future temporal operators.

To summarize our contribution, [MR15a] answers the following three questions: (1) is GR(1) expressive enough to support the Dwyer et al. patterns, which are well-recognized as common in industrial specifications?, (2) can the translation be done automatically (and correctly)?, and (3) what's the extra cost of doing it (e.g., in number of auxiliary variables)?

To answer the first two questions, we have implemented and automated the translation, and our findings show that 52 of the 55 patterns from the original work of Dwyer et al. [DAC99] can be expressed as assumptions and guarantees in the GR(1) fragment. Moreover, our work shows that the remaining 3 patterns are indeed not expressible as assumptions or guarantees in the GR(1) fragment by our approach.

To answer the third question, our pattern representation in GR(1) requires at most 3 auxiliary variables per pattern instance. This gives an upper bound for the complexity of a GR(1) synthesis problem where patterns are used as assumptions or guarantees. Note that this is a very satisfying result, since based on the translation via a DBW, one could expect in the worst case an exponential number of auxiliary variables per pattern.

References

[Bl12] Bloem, Roderick; Jobstmann, Barbara; Piterman, Nir; Pnueli, Amir; Sa'ar, Yaniv: Synthesis of Reactive(1) Designs. J. Comput. Syst. Sci., 78(3):911–938, 2012.

[DAC99] Dwyer, Matthew B.; Avrunin, George S.; Corbett, James C.: Patterns in Property Specifications for Finite-State Verification. In: ICSE. ACM, pp. 411–420, 1999.

[KFP09] Kress-Gazit, Hadas; Fainekos, Georgios E.; Pappas, George J.: Temporal-Logic-Based Reactive Mission and Motion Planning. IEEE Trans. Robotics, 25(6):1370–1381, 2009.

[MR15a] Maoz, Shahar; Ringert, Jan Oliver: GR(1) Synthesis for LTL Specification Patterns. In: ESEC/FSE. ACM, pp. 96–106, 2015. Supporting materials: `http://smlab.cs.tau.ac.il/syntech/patterns/`.

[MR15b] Maoz, Shahar; Ringert, Jan Oliver: Synthesizing a Lego Forklift Controller in GR(1): A Case Study. In: Proc. 4th Workshop on Synthesis (SYNT), co-located with CAV. 2015.

[MS12] Maoz, Shahar; Sa'ar, Yaniv: Assume-Guarantee Scenarios: Semantics and Synthesis. In: MODELS. volume 7590 of LNCS. Springer, pp. 335–351, 2012.

[PPS06] Piterman, Nir; Pnueli, Amir; Sa'ar, Yaniv: Synthesis of Reactive(1) Designs. In: VMCAI. volume 3855 of LNCS. Springer, pp. 364–380, 2006.

[PR89] Pnueli, Amir; Rosner, Roni: On the Synthesis of a Reactive Module. In: POPL. ACM Press, pp. 179–190, 1989.

Verification Witnesses *

Dirk Beyer [1], Matthias Dangl [1], Daniel Dietsch [2],
Matthias Heizmann [2], and Andreas Stahlbauer [1]

[1] University of Passau, Germany [2] University of Freiburg, Germany

http://www.sosy-lab.org/~dbeyer/verification-witnesses/

Abstract: It is commonly understood that a verification tool should provide a counterex-ample to witness a specification violation. Until recently, software verifiers dumped error witnesses in proprietary formats, which are often neither human- nor machine-readable, and an exchange of witnesses between different verifiers was impossible. We have defined an *exchange format for error witnesses* that is easy to write and read by verification tools (for further processing, e.g., witness validation). To eliminate manual inspection of false alarms, we develop the notion of *stepwise testification*: in a first step, a verifier finds a problematic program path and, in addition to the verification result FALSE, constructs a witness for this path; in the next step, another verifier re-verifies that the witness indeed violates the specification. This process can have more than two steps, each reducing the state space around the error path, making it easier to validate the witness in a later step. An obvious application for testification is the setting where we have two verifiers: one that is efficient but imprecise and another one that is precise but expensive. The technique of error-witness-driven program analysis is implemented in two state-of-the-art verification tools, CPACHECKER and ULTIMATE AUTOMIZER.

Overview

Software verification becomes more and more important in practice; several breakthroughs in verification research were achieved during the last decade, and several successful verification tools were developed. The TACAS International Competition on Software Verification (SV-COMP) [1] [Bey14, Bey15] serves as a showcase of the state-of-the-art. Users can choose from a wide range of verifiers, and the SV-COMP categories give an approximate guidance on which verifier is good for which kind of programs. One important and unsolved problem of applying verification technology in practice is that verification tools sometimes produce false alarms, and it still requires an enormous manual effort to find out if a reported bug indeed represents a genuine specification violation.

Our solution comprises two components: we developed an *exchange format for error witnesses* and evaluated its effectiveness by a thorough experimental evaluation, and we develop the notion of *stepwise testification*, as the technique of witness validation immediately leads to the notion of witness refinement, enabling a chain of verifiers (or testifiers) to continuously refine the erroneous state space until a test vector for the error is found.

*This is a summary of a full article on this topic that appeared in Proc. FSE 2015 [BDD+15].

[1] http://sv-comp.sosy-lab.org/

Testification is the process of giving evidence for a claim that a given program satisfies, or violates, its specification. The evidence of the absence, or presence, of a specification violation is given by one or more witnesses. A verification tool is a *testifier* if it provides evidence to support its claim, i.e., if it produces a witness for correctness or for a violation of the specification. *Stepwise testification* is the process of applying testification in several steps, on ever refined witnesses, possibly using different verification tools, combining different strengths. Figure 1 illustrates the process of stepwise testification. Our study explores stepwise testification of specification violations by producing error witnesses (left part), while conditional model checking [BHKW12] focuses on stepwise testification of correctness.

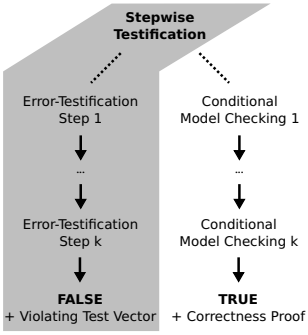


Figure 1: Stepwise testification: conceptual view

We accompany the bug report of verifier V_1 with an error witness, which represents information that can effectively guide another verifier V_2 to efficiently re-explore the state space that verifier V_1 reported to contain a bug. Our experimental study [BDD+15] confirms the following insights: (1) our exchange format makes it possible to communicate error witnesses across verifiers, (2) verifier V_2 needs on average considerably less resources to validate the witness than verifier V_1 needed to find the error, even if V_2 uses a more expensive verification technology (e.g., V_1 using linear and V_2 using bit-precise arithmetic), (3) stepwise testification can be more efficient than verification, i.e., the CPU time for V_1-verification + V_2-witness-validation can be less than the CPU time for V_2-verification alone, (4) the state-space to be analyzed by V_2 is effectively reduced.

On the syntactic level, we use XML, more specifically GraphML, as a language to represent error witnesses. On the semantic level, we use the standard concept of (non-deterministic) finite automata to represent an error witness. A witness automaton observes the paths that the verifier explores and directs the exploration engine along the paths that the witness describes, i.e., towards the violation of the specification. Witnesses can be read by humans or a witness validator.

Our technique was already used in the two most recent editions of the competition on software verification. The SV-COMP community manifested in the competition rules that each answer FALSE must be accompanied by an error witness [Bey15], and requires the organizer to reasonably validate each witness before assigning a success score, in order to get more confidence that the error witness indeed represents a valid bug.

References

[BDD+15] D. Beyer, M. Dangl, D. Dietsch, M. Heizmann, and A. Stahlbauer. Witness Validation and Stepwise Testification across Software Verifiers. In *Proc. ESEC/FSE*, pages 721–733. ACM, 2015.

[Bey14] D. Beyer. Status Report on Software Verification. In *Proc. TACAS*, LNCS 8413, pages 373–388. Springer, 2014.

[Bey15] Dirk Beyer. Software Verification and Verifiable Witnesses (Report on SV-COMP 2015). In *Proc. TACAS*, LNCS 9035, pages 401–416. Springer, 2015.

[BHKW12] D. Beyer, T. A. Henzinger, M. E. Keremoglu, and P. Wendler. Conditional Model Checking: A Technique to Pass Information between Verifiers. In *FSE*. ACM, 2012.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 107

Evolution of Software in Automated Production Systems: Challenges and Research Directions

Birgit Vogel-Heuser[1], Alexander Fay[2], Ina Schaefer[3], and Matthias Tichy[4]

Abstract: Coping with evolution in automated production systems implies cross-disciplinary challenges along the system's life-cycle for variant-rich systems of high complexity. We provide an interdisciplinary survey on challenges and research directions in the evolution of automated production systems. After an initial discussion about the nature of automated production systems and their specific development process, we sketch in this extended abstract the challenges associated with evolution in the different development phases and a couple of cross-cutting areas.

Keywords: Automated Production Systems, Cross-Disciplinary Development, Challenges, Research Directions

1 Automated Production Systems

Automated production systems (aPS) form the backbone of the world's industrial production. They are highly specialized technical systems, which are comprised of mechanical, electrical and electronic parts and software, all closely interwoven. Software is the defining factor to realize modern trends in manufacturing as defined by mass customization, small lot sizes, high variability of product types, and a changing product portfolio during the lifecycle of an automated production system. Hence, the evolution of automated production systems always requires addressing cross-disciplinary evolution challenges.

aPS in special machinery and plant manufacturing industry are typically designed-to-order, i.e. they are unique systems, which are designed and implemented once a customer has awarded a contract to an aPS supplier. Until completion, such projects usually last between several weeks and several months. In order to shorten project durations and to reduce costs, reusable (partial) solutions are usually developed by system suppliers and combined to realize the automated production system. aPS are supposed to be in operation and continuously evolved for decades before they are finally taken out of operation and are demolished.

[1] Institute of Automation and Information Systems, Technische Universität München, Boltzmannstr. 15, 85748 Garching near Munich, Germany, email: vogel-heuser@tum.de
[2] Institute of Automation Technology, Helmut Schmidt University, Holstenhofweg 85, 22043 Hamburg, Germany, email: alexander.fay@hsu-hh.de
[3] Institute of Software Engineering and Automotive Informatics, Technische Universität Braunschweig, Mühlenpfordtstr. 23, 38106 Braunschweig, Germany, email: i.schaefer@tu-braunschweig.de
[4] Institute of Software Engineering and Programming Languages, Universität Ulm, 89069 Ulm, Germany, email: matthias.tichy@uni-ulm.de

2 Challenges and Research Directions

In the following, we discuss the evolution challenges and research directions covering the different development phases as well as several important cross-cutting aspects. We refer to the full paper [1] for a complete discussion of challenges, state of the art, and research directions illustrated using a simple production system.

A light weight and efficient way to define *requirements and system specification* for aPS and refine or change them during the design process needs to be developed for both functional and non-functional requirements. They should be formalized in a way that they are quantifiable but still technology-independent and cover all aspects of the aPS. Thus, their continuous fulfillment can be verified before and after an evolution step. Challenges in *system design* include the need to ensure consistency between design artefacts of the different disciplines as well as different aspects of the system to be built. Furthermore, the concept of technical debt needs to be investigated w.r.t. aPS. During *system realization and implementation of the design* of aPS lack of modularity concepts fulfilling the cross disciplinary requirements of aPS can result in reuse by copy&paste leading to clones. Additionally, the evolution during operation is performed by customer personal on-site which are experts in the production process but not in software engineering and work under stress and time pressure. That can lead to inconsistencies between design and implementation as well as unclear code structures. The main challenge for *validation and verification* under system evolution is to provide efficient techniques to establish the desired system properties after evolution without the need to re-verify the complete evolved system from scratch. In particular, compositional and incremental verification and validation techniques should be developed to cope with cross-discipline models and reduce the effort for re-establishing properties.

The main challenge concerning *variability management* of aPS is the management of multi-disciplinary variability in problem and solution space for functional requirements and non-functional requirements with different levels of abstraction and granularity. A particular problem is here to ensure the consistency of the variability models across disciplines.. There exists research to use *Model-Driven Engineering* for the development of aPS, the challenge is to avoid inconsistencies between the models and the generated code in case of changes on site. Finally, as aPS have a cross-disciplinary nature and many different artifacts are built during development ensuring *traceability* between all artifacts is an important challenge.

References

[1] Birgit Vogel-Heuser , Alexander Fay , Ina Schaefer , Matthias Tichy , Evolution of software in automated production systems - Challenges and Research Directions, The Journal of Systems & Software (2015), Elsevier, doi:10.1016/j.jss.2015.08.026

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 109

The Benefit of Requirements Traceability When Evolving a Software Product: A Controlled Experiment

Patrick Mäder[1] and Alexander Egyed[2]

Abstract: Software traceability is a required component of many software development processes. Advocates of requirements traceability cite advantages like easier program comprehension and support for software maintenance (i.e., software change). However, despite its growing popularity, there exists no published evaluation about the usefulness of requirements traceability. It is important, if not crucial, to investigate whether the use of requirements traceability can significantly support development tasks to eventually justify its costs. We thus conducted a controlled experiment with 71 subjects re-performing real maintenance tasks on two third-party development projects: half of the tasks with and the other half without traceability. Subjects sketched their task solutions on paper to focus on the their ability to solving the problems rather than their programming skills. Our findings show that subjects with traceability performed on average 24% faster on a given task and created on average 50% more correct solutions – suggesting that traceability not only saves effort but can profoundly improve software maintenance quality.

Keywords: requirements traceability; software evolution; effect; controlled experiment; study

1 Motivation and Study

Capture and maintenance of requirements-to-code traces reflect knowledge where requirements are implemented in the code is the focus of extensive research [CHGHH+14]. Despite its growing popularity [MGP09, RMK13] surprisingly little is known about its benefits. Intuitively, requirements-to-code traces should be useful for many areas of software engineering. Researchers refer to better program comprehension and support for software maintenance. Nonetheless, there exists no empirical work in which the effect of requirements traceability was measured. Does it save effort? Does it improve quality?

In this study, originally published at [ME15], we assessed whether available requirements-to-code traces improve the performance of subjects during software maintenance and evolution tasks. Therefore, we conducted an experiment involving 71 practitioners and students with a wide range of experiences. The subjects were asked to solve tasks taken from two software projects: the open source Gantt Project (47 KLOC) and the iTrust system (15 KLOC). Eight tasks were selected, covering real bug fixes and feature extensions taken from the projects' documented archives. Tasks were randomly assigned to subjects, half with and the other half without traceability. Task solutions were recorded on paper and not implemented by the subjects. We measured the performance of subjects as the time they

[1] Technische Universität Ilmenau, Software Systems Group, Ilmenau, Germany, patrick.maeder@tu-ilmenau.de
[2] Johannes Kepler University, Institute for Software Systems Engineering (ISSE), Linz, Austria, alexander.egyed@jku.at

spent to solve a task and the correctness of their solution. The selected, real maintenance tasks also provided us with a gold standard as to how the original developers solved the given tasks. Furthermore, we assessed the influence of subject experience, the kind of tasks subjects were expected to solve, and the different project domains. All subjects were not familiar with the projects – a situation commonly occurring during software maintenance and a situation under which developers are expected to benefit most from traceability.

2 Results and Conclusions

In total, subjects solved 461 tasks (i.e., 6.5 tasks per subject on average). Our findings show that subjects working on tasks with traceability performed better than subjects working without traceability. In particular, subjects with traceability performed on average 24% faster on tasks and created on average 50% more correct solutions. This demonstrates that traceability is not just a means for saving some effort but can profoundly improve the quality of the software maintenance process. There are likely many subsequent benefits such as more effective maintenance, faster time to market, or less code degradation. We also found that some tasks benefited more from traceability than others, especially with regard to the correctness of the solution. Furthermore, we found that our observations were consistent regardless of subject experience and the project domain.

The implications of this study are numerous. Traceability strongly benefits software maintenance regardless of subject experience. Though often perceived tedious and ineffective, this work demonstrates a clear, measurable performance improvement to justify traceability cost. Since this work clearly characterizes the effect of traceability, practitioners and researchers alike may use this information to better understand the cost/benefit trade-off of traceability – a point that will also be the focus of our future work.

Acknowledgments We are funded by the German Ministry of Education and Research (BMBF) grant 01IS14026B and the Austria Science Fund (FWF) grant FWF P 25289-N15.

References

[CHGHH+14] Jane Cleland-Huang, Orlena Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. Software Traceability: Trends and Future Directions. In *Proc. 36th International Conference on Software Engineering (ICSE)*, pages 55–69, 2014.

[ME15] Patrick Mäder and Alexander Egyed. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering*, 20(2):413–441, 2015.

[MGP09] Patrick Mäder, Orlena Gotel, and Ilka Philippow. Motivation Matters in the Traceability Trenches. In *Proc. 17th International Requirements Engineering Conference (RE09)*, pages 143–148, 2009.

[RMK13] Patrick Rempel, Patrick Mäder, and Tobias Kuschke. An Empirical Study on Project-Specific Traceability Strategies. In *Proc. 21st International Requirements Engineering Conference (RE13)*, pages 195–204, 2013.

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 111

Development of Flexible Software Process Lines with Variability Operations: A Longitudinal Case Study

Patrick Dohrmann[1], Joachim Schramm[1] and Marco Kuhrmann[2]

Abstract: *Context:* A software process line helps to systematically develop and manage families of processes and, as part of this, variability operations provide means to modify and reuse pre-defined process assets. *Objective:* Our goal is to evaluate the feasibility of variability operations to support the development of flexible software process lines. *Method:* We conducted a longitudinal study in which we studied 5 variants of the V-Modell XT process line for 2 years. *Results:* Our results show the variability operation instrument feasible in practice. We analyzed 616 operation exemplars addressing various customization scenarios, and we found 87 different operation types. *Conclusions:* Although variability operations are only one instrument among others, our results suggest this instrument useful to implement variability in real-life software process lines.

This summary refers to the paper *Development of Flexible Software Process Lines with Variability Operations: A Longitudinal Case Study* [Do15]. This paper was published as full research paper in the *EASE'2015* proceedings.

Keywords: software process, software process lines, variability operations, longitudinal study

1 Introduction

Different studies show manifold of processes used in practice and companies combine multiple processes and adopt these to specific requirements. Yet, defining adequate processes is a complex activity requiring deep knowledge of the actual domain in particular and software engineering in general. To overcome these challenges, in [Ro05], Rombach votes for adopting well-known concepts from software product lines to develop *Software Process Lines* (SPrL). Still, these approaches lack in evidence of their feasibility in practice. In Germany, the V-Modell XT is the standard software process for IT development projects in Germany's public sector. Starting with its release in 2005, the number of process variants using the so-called reference model increased and led to serious problems when the reference model evolved. Therefore, it was decided to adopt concepts from SPrLs to the V-Modell XT for the purpose of improving support for an efficient management of the reference model and its variants, e.g., automatic updates.

Problem. While defining the variability operations for the V-Modell XT framework, the major problem was to find suitable and actionable variability operations. Furthermore, we lack long-term studies analyzing the feasibility of SPrL approaches.

[1] Technische Universität Clausthal, Institut für Informatik - Software Systems Engineering, Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, {jschr,pdo}@tu-clausthal.de
[2] University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark, kuhrmann@acm.org

Objective, Method, and Contribution. To understand software process variability, we analyze the feasibility of the variability operation instrument and its improvement. We conducted a longitudinal case study in which we investigated two baselines (a reference model and 5 of its variants [Ku14] and [Do15]) of the V-Modell XT conducted by two teams of researchers. In our study, we contribute a catalog 87 different (78 unique) variability operations and a quantitative analysis of their use in practice.

2 Results

Our study of two major releases of the reference model and its variants resulted in 87 variability operations for modifying process structure and content. In the study, we also observed an evolution, and we observed parallel development of variability operation sets by different process-engineering team (studying the result sets in detail shows copied/modified operations, so that we end up with 78 unique variability operations). In total, all variants use variability operations and the number of operation exemplars increased over time (more variants using this instrument, more operation exemplars). Among the 616 operation exemplars, 223 are instances of only 6 types indicating to customization patterns. Further findings show variability operations also aiding process metamodel evolution and the combination with other variability instruments.

3 Conclusion

In summary, we found the concept of variability operations sufficient to support process engineers in constructing a process variant from a software process line. However, variability operations are only one instrument among others and, thus, can (and should) be combined with other instruments.

References

[Do15] Dohrmann, P.; Schramm, J.; Kuhrmann, M.: Development of Flexible Software Process Lines with Variability Operations: A Longitudinal Case Study. Proc. of Int. Conf. on Evaluation and Assessment in Software Engineering, ACM, New York, NY, pp. 13:1-13:10, 2015.

[Ku14] Kuhrmann, M.; Mendez Fernandez, D.; Ternite, T.: Realizing software process lines: Insights and experiences. Proc. of Intl. Conf. on Software and Systems Process, ACM, New York, NY, pp.110–119, 2014.

[Ro05] Rombach, D.: Integrated software process and product lines. Proc. of Int. Software Process Workshop, Springer, Berlin-Heidelberg, pp. 83-90, 2005.

Keynote: Continuous Software Engineering

Wilhelm Hasselbring[1]

Continuous Software Engineering ist seit Jahrzehnten ein kontinuierlich behandeltes Thema in der Forschung und der Praxis des Software Engineering. Traditionell fokussiert die kontinuierliche Softwareentwicklung auf Reverse und Re-Engineering Aktivitäten zur Weiterentwicklung langlebiger Softwaresysteme.

In den letzten Jahren hat dieses Thema in der Praxis durch die Verfügbarkeit von leistungsfähigen Werkzeugen zur Automatisierung stark an Bedeutung gewonnen. Gleichzeitig ist eine Betonung automatisierter Qualitätssicherungsmaßnahmen zu beobachten. Continuous Integration und Continuous Delivery/Deployment sind dazu aktuelle Schlagworte.

Für die kontinuierliche Softwareentwicklung ist es nun auch wichtig neben der Konstruktion und Weiterentwicklung von Software auch schon in der Entwicklung den späteren Betrieb im Rechenzentrum oder in eingebetteten Systemen zu berücksichtigen. Die DevOps-Bewegung zielt darauf ab die Zusammenarbeit von Softwareentwicklung (Dev für Development) und Betrieb (Ops für Operations) zu optimieren und Reibungsverluste zu vermeiden.

In der (agilen) Softwareentwicklung ist es das Ziel, schnell viele Features bereitzustellen. Im Betrieb ist es das Ziel, stabile Dienste bereitzustellen - häufige Änderungen werden hier traditionell als unerwünscht angesehen. DevOps verfolgt nun den Ansatz viele, stabile Releases bereitzustellen. Die dazu erforderliche Qualitätssicherung und Effizienzsteigerung wird durch die Automatisierung von Entwicklungs- und Betriebsaufgaben erreicht.

Generell führt die kontinuierliche Integration von Qualitätssicherungsmaßnahmen zu einer kontinuierlich hohen Qualität und damit zu vielen stabilen Releases. Zur kontinuierlichen Überwachung der resultierenden Softwaredienste und auch der sogenannten Deployment-Pipelines muss möglichst viel automatisiert gemessen und überwacht werden (Monitoring).

Durch DevOps bekommt das Thema Softwarearchitektur auch in der agilen Softwareentwicklung eine stärkere Bedeutung und Würdigung. Für DevOps ist es sinnvoll präskriptive und deskriptive Architektur-Modelle zu kombinieren. Präskriptive Modelle kommen aus der Softwareentwicklung (Forward Engineering). Deskriptive

[1] KoSSE, Universität Kiel, Deutschland, hasselbring@email.uni-kiel.de

Modelle kommen aus der Beobachtung der im Betrieb befindlichen Softwaredienste (Reverse Engineering durch dynamische Analyse). In diesem Vortrag werde ich diskutieren, warum Softwarearchitektur ein zentrales Artefakt an der Schnittstelle zwischen Entwicklung und Betrieb ist. Speziell Microservice-Architekturen und dem kontinuierlichen Monitoring der resultierenden Systeme kommt hier eine besondere Rolle zu.

Keynote: Working with Robots in Smart Homes and Smart Factories – Robotic Co-Working

Uwe Aßmann[1]

Co-working is a new trend for integrating service robots into smart environments, such as homes or assembly lines of manufactories. Modern sensitive robots recognize human beings in their neighborhood and stop when touched, so that they can be integrated into their environment much better as in the past. Robots come out of the cage, and this creates a lot of opportunities for scalable automation in home and factory.

In the future home, service robots will help elderly and handicapped people. In manufacturing lines in small and medium enterprises, simple steps can be taught to a smart robot, while the difficult steps can be left to humans.

In both application areas, due to the safe integration into the smart environments, the investment costs for the use of robots sink considerably, and the degree of automation can be scaled accordingly. For industrial workshops, this new deployment model of sensitive robots will have a tremendous effect on all kinds of manufacture, because it shrinks the costs of robot-based automation and can be afforded by small companies.

Thus, entire industries could make use of robots that did not deploy them so far. For the smart home, this means that once a sensitive service robot costs less than 20kEuro, the support of elderly people with automation-based services might become affordable and economic.

[1] Technische Universität Dresden, Fakultät Informatik, Institut für Software- und Multimediatechnik, Lehrstuhl Softwaretechnologie, https://www.facebook.com/stdresden, http://st.inf.tu-dresden.de, uwe.assmann@tu-dresden.de

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 117

SE FIT: Software Engineering Forum der IT Transferinstitute

Michael Felderer[1], Wilhelm Hasselbring[2]

Software Engineering ist eine Ingenieursdisziplin, die von einem regen Austausch zwischen Wirtschaft und Wissenschaft profitiert. Es gibt deshalb einige inner- und außeruniversitäre Institute, die sich der Zusammenarbeit mit Unternehmen in Forschung und Entwicklung und dem Transfer von Wissen und Technologien verschrieben haben. Unternehmen und wissenschaftliche Einrichtungen profitieren von diesem Austausch gleichermaßen.

Das Forum SE FIT bietet Unternehmen, Wissenschaftlern und Transferinstituten die Gelegenheit zum Kennenlernen und zum Erfahrungsaustausch. Es soll als Plattform für den Wissens- und Technologietransfer im Bereich Software Engineering fungieren und die Kommunikation und Kooperation katalysieren.

SE FIT findet parallel zu den Workshops der SE 2016 am Dienstag, den 23. Februar 2016 und zur Hauptkonferenz am Mittwoch, den 24. Februar 2016 in Wien statt. An der Veranstaltung nehmen insgesamt 9 Transfereinrichtungen teil, nämlich Austrian Institute of Technology (AIT), fortiss, Fraunhofer-Institut für Experimentelles Software Engineering (IESE), Institut für Angewandte Informatik (InfAI), Kompetenzverbund Software Systems Engineering (KoSSE), Quality Engineering Competence Center QE Lab, SBA Research, Software Competence Center Hagenberg (SCCH) sowie Software Innovation Campus Paderborn (SICP).

Das Programm umfasst einen Impulsvortrag zur Forschungsförderung, die Vorstellung der einzelnen Institute sowie die Diskussion von Kooperationsmöglichkeiten. Weiters stellen die Institute im Rahmen einer Ausstellung ihr Profil und Leistungsangebot vor. Den Besuchern der Hauptkonferenz wird unter dem Motto „FIT for Lunch“ die Möglichkeit gegeben, diese Ausstellung im Rahmen der Mittagspause zu besuchen.

[1] Universität Innsbruck, QE LaB, Österreich, michael.felderer@uibk.ac.at
[2] Universität Kiel, KoSSE, Deutschland, hasselbring@email.uni-kiel.de

9. Arbeitstagung Programmiersprachen (ATPS 2016)

Andreas Krall[1], Ina Schaefer[2]

Die Arbeitstagung Programmiersprachen dient dem Austausch zwischen Forschern, Entwicklern und Anwendern in Hochschule, Wirtschaft und Industrie, die sich mit Themen aus dem Bereich der Programmiersprachen beschäftigen. Dabei sind alle Programmierparadigmen gleichermassen von Interesse: imperative, objektorientierte, funktionale, logische, parallele und graphische Programmiersprachen ebenso wie verteilte und nebenläufige Programmierung in Intra- und Internet-Anwendungen sowie Konzepte zur Integration dieser Paradigmen. Ebenfalls von Interesse sind Arbeiten zu Techniken, Methoden, Konzepten oder Werkzeugen, mit denen Sicherheit und Zuverlässigkeit bei der Ausführung von Programmen erhöht werden können.

Typische, aber nicht ausschliessliche Themenbereiche der Tagungsreihe 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
- Domainspezifische Sprachen
- Programm- und Implementierungsverifikation
- Werkzeuge und Programmierumgebungen
- Frameworks, Architekturen, generative Ansätze
- Erfahrungen bei exemplarischen Anwendungen
- Verbindung von Sprachen, Architekturen, Prozessoren

[1] TU Wien, Institut für Computersprachen, Argentinierstr. 8, 1040 Wien, Österreich, andi@complang.tuwien.ac.at
[2] TU Braunschweig, Institut für Softwaretechnik und Fahrzeuginformatik, Mühlenpfordtstr. 23, D-38106 Braunschweig, Deutschland, i.schaefer@tu-braunschweig.de

Neben neuen Arbeiten sind stets auch Beiträge erwünscht, die existierende Arbeiten oder Projekte zusammenfassen oder aus einem neuen Blickwinkel präsentieren, und sie so insbesondere einem deutschsprachigen Publikum vorstellen.

Bei der Zusammenstellung des Programmkomitees liegt ein besonderer Schwerpunkt in der Einbindung deutschsprachiger Wissenschaftler, die beruflich ausserhalb des deutschen Sprachraums tätig sind. Dies dient zum einen der Vergrösserung der wissenschaftlichen Basis und zum anderen der langfristigen Kontaktpflege und Kooperation zwischen deutschsprachigen Wissenschaftlern im In- und Ausland.

Programmkomitee

Walter Binder	USI Lugano, Schweiz
Michael Hanus	Univ. Kiel, Deutschland
Christian Heinlein	Hochschule Aalen, Deutschland
Andreas Krall	TU Wien, österrreich, Co-Vorsitzender
Welf Löwe	Linnaeus Univ. Växjö, Schweden
Gerald Lüttgen	Univ. Bamberg, Deutschland
Thomas Noll	RWTH Aachen, Deutschland
Markus Müller-Olm	Univ. Münster, Deutschland
Christian W. Probst	DTU Lyngby, Dänemark
Ina Schaefer	TU Braunschweig, Deutschland, Co-Vorsitzende
Volker Stolz	Univ. Oslo, Norwegen
Peter Thiemann	Univ. Freiburg, Deutschland
Janis Voigtländer	Univ. Bonn, Deutschland
Guido Wachsmuth	TU Delft, Niederlande
Baltasar Trancn Y Widemann	TU Ilmenau, Deutschland

CPSSC - 1st International Workshop on Cyber-Physical Systems in the Context of Smart Cities

Constantin Scheuermann[1], Andreas Seitz[2]

1 Motivation

The interconnection and exchange of information is a basic requirement to enable Smart Cities. Such an interconnection must be realized within each domain (vertically) as well as among different domains (horizontally). Typical domains cover energy, automotive, health, intelligent buildings, transportation, manufacturing as well as the military domain.

Therefore, horizontal and vertical information sharing within Smart Cities is essential and comes with challenges for software developers. Which models, architectures, patterns and data structures need to be developed to enable and further improve the interconnection and digitization of Smart Cities, are major questions that need to be addressed.

2 Description

This workshop is a forum for authors to present their early research findings in the field of Cyber-Physical Systems in the context of Smart Cities. The workshop aims at overviews, theoretical approaches, tools and frameworks, applications, system infrastructures and test beds for Cyber-Physical Systems. The following list depicts the main fields of interest:

- Theoretical foundations of Cyber-Physical Systems
- Smart City Applications
- Smart Environments
- Novel Industrial Applications of Cyber-Physical (Human) Systems
- Detailed Case Studies
- Infrastructure Applications for Smart Cities
- Security/Privacy
- Architecture and Modeling of Cyber-Physical Systems

[1] Technical University of Munich (TUM), Munich, Germany, constantin.scheuermann@in.tum.de
[2] Technical University of Munich (TUM), Munich, Germany, seitz@in.tum.de

3 Programm Committee and Organization

Programm Committee:

• Mathias Althoff, Technical University of Munich (TUM), Munich, Germany

• Bernd Bruegge, Technical University of Munich (TUM), Munich, Germany

• Michael Heiss, Siemens AG, Head of Research Group CPSs, Vienna, Austria

• Bruce Horn, Intel Cooperation, USA

• Hyun-Wook Jin, Konkuk University, Seoul, Korea

• Sung-Soo Lim, Kookmin University, Seoul, Korea

• Chi-Sheng Shih, National Taiwan University, Taipei, Taiwan

• Monika Sturm, Leibniz Universit¨at Hannover, Siemens AG, CPS Principal, Vienna, Austria

• Birgit Vogel-Heuser, Technical University of Munich (TUM), Munich, Germany

Organization:

• Constantin Scheuermann, Technical University of Munich (TUM), Munich, Germany

• Andreas Seitz, Technical University of Munich (TUM), Munich, Germany

4 Program

09:00-10:00	Keynote - Volker Hartkopf
10:00-10:30	Coffee Break
10:30-12:00	Presentations
12:00-14:00	Lunch Break
14:00-15:00	Invited Talk - Oliver Juli
15:00-15:30	Coffee Break
15:30-17:00	Panel Discussion: The Future of Smart Cities

5 Discussion Panel: The Future of Smart Cities

As part of the CPSSC Workshop a Panel about The Future of Smart Cities with well-known experts in the area of Smart Cities, Cyber-Physical Systems and Smart Office Environments is organized. The discussion will be moderated by Monika Sturm.

Workshop on Continuous Software Engineering

Horst Lichter[1], Bernd Brügge[2], Dirk Riehle[3]

In order to develop and deliver high-quality products to their customers, software companies have to adopt state-of-the-art software development processes. To face this challenge, companies are applying innovative methods, approaches and techniques like agile methods, DevOps, Continuous Delivery, test automation, infrastructure as code or container-based virtualization.

These new approaches have a high impact on the specification, design, development, maintenance, operation and the evolution of software systems. Therefore, common software engineering activities, organizational forms and processes have to be questioned, adapted and extended to ensure continuous and unobstructed software development (Continuous Software Engineering). So far, there is a lack of systematic approaches to face these challenges.

The goal of this workshop is to present and discuss innovative solutions, ideas and experiences in the area of Continuous Software Engineering (CSE).

The workshop aims to cover the following topics:

- DevOps & Release Engineering
- Approaches to Continuous Integration/Delivery/Deployment
- Infrastructure as Code
- Test Automation & Optimization
- Monitoring & Performance
- Security for DevOps
- Provisioning of Soft-ware & Infrastructure
- Application Virtualization with Container
- Engineering of Deployment Pipelines
- Quality & Metrics for DevOps
- Design for Scalability
- Organizational issues for CSE

[1] 1RWTH Aachen University, Research Group Software Construction, horst.lichter@swc.rwth-aachen.de
[2] Technische Universität München, Institut für Informatik / I1T, bruegge@in.tum.de
[3] Friedrich-Alexander-University Erlangen-Nürnberg, Open Source Research Group, dirk.riehle@fau.de

- Continuous Delivery for Requirements Engineering/Early Proto-typing
- Change Management - Handling user feed-back
- Teaching CSE approaches
- Software Architectures for CSE
- Microservices
- Software Development Lifecycle for CSE.

As we want to have contributions from industry and academia presented and discussed in the workshop, we asked for original and evaluated research as well as for papers describing novel ideas, identified challenges, and especially experience reports related to the workshop's theme.

The presented papers cover different topics of CSE like dedicated process models and their application in CSE, new architectural styles like microservices and their integration with existing methodologies, and approaches to improve DevOps in organizations.

Program Committee

Bernd Brügge	TU München
Willi Hasselbring	Universität Kiel
Martin Jung	develop group, Erlangen
Stephan Krusche	TU München
Horst Lichter	RWTH Aachen University
Christian Nester	Google Inc.
Dirk Riehle	FAU Nürnberg
Heinz-Josef Schlebusch	Kisters AG, Aachen
Christian Uhl	codecentric AG, Düsseldorf
Stefan Wagner	Universität Stuttgart
Heinz Züllighoven	WPS und Universität Hamburg

Workshop Organizers

Lukas Alperowitz	TU München
Andreas Steffens	RWTH Aachen University

Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 125

3nd Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS'16)

Robert Heinrich[1] Reiner Jung[2] Marco Konersmann[3] Eric Schmieders[3]

Langlebige software-intensive Systeme sind während ihrer Nutzung einer Vielzahl an Änderungen ihrer Anforderungen sowie ihres technologischen Kontextes ausgesetzt. Das kann unter anderem zu inkonsistenten Anforderungsspezifikationen, Architekturerosion und SLA-Verletzungen führen. Die Relevanz dieser Problematik ergibt sich vor allem in der industriellen Praxis, in der ein solches System nicht nur eine initiale Entwicklung erfährt, sondern ständig weiterentwickelt werden muss. Traditionelle Entwicklungsprozesse bieten bislang keine ausdrückliche Unterstützung von Langlebigkeit. Die Schnittstellen zwischen den Phasen wie Anforderungserhebung, Systemdesign und -entwicklung, sowie Betrieb sehen daher bislang noch kein systematisches Erfassen und Beschreiben ihrer Abhängigkeiten hinsichtlich Langlebigkeit vor.

In der Wissenschaft finden derzeit zahlreiche Bemühungen statt um die typische Entwicklungsphasen auf Langlebigkeit auszurichten. Häufig sind die Arbeiten jedoch auf einzelne Phasen fokussiert und lassen übergreifende Herausforderungen ausser Acht. Phasenübergreifende Probleme, wie z.B. das Einholen und Dokumentieren von Anforderungen zur Unterstützung von Selbstadaption eines langlebigen Systems, erfordern aber gerade das Erforschen des Zusammenspiels der unterschiedlichen Entwicklungsphasen ebenso wie ihrer Abhängigkeiten.

Ziel des dritten EMLS Workshops ist es, die Perspektiven der Forschung und der Industrie zusammenzubringen. Die Schwerpunkte des Workshops sind Problemstellungen, Lösungsansätze und Evaluationsansätze im Rahmen der Evolution und Wartung. Entlang der vorgestellten Beiträge sollen Kooperationsmöglichkeiten aufgedeckt werden, um so Forscher miteinander und Industrievertreter mit Forschern besser zu vernetzen und Synergien zu ermöglichen.

Der Beitrag "Challenges in Secure Software Evolution - The Role of Software Architecture" befasst sich mit den Herausforderungen der Evolution von Software bezogen auf Sicherheitseigenschaften und deren Modellierung. Es stellt dabei Herausforderungen und eine Lösungsmöglichkeit vor. Der Beitrag "Structured Model-Based Engineering of Long-living Embedded Systems: The SPES Methodological Building Blocks Framework" zeigt einen Ansatz um Entwicklungsprozesse für langlebige eingebetette Systeme zu definieren. Dabei geht das Papier auch auf den breiteren industriellen Kontext dieser Prozesse ein. Im Beitrag "Challenges in the Evolution of Metamodels" werden Herausforderungen

[1] Karlsruher Institut für Technologie, Am Fasanengarten 5, 76131 Karlsruhe, robert.heinrich@kit.edu
[2] Universität Kiel, Christian-Albrechts-Platz 4, 24118 Kiel, reiner.jung@email.uni-kiel.de
[3] Universität Duisburg-Essen, Gerlingstrasse 16, 45127 Essen, marco.konersmann@paluno.uni-due.de

aufgezeigt, die die Evolution von Meta-Modellen mit sich bringt. Der Beitrag basiert auf der Erfahrung mit dem Palladio Component Model, welches seit 2006 beständig weiter entwickelt wurde.

Die akzeptierten Beiträge werden im Verlauf des Workshops vorgestellt und diskutiert. In Kleingruppen sollen die Teilnehmer konkrete Ideen besprechen und ausarbeiten, sowie nächste Schritte für mögliche Kooperationen erarbeiten und im Workshop vorstellen. Die Ergebnisse sollen langfristig zu gemeinsamen Projekten, Publikationen, Technologien oder Benchmarks beitragen.

2nd Workshop on Fail Safety in Medical Cyber-Physical Systems (FS-MCPS)

Alexander Schlaefer and Sibylle Schupp and André Stollenwerk

{schlaefer,schupp}@tuhh.de, stollenwerk@embedded.rwth-aachen.de
http://www.aa4r.org/fs-mcps2016.html

Medical cyber-physical systems (MCPSs) extend the notion of conventional medical devices to more complex technical systems in close connection to humans, e.g., acquiring sensor data, controlling a treatment, or monitoring recovery. Typically, these systems include the patient in the loop and require a high degree of dependability and fail safety. One challenge is the complex nature of physiological processes, which are often patient-specific and less deterministic than in, e.g., engineering scenarios. Thus, many of the underlying interactions are today still not modeled in detail. Another challenge is the growing complexity of medical systems and devices themselves. Hence, fail safety of a MCPS cannot be achieved within a single component or layer — neither the software layer nor any other isolated layer —, but requires an interdisciplinary effort addressing different aspects, including patient modeling, hardware, software, and communication.

The workshop covers these aspects and discusses software-engineering issues of MCPS. Intended as a platform for interdisciplinary exchange the topics range from theoretical foundation of fail safety to actual applications of MCPS. Interoperability and integration of different devices has been an active research field, particularly as computer assistance for decision support and guidance of interventional procedures requires an aggregation of data provided by different systems. The design of interfaces and emerging standards for interoperability must consider the safe operation of the overall system. This includes meeting temporal constraints, e.g., when illustrating organ movements during image guidance or for automated motion compensation. Moreover, consistent, fail safe, and secure data exchange on the hardware and software level are essential for connecting devices, particularly when longer distances are covered, e.g., for remote and ambient assistance scenarios. The latter also require reliable network protocols and resource management. Architectures and algorithms for the interaction in a MCPS, which are capable of tolerating latency and package loss due to unstable connections, are desired. Furthermore, the implications of change at one point in a system to the connected remainder is addressed.

Clearly, the move towards more complex systems in clinical practice is gradual and requires compliance with existing regulations and integration with existing devices. Bringing together researchers and practitioners in the field of MPCS we summarize the state of the art and discuss obstacles and challenges on the way to fail-safe MCPS. We are grateful for the keynote talk by Wolfgang Reisig (Humboldt Universität zu Berlin) and the two tutorials on Polyspace and Uppaal by Christian Guss and Jakob Taankvist, who

present theoretical and practical aspects of software engineering and verification for medical applications. We would also like acknowledge the members of our program committee Sabine Glesner, Christian Hansen, Klaus Radermacher, Asarnusch Rashid, Wolfgang Reisig, Bernhard Rumpe, Stefan Schlichting, and Annette Stümpel.

Hamburg and Aachen, January 2016

Sibylle Schupp, Alexander Schlaefer, and André Stollenwerk

2. Workshop „Lehre für Requirements Engineering“ (LehRE)

Rüdiger Weißbach[1], Jörn Fahsel[2], Andrea Herrmann[3], Anne Hoffmann[4], Dieter Landes[5]

Abstract: LehRE ist ein Workshop über Lehre und Training für Requirements Engineering. Auf der SE2016 stehen Kompetenzorientierung und Agilität in der Lehre im Vordergrund, außerdem sollen Einsatzmöglichkeiten elektronischer Lehrplattformen in der RE-Lehre in einem Gastvortrag diskutiert werden.

Keywords: Kompetenzorientierung, Agile Lehre, elektronische Lernplattformen, digitale Lernplattformen

1 Zielsetzung des Workshops

LehRE ist ein Workshop über Lehre und Training für Requirements Engineering [RE]. Auf der SE2014 wurde dieser Workshop mit Teilnehmern aus dem akademischen und dem industriellen Umfeld erstmals durchgeführt. Der Erfahrungsaustausch zwischen Personen, die RE im Studium oder berufsbegleitend lehren, ist zentraler Inhalt des Workshops. Außerdem ist explizit die Teilnahme von Studierenden erwünscht, die über Lernerwartungen und Lernerfahrungen berichten sollen.

Der Workshop resultiert aus der Arbeit des Arbeitskreises „Requirements Engineering in der Lehre“ der Fachgruppe 2.1.6, Requirements Engineering (RE), der Gesellschaft für Informatik e.V. (GI). Die Organisatoren des Workshops haben alle langjährige Erfahrung im Requirements Engineering und dessen Vermittlung.

2 Inhalte

In dem Workshop werden Erfahrungsberichte aus der Ausbildung im Software Engineering und aus der Ausbildung von „Nicht-Informatikern“ im Requirements Engineering für die Zusammenarbeit mit Informatikern vorgestellt. Thomas Lehmann

[1] Hochschule für Angewandte Wissenschaften (HAW) Hamburg, Berliner Tor 5, 20099 Hamburg, ruediger.weissbach@haw-hamburg.de
[2] Friedrich-Alexander-Universtität Erlangen-Nürnberg, Katholischer Kirchenplatz 9, 91054 Erlangen, joern.fahsel@fau.de
[3] Freie Software Engineering Trainerin , Daimlerstr. 121, 70372 Stuttgart, herrmann@herrmann-ehrlich.de
[4] Siemens AG, Freyeslebenstr. 1, 91058 Erlangen, anne.hoffmann@siemens.com
[5] Hochschule für für angewandte Wissenschaften Coburg, Friedrich-Streib-Str. 2, 96450 Coburg, dieter.landes@hs-coburg.de

und Bettina Buth (HAW Hamburg) greifen aktuelle Entwicklungen in der Hochschuldidaktik auf, die die Vermittlung von Kompetenzen in den Vordergrund stellen und ihre Erfahrungen aus der RE-Vermittlung im Software Engineering mitteilen. Vera Kraus (FAU Erlangen-Nürnberg) ist Masterstudentin im Bereich Buchwissenschaften. Im Kontext der „Digitalisierung“ traditioneller Produktbereich erscheint die Fähigkeit der fachlichen Stakeholder, sich an dem RE-Prozess aktiv zu beteiligen bzw. diesen zu steuern, als neue Erfordernis. Kraus stellt ihre Lernerfahrungen aus einer Veranstaltung dar, in der die Studierenden aus konventionellen Buchprodukten crossmediale Produkte erzeugt haben. Den konzeptionellen Rahmen dieser Veranstaltung erläutert Jörn Fahsel (FAU Erlangen-Nürnberg) in seiner Darstellung des Konzepts einer „Agilen Lehre“, in der Methoden individualisiert und anforderungsgerecht kombiniert werden.

Neben diesen Erfahrungsberichten sind die Vorstellung eines Tools zur Unterstützung der Lehre und die Diskussion über dessen Einsatzmöglichkeiten in der RE-Ausbildung weitere Programmpunkte.

GI-Edition Lecture Notes in Informatics

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Lan-guage to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Business 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Er-fahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometrics and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment

P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society

P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications

P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven

P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik

P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004

P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration

P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Adhoc-Netze Medienintegration

P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government

P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen

P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality

P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Ranneberg, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments

P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für inforrmatische Bildung

P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen

P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit

P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and ist Applications

P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005

P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolffried Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web

P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik

P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)

P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)

P-69 Robert Hirschfeld, Ryszard Kowalcyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005

P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)

P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005

P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): "Heute schon das Morgen sehen"

P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology

P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture

P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz

P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006

P-78 K.-O. Wenkel, P. Wagner, M. Morgenstern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce

P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS´06

P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006

P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics

P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications

P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems

P-86 Robert Krimmer (Ed.): Electronic Voting 2006

P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik

P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006

P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur

P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006

P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006

P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1

P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2

P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen

P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies

P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices

P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik

P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik

P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten

P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments

P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)

P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme

P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007

P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops

P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications

P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures

P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1

P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2

P-111 Christian Eibl, Johannes Magenheim, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik

P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis

P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)

P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics

P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walther (Eds.) German conference on bioinformatics GCB 2007

P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007

P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and manegement The 4th International Conference on Grid Service Engineering and Management GSEM 2007

P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007

P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications

P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges

P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik

P-122 Walid Maalej, Bernd Bruegge (Hrsg.) Software Engineering 2008 - Workshopband Fachtagung des GI-Fachbereichs Softwaretechnik

P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)

P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Speciel Interest Groups PARS and PARVA

P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT:
Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung

P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)

P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008

P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany

P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems

P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung

P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting. CC

P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik

P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 1

P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 2

P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik –
Aktuelle Forschungsergebnisse

P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008

P-137 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures

P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008

P-139 Georg Herzwurm, Martin Mikusz (Hrsg.)
Industrialisierung des Software-Managements
Fachtagung des GI-Fachausschusses Management der 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 Informations-systeme (MobIS 2008)
Modellierung zwischen SOA und Compliance Management

P-142 R. Bill, P. Korduan, L. Theuvsen, M. Morgenstern (Hrsg.)
Anforderungen an die Agrarinformatik durch Globalisierung und Klimaveränderung

P-143 Peter Liggesmeyer, Gregor Engels, Jürgen Münch, Jörg Dörr, Norman Riegel (Hrsg.)
Software Engineering 2009
Fachtagung des GI-Fachbereichs Softwaretechnik

P-144 Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)
Datenbanksysteme in Business, Technologie und Web (BTW)

P-145 Knut Hinkelmann, Holger Wache (Eds.)
WM2009: 5th Conference on Professional Knowledge Management

P-146 Markus Bick, Martin Breunig, Hagen Höpfner (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Entwicklung, Implementierung und Anwendung
4. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2009)

P-147 Witold Abramowicz, Leszek Maciaszek, Ryszard Kowalczyk, Andreas Speck (Eds.)
Business Process, Services Computing and Intelligent Service Management
BPSC 2009 · ISM 2009 · YRW-MBP 2009

P-148 Christian Erfurth, Gerald Eichler, Volkmar Schau (Eds.)
9th International Conference on Innovative Internet Community Systems
I^{2}CS 2009

P-149 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
2. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-150 Jürgen Münch, Peter Liggesmeyer (Hrsg.)
Software Engineering
2009 - Workshopband

P-151 Armin Heinzl, Peter Dadam, Stefan Kirn, Peter Lockemann (Eds.)
PRIMIUM
Process Innovation for Enterprise Software

P-152 Jan Mendling, Stefanie Rinderle-Ma, Werner Esswein (Eds.)
Enterprise Modelling and Information Systems Architectures
Proceedings of the 3rd Int'l Workshop EMISA 2009

P-153 Andreas Schwill, Nicolas Apostolopoulos (Hrsg.)
Lernen im Digitalen Zeitalter
DeLFI 2009 – Die 7. E-Learning Fachtagung Informatik

P-154 Stefan Fischer, Erik Maehle Rüdiger Reischuk (Hrsg.)
INFORMATIK 2009
Im Focus das Leben

P-155 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2009:
Biometrics and Electronic Signatures
Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-156 Bernhard Koerber (Hrsg.)
Zukunft braucht Herkunft
25 Jahre »INFOS – Informatik und Schule«

P-157 Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, Peter Stadler (Eds.)
German Conference on Bioinformatics 2009

P-158 W. Claupein, L. Theuvsen, A. Kämpf, M. Morgenstern (Hrsg.)
Precision Agriculture Reloaded – Informationsgestützte Landwirtschaft

P-159 Gregor Engels, Markus Luckey, Wilhelm Schäfer (Hrsg.)
Software Engineering 2010

P-160 Gregor Engels, Markus Luckey, Alexander Pretschner, Ralf Reussner (Hrsg.)
Software Engineering 2010 – Workshopband
(inkl. Doktorandensymposium)

P-161 Gregor Engels, Dimitris Karagiannis Heinrich C. Mayr (Hrsg.)
Modellierung 2010

P-162 Maria A. Wimmer, Uwe Brinkhoff, Siegfried Kaiser, Dagmar Lück-Schneider, Erich Schweighofer, Andreas Wiebe (Hrsg.)
Vernetzte IT für einen effektiven Staat
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2010

P-163 Markus Bick, Stefan Eulgem, Elgar Fleisch, J. Felix Hampe, Birgitta König-Ries, Franz Lehner, Key Pousttchi, Kai Rannenberg (Hrsg.)
Mobile und Ubiquitäre Informationssysteme
Technologien, Anwendungen und Dienste zur Unterstützung von mobiler Kollaboration

P-164 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2010: Biometrics and Electronic Signatures Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-165 Gerald Eichler, Peter Kropf, Ulrike Lechner, Phayung Meesad, Herwig Unger (Eds.)
10th International Conference on Innovative Internet Community Systems (I^2CS) – Jubilee Edition 2010 –

P-166 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
3. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-167 Robert Krimmer, Rüdiger Grimm (Eds.)
4th International Conference on Electronic Voting 2010
co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-168 Ira Diethelm, Christina Dörge, Claudia Hildebrandt, Carsten Schulte (Hrsg.)
Didaktik der Informatik
Möglichkeiten empirischer Forschungsmethoden und Perspektiven der Fachdidaktik

P-169 Michael Kerres, Nadine Ojstersek Ulrik Schroeder, Ulrich Hoppe (Hrsg.)
DeLFI 2010 - 8. Tagung der Fachgruppe E-Learning der Gesellschaft für Informatik e.V.

P-170 Felix C. Freiling (Hrsg.)
Sicherheit 2010
Sicherheit, Schutz und Zuverlässigkeit

P-171 Werner Esswein, Klaus Turowski, Martin Juhrisch (Hrsg.)
Modellierung betrieblicher Informationssysteme (MobIS 2010)
Modellgestütztes Management

P-172 Stefan Klink, Agnes Koschmider Marco Mevius, Andreas Oberweis (Hrsg.)
EMISA 2010
Einflussfaktoren auf die Entwicklung flexibler, integrierter Informationssysteme
Beiträge des Workshops der GI-Fachgruppe EMISA (Entwicklungsmethoden für Informationssysteme und deren Anwendung)

P-173 Dietmar Schomburg, Andreas Grote (Eds.)
German Conference on Bioinformatics 2010

P-174 Arslan Brömme, Torsten Eymann, Detlef Hühnlein, Heiko Roßnagel, Paul Schmücker (Hrsg.)
perspeGKtive 2010
Workshop „Innovative und sichere Informationstechnologie für das Gesundheitswesen von morgen“

P-175 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 1

P-176 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 2

P-177 Witold Abramowicz, Rainer Alt, Klaus-Peter Fähnrich, Bogdan Franczyk, Leszek A. Maciaszek (Eds.)
INFORMATIK 2010
Business Process and Service Science – Proceedings of ISSS and BPSC

P-178 Wolfram Pietsch, Benedikt Krams (Hrsg.)
Vom Projekt zum Produkt
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschafts-informatik (WI-MAW), Aachen, 2010

P-179 Stefan Gruner, Bernhard Rumpe (Eds.)
FM+AM`2010
Second International Workshop on Formal Methods and Agile Methods

P-180 Theo Härder, Wolfgang Lehner, Bernhard Mitschang, Harald Schöning, Holger Schwarz (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW)
14. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS)

P-181 Michael Clasen, Otto Schätzel, Brigitte Theuvsen (Hrsg.)
Qualität und Effizienz durch informationsgestützte Landwirtschaft, Fokus: Moderne Weinwirtschaft

P-182 Ronald Maier (Hrsg.)
6th Conference on Professional Knowledge Management
From Knowledge to Action

P-183 Ralf Reussner, Matthias Grund, Andreas Oberweis, Walter Tichy (Hrsg.)
Software Engineering 2011
Fachtagung des GI-Fachbereichs Softwaretechnik

P-184 Ralf Reussner, Alexander Pretschner, Stefan Jähnichen (Hrsg.)
Software Engineering 2011
Workshopband
(inkl. Doktorandensymposium)

P-185 Hagen Höpfner, Günther Specht, Thomas Ritz, Christian Bunse (Hrsg.)
MMS 2011: Mobile und ubiquitäre Informationssysteme Proceedings zur 6. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2011)

P-186 Gerald Eichler, Axel Küpper, Volkmar Schau, Hacène Fouchal, Herwig Unger (Eds.)
11th International Conference on Innovative Internet Community Systems (I^2CS)

P-187 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
4. DFN-Forum Kommunikations-technologien, Beiträge der Fachtagung 20. Juni bis 21. Juni 2011 Bonn

P-188 Holger Rohland, Andrea Kienle, Steffen Friedrich (Hrsg.)
DeLFI 2011 – Die 9. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. 5.–8. September 2011, Dresden

P-189 Thomas, Marco (Hrsg.)
Informatik in Bildung und Beruf INFOS 2011
14. GI-Fachtagung Informatik und Schule

P-190 Markus Nüttgens, Oliver Thomas, Barbara Weber (Eds.)
Enterprise Modelling and Information Systems Architectures (EMISA 2011)

P-191 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2011
International Conference of the Biometrics Special Interest Group

P-192 Hans-Ulrich Heiß, Peter Pepper, Holger Schlingloff, Jörg Schneider (Hrsg.)
INFORMATIK 2011
Informatik schafft Communities

P-193 Wolfgang Lehner, Gunther Piller (Hrsg.)
IMDM 2011

P-194 M. Clasen, G. Fröhlich, H. Bernhardt, K. Hildebrand, B. Theuvsen (Hrsg.)
Informationstechnologie für eine nachhaltige Landbewirtschaftung
Fokus Forstwirtschaft

P-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 11th 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 Kommunikations-technologien
Beiträge der Fachtagung

P-204 Gerald Eichler, Leendert W. M. Wienhofen, Anders Kofod-Petersen, Herwig Unger (Eds.)
12th International Conference on Innovative Internet Community Systems (I2CS 2012)

P-205 Manuel J. Kripp, Melanie Volkamer, Rüdiger Grimm (Eds.)
5th International Conference on Electronic Voting 2012 (EVOTE2012)
Co-organized by the Council of Europe, Gesellschaft für Informatik and 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 Rumpe (Hrsg.)
Software Engineering 2013
Fachtagung des GI-Fachbereichs Softwaretechnik

P-214 Volker Markl, Gunter Saake, Kai-Uwe Sattler, Gregor Hackenbroich, Bernhard Mit schang, 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 Dreo 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-225 Hans-Georg Fill, Dimitris Karagiannis, Ulrich Reimer (Hrsg.)
Modellierung 2014
19. – 21. März 2014, Wien

P-226 M. Clasen, M. Hamer, S. Lehnert, B. Petersen, B. Theuvsen (Hrsg.)
IT-Standards in der Agrar- und Ernährungswirtschaft Fokus: Risiko- und Krisenmanagement
Referate der 34. GIL-Jahrestagung
24. – 25. Februar 2014, Bonn

P-227 Wilhelm Hasselbring,
Nils Christian Ehmke (Hrsg.)
Software Engineering 2014
Fachtagung des GI-Fachbereichs Softwaretechnik
25. – 28. Februar 2014
Kiel, Deutschland

P-228 Stefan Katzenbeisser, Volkmar Lotz, Edgar Weippl (Hrsg.)
Sicherheit 2014
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 7. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
19. – 21. März 2014, Wien

P-230 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2014
Proceedings of the 13th International Conference of the Biometrics Special Interest Group
10. – 12. September 2014 in Darmstadt, Germany

P-231 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
7. DFN-Forum Kommunikationstechnologien
16. – 17. Juni 2014
Fulda

P-232 E. Plödereder, L. Grunske, E. Schneider, D. Ull (Hrsg.)
INFORMATIK 2014
Big Data – Komplexität meistern
22. – 26. September 2014
Stuttgart

P-233 Stephan Trahasch, Rolf Plötzner, Gerhard Schneider, Claudia Gayer, Daniel Sassiat, Nicole Wöhrle (Hrsg.)
DeLFI 2014 – Die 12. e-Learning Fachtagung Informatik
der Gesellschaft für Informatik e.V.
15. – 17. September 2014
Freiburg

P-234 Fernand Feltz, Bela Mutschler, Benoît Otjacques (Eds.)
Enterprise Modelling and Information Systems Architectures
(EMISA 2014)
Luxembourg, September 25-26, 2014

P-235 Robert Giegerich,
Ralf Hofestädt,
Tim W. Nattkemper (Eds.)
German Conference on Bioinformatics 2014
September 28 – October 1
Bielefeld, Germany

P-236 Martin Engstler, Eckhart Hanser, Martin Mikusz, Georg Herzwurm (Hrsg.)
Projektmanagement und Vorgehensmodelle 2014
Soziale Aspekte und Standardisierung
Gemeinsame Tagung der Fachgruppen Projektmanagement (WI-PM) und Vorgehensmodelle (WI-VM) im Fachgebiet Wirtschaftsinformatik der Gesellschaft für Informatik e.V., Stuttgart 2014

P-237 Detlef Hühnlein, Heiko Roßnagel (Hrsg.)
Open Identity Summit 2014
4.–6. November 2014
Stuttgart, Germany

P-238 Arno Ruckelshausen, Hans-Peter Schwarz, Brigitte Theuvsen (Hrsg.)
Informatik in der Land-, Forst- und Ernährungswirtschaft
Referate der 35. GIL-Jahrestagung
23. – 24. Februar 2015, Geisenheim

P-239 Uwe Aßmann, Birgit Demuth, Thorsten Spitta, Georg Püschel, Ronny Kaiser (Hrsg.)
Software Engineering & Management 2015
17.-20. März 2015, Dresden

P-240 Herbert Klenk, Hubert B. Keller, Erhard Plödereder, Peter Dencker (Hrsg.)
Automotive – Safety & Security 2015
Sicherheit und Zuverlässigkeit für automobile Informationstechnik
21.–22. April 2015, Stuttgart

P-241 Thomas Seidl, Norbert Ritter, Harald Schöning, Kai-Uwe Sattler, Theo Härder, Steffen Friedrich, Wolfram Wingerath (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2015)
04. – 06. März 2015, Hamburg

P-242 Norbert Ritter, Andreas Henrich, Wolfgang Lehner, Andreas Thor, Steffen Friedrich, Wolfram Wingerath (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW 2015) – Workshopband
02. – 03. März 2015, Hamburg

P-243 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
8. DFN-Forum Kommunikationstechnologien
06.–09. Juni 2015, Lübeck

P-244 Alfred Zimmermann, Alexander Rossmann (Eds.) Digital Enterprise Computing (DEC 2015) Böblingen, Germany June 25-26, 2015

P-245 Arslan Brömme, Christoph Busch , Christian Rathgeb, Andreas Uhl (Eds.) BIOSIG 2015 Proceedings of the 14th International Conference of the Biometrics Special Interest Group 09.–11. September 2015 Darmstadt, Germany

P-246 Douglas W. Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt (Hrsg.) INFORMATIK 2015 28.9.-2.10. 2015, Cottbus

P-247 Hans Pongratz, Reinhard Keil (Hrsg.) DeLFI 2015 – Die 13. E-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. (GI) 1.–4. September 2015 München

P-248 Jens Kolb, Henrik Leopold, Jan Mendling (Eds.) Enterprise Modelling and Information Systems Architectures Proceedings of the 6th Int. Workshop on Enterprise Modelling and Information Systems Architectures, Innsbruck, Austria September 3-4, 2015

P-249 Jens Gallenbacher (Hrsg.) Informatik allgemeinbildend begreifen INFOS 2015 16. GI-Fachtagung Informatik und Schule 20.–23. September 2015

P-250 Martin Engstler, Masud Fazal-Baqaie, Eckhart Hanser, Martin Mikusz, Alexander Volland (Hrsg.) Projektmanagement und Vorgehensmodelle 2015 Hybride Projektstrukturen erfolgreich umsetzen Gemeinsame Tagung der Fachgruppen Projektmanagement (WI-PM) und Vorgehensmodelle (WI-VM) im Fachgebiet Wirtschaftsinformatik der Gesellschaft für Informatik e.V., Elmshorn 2015

P-251 Detlef Hühnlein, Heiko Roßnagel, Raik Kuhlisch, Jan Ziesing (Eds.) Open Identity Summit 2015 10.–11. November 2015 Berlin, Germany

P-252 Jens Knoop, Uwe Zdun (Hrsg.) Software Engineering 2016 Fachtagung des GI-Fachbereichs Softwaretechnik 23.–26. Februar 2016, Wien

P-253 A. Ruckelshausen, A. Meyer-Aurich, T. Rath, G. Recke, B. Theuvsen (Hrsg.) Informatik in der Land-, Forst- und Ernährungswirtschaft Fokus: Intelligente Systeme – Stand der Technik und neue Möglichkeiten Referate der 36. GIL-Jahrestagung 22.-23. Februar 2016, Osnabrück

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