Report from Dagstuhl Seminar 13091

Analysis, Test and Verification in The Presence of Variability

Edited by

Paulo Borba[1], Myra B. Cohen[2], Axel Legay[3], and Andrzej Wąsowski[4]

1 **Federal University of Pernambuco – Recife, BR,** phmb@cin.ufpe.br
2 **University of Nebraska – Lincoln, US,** myra@cse.unl.edu
3 **University of Liège, BE,** alegay@irisa.fr
4 **IT University of Copenhagen, DK,** wasowski@itu.dk

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 13091 "Analysis, Test and Verification in The Presence of Variability". The seminar had the goal of consolidating and stimulating research on analysis of software models with variability, enabling the design of variability-aware tool chains. We brought together 46 key researchers from three continents, working on quality assurance challenges that arise from introducing variability, and some who do not work with variability, but that are experts in their respective areas in the broader domain of software analysis or testing research. As a result of interactions triggered by sessions of different formats, the participants were able to classify their approaches with respect to a number of dimensions that helped to identify similarities and differences that have already been useful to improve understanding and foster new collaborations among the participants.

Seminar 24. February to 1. March, 2013 – www.dagstuhl.de/13091
1998 ACM Subject Classification D.2.4 Software/Program Verification, D.2.5 Testing and Debugging, D.2.13 Reusable Software, D.3.1 Formal Definitions and Theory, F.3.1 Specifying and Verifying and Reasoning about Programs, F.3.2 Semantics of Programming Languages
Keywords and phrases Verification, Program Analysis, Testing, Semantics of Programming Languages, Software Engineering
Digital Object Identifier 10.4230/DagRep.3.2.144
Edited in cooperation with Leopoldo Teixeira

1 Executive Summary

Paulo Borba
Myra B. Cohen
Axel Legay
Andrzej Wąsowski

The seminar "Analysis, Test and Verification in The Presence of Variability" that took place at Schloss Dagstuhl from February 24 to March 1, 2013, had the goal of consolidating and stimulating research on analysis of software models with variability, enabling the design of variability-aware tool chains. We brought together 46 key researchers from three continents, working on quality assurance challenges that arise from introducing variability, and some who do not work with variability, but that are experts in their respective areas in the broader domain of software analysis or testing research. The participants ranged from those in senior academic positions to successful graduate students. We also enjoyed the presence of several relevant experts from the software development industry.

The seminar included:

1. Invited presentations on state of the art research in SPL testing and verification

The presentations were delivered by experts in variability research. The topics included classifying and unifying product-line analyses, combinatorial interaction testing, model-based testing, analysis of programs with variability and model checking with variability.

Material relevant to the topic of this Dagstuhl was organized in a recent classification by Thüm and coauthors [4]. The Dagstuhl seminar opened with a presentation of this classification, which created a common ontology for later presentations and discussions. This was very helpful for participants who had different areas of expertise.

2. A keynote presentation on the Challenges and Science of Variability

We organized a special keynote shared with the German FOSD meeting, that took place in parallel at the Schloss Dagstuhl facilities. The keynote speaker, Professor Don Batory, called for creating a simple meta-theory identifying and relating the core concepts and properties of variability science, i.e. the body of knowledge created by the community of researchers studying engineering of highly configurable systems. During the workshop, several candidates for the starting point of such theory were mentioned, such as using simple models in constructive logic [2], choice calculus [3] or Clafer [1].

3. A series of presentations on recent results in Variability Analysis

The bulk of the programme was filled with a mixture of research presentations about recent research advances in verification, analysis and test of software with variability. This function of the seminar was particularly important, as the usual dissemination outlets for these contributions are often disjoint – much of the work is normally presented in domain specific publication channels devoted to only test, verification or programming languages. For many participants the seminar created an opportunity to learn about advances at addressing similar problems in the neighboring research communities – an experience that is rarely possible outside of Dagstuhl.

4. A session of student presentations

In order to enrich the presentations by senior researchers with a stream of fresh ideas, we organized a special session devoted to short student presentations. The presenters were selected from the participants of the German FOSD meeting. For many of the students it was a rare opportunity to share their ideas with international authorities in their work area. The topics of these lightning presentations were closely related to the seminar goals and included among others, discussions of experimental evaluation of product line analysis strategies, static analysis, type checking for variability, and performance prediction for configurable systems. The session enabled closer integration between the participants of the two events. Many discussions between the two groups continued throughout the week.

5. Dynamically planned sessions on how to address the challenges, how to transfer knowledge, tools, and benchmarks between research areas

The first session (run by Professor Krzysztof Czarnecki) was devoted to extracting challenges for variability analysis out of industrial requirements. Participants from industry and participants from academia involved in industrial projects provided background on requirements known from projects in avionics, automotive and risk assessment domains. These were further discussed to identify research challenges for future work. The discussions were continued in a breakout session on product lines of safety critical systems. Other breakout sessions included dynamic product lines, generic representation of variability, and testing and modeling variability.

Overall, a core set of techniques were discussed at this seminar which include program analysis, model checking, type checking, and testing. We believe that the seminar fruitfully mixed computer science and software engineering researchers from several research subdomains, allowing them to derive interesting basic research problems stemming from practical needs all related to how variability impacts their respective domains, with the sub-goal of inspiring the use of the latest research advances in software analysis technology to advance variability management tools.

Results

The different kinds of interactions offered by the seminar helped the participants to relate work covering different aspects in a number of dimensions such as:

1. An overall approach to thinking about variability, as defined by Thüm's classification [4] of analysis into product based, family based, feature based and hybrids;
2. Core techniques: testing, verification, refactoring, model checking, static analysis;
3. Mechanisms for representing variability: if-defs, deltas, generic representation, etc.;
4. Application domains;
5. The nature of variability: static product lines, dynamic product lines, configurable systems.

The seminar also produced a bibliography of core readings on the topic, that can enable new graduate students to engage more quickly in this area of research.

Trying to classify approaches with respect to these dimensions helped to identify similarities and differences among different techniques (static analysis, model checking, testing, and verification). This, in turn, might trigger new collaborations and research results. The presentations and the ad-hoc discussion sessions helped people to clarify differences and similarities among configurable systems and dynamic and static product lines, with similar consequences to the ones described above. More generally, of course, the Dagstuhl provided the benefit of mixing young and experienced researchers, from different countries and research areas.

An informal survey among a handful of participants has shown that each of them have started 2-3 new collaborations as a result of the seminar. These collaborations took the form of initiated research papers, mutual research visits, or student exchanges. In one anecdotal case, a researcher started a collaboration with a colleague sitting in the same corridor at his home university— but apparently one had to meet in Dagstuhl to enable the exchange of ideas. We can thus expect a new wave of research results in this area to flourish about a year from the seminar time. Because of this success, we intend to organize a follow up event in several years, be it under the Schloss Dagstuhl programme or under some other appropriate venue.

References

1 Kacper Bąk, Krzysztof Czarnecki, and Andrzej Wąsowski. Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled. In *Proc. of the 3rd Int'l Conf. on Software Language Engineering (SLE'10)*, LNCS, Vol. 6563, pp. 102–122, Springer, 2011. DOI: 10.1007/978-3-642-19440-5_7.

2 Benjamin Delaware, William R. Cook, and Don S. Batory. Product lines of theorems. In *Proc. of the 2011 ACM Int'l Conf. on Object-oriented Programming Systems, Languages, and Applications (OOPSLA'11)*, pp. 595–608, ACM, 2011. DOI: 10.1145/2048066.2048113.

3 Martin Erwig and Eric Walkingshaw. The Choice Calculus: A Representation for Software Variation. *ACM Trans. Softw. Eng. Methodol.*, Vol. 21, Issue 1, pp. 6:1–6:27, 2011. DOI: 10.1145/2063239.2063245.

4 Thomas Thüm, Sven Apel, Christian Kästner, Martin Kuhlemann, Ina Schaefer, and Gunter Saake. Analysis Strategies for Software Product Lines. Technical Report FIN-004-2012, School of Computer Science, University of Magdeburg, April 2012. http://www.cs.uni-magdeburg.de/inf_media/downloads/forschung/technical_reports_und_preprints/2012/04_2012.pdf.

2 Table of Contents

3 Overview of Talks

3.1 Family and Sampling-based Reliability Analysis in Dynamic Software Product Line: the Body Area Network Case

Vander Alves (University of Brasilia, BR)

Demographic and social changes have increased the number of elderly people living alone. Many of these need continuous medical assistance, yet it is not sustainable to have dedicated medical professional for each of them. As a result, automated support has been proposed, in particular, Body Area Network, in which a person goes about his or her daily activities at home or outdoors, but wears sensors monitoring vital signs and providing emergency detection and prevention. Such systems often have to reconfigure themselves based on some context change such as the persons' medical situation to meet a new and more suitable quality goal for that new situation. However, current approaches provide limited support for reliability-aware dynamic adaptation. Accordingly, we explore how family- and sampling-based analysis in Dynamic Software Product Line (DSPL) support reliability-aware dynamic adaptation. First, we present a domain reliability model relying on a state machine whose transitions are medical events (e.g., fall, stroke) and states are target reliability goals, prompting a reconfiguration to meet them. Second, the reliability of any given configuration is measured by a single function over the features of the DSPL. This function is derived from a family-based analysis leveraging a parametric discrete time Markov chain model representing the reliability of the DSPL. Lastly, the configuration space is searched in a bounded product analysis to find suitable configurations meeting reliability goals.

3.2 Classifying and Unifying Product-Line Analyses

Sven Apel (Universität Passau, DE)

Joint work of von Rhein, Alexander; Apel, Sven; Kästner, Christian; Thüm, Thomas; Schaefer, Ina
Main reference A. von Rhein, S. Apel, C. Kästner, Thomas Thüm, and Ina Schaefer, "The PLA Model: On the Combination of Product-Line Analyses," in Proc. of the 7th Int'l Workshop on Variability Modelling of Software-intensive Systems (VaMoS'13), pp. 73—80, ACM, 2013.
URL http://dx.doi.org/10.1145/2430502.2430522
URL http://www.infosun.fim.uni-passau.de/publications/docs/RAK+13vamos.pdf

Product-line analysis has received considerable attention in the last decade. As it is often infeasible to analyze each product of a product line individually, researchers have developed analyses, called variability-aware analyses, that consider and exploit variability manifested in a code base. Variability-aware analyses are often significantly more efficient than traditional analyses, but each of them has certain weaknesses regarding applicability or scalability. We present the Product-Line-Analysis model, a model for the classification and comparison of existing analyses, including traditional and variability-aware analyses, and lay a foundation for formulating and exploring further, combined analyses. This talk is based on a number of previous publications [1, 2, 3].

References

1 Alexander von Rhein, Sven Apel, Christian Kästner, Thomas Thüm, and Ina Schaefer. The PLA Model: On the Combination of Product-Line Analyses. In *Proceedings of the International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 73–80. ACM, January 2013.

2 Christian Kästner and Sven Apel. Feature-Oriented Software Development. In *Generative and Transformational Techniques in Software Engineering IV*, volume 7680 of *Lecture Notes in Computer Science*, pages 346–382. Springer-Verlag, January 2013.

3 Thomas Thüm, Sven Apel, Christian Kästner, Martin Kuhlemann, Ina Schaefer, and Gunter Saake. Analysis Strategies for Software Product Lines. Technical Report FIN-004-2012, School of Computer Science, University of Magdeburg, April 2012.

3.3 Software Product Lines in Clafer

Kacper Bąk (University of Waterloo, CA)

Joint work of Bąk, Kacper; Czarnecki, Krzysztof; Wąsowski, Andrzej; Antkiewicz, Michal; Diskin, Zinovy; Liang, Jimmy; Olaechea, Rafael; Murashkin, Alexandr
Main reference K. Bąk, K. Czarnecki, A. Wąsowski, "Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled", in Proc. of the 3rd Int'l Conf. on Software Language Engineering (SLE'10), LNCS, Vol. 6563, pp. 102–122, Springer, 2011.
URL http://dx.doi.org/10.1007/978-3-642-19440-5_7
URL http://clafer.org

Clafer is a lightweight modeling language for modeling and analysis of software product lines. The talk presents Clafer and showcases its features, such as feature modeling, the constraint language, staged configuration, partial instances, and multi-objective optimization. It also discusses two controversial design choices: concept unification and type-instance unification.

3.4 Analyzing Software Product Lines in Minutes instead of Years

Eric Bodden (TU Darmstadt, DE)

Joint work of Bodden, Eric; Toledo, Tarsis; Ribeiro, Marcio; Brabrand, Claus; Borba, Paulo; Mezini, Mira
Main reference E.Bodden, T. Tolêdo, M. Ribeiro, C. Brabrand, P. Borba, M. Mezini, "SPL$^{\mathrm{LIFT}}$ – Statically Analyzing Software Product Lines in Minutes Instead of Years", in Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI'13), ACM, 2013.
URL http://dx.doi.org/10.1145/2491956.2491976
URL http://www.bodden.de/pubs/bmb+13spllift.pdf
URL http://www.bodden.de/2013/02/18/pldi-spllift/

A software product line (SPL) encodes a potentially large variety of software products as variants of some common code base. Up until now, re-using traditional static analyses for SPLs was virtually intractable, as it required programmers to generate and analyze all products individually. In this work, however, we show how an important class of existing inter- procedural static analyses can be transparently lifted to SPLs. Without requiring programmers to change a single line of code, our approach SPL$^{\mathrm{LIFT}}$ automatically converts any analysis formulated for traditional programs within the popular IFDS framework for inter- procedural, finite, distributive, subset problems to an SPL- aware analysis formulated in the IDE framework, a well-known extension to IFDS. Using a full implementation based on

Soot, CIDE and JavaBDD, we show that with SPL$^{\text{LIFT}}$ one can reuse IFDS-based analyses without changing a single line of code. Through experiments using three static analyses applied to four Java-based product lines, we were able to show that our approach produces correct results and outperforms the traditional approach by several orders of magnitude.

3.5 State of The Art in Analysis of Programs with Variability

Eric Bodden (TU Darmstadt, DE)

In this talk I will explain general principles behind static data-flow analysis, how data-flow analysis has been used to decide interesting properties about software product lines, and recent efforts on trying to lift existing static program analysis to software-product-line analyses.

3.6 Intraprocedural Dataflow Analysis for Software Product Lines

Claus Brabrand (IT University of Copenhagen, DK)

Joint work of Brabrand, Claus; Ribeiro, Márcio; Tolêdo, Társis; Winther, Johnni; Borba, Paulo
Main reference C. Brabrand, M. Ribeiro, T. Tolêdo, J. Winther, P. Borba, "Intraprocedural Dataflow Analysis for Software Product Lines", Transactions on Aspect-Oriented Software Development X, Vol. 10, pp. 73–108, LNCS, Vol. 7800, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-642-36964-3_3

Software product lines (SPLs) developed using annotative approaches such as conditional compilation come with an inherent risk of constructing erroneous products. For this reason, it is essential to be able to analyze such SPLs. However, as dataflow analysis techniques are not able to deal with SPLs, developers must generate and analyze all valid products individually, which is expensive for non-trivial SPLs.

In this talk, we demonstrate how to take any standard intraprocedural dataflow analysis and automatically turn it into a feature-sensitive dataflow analysis in five different ways where the last is a combination of the other four. All analyses are capable of analyzing all valid products of an SPL without having to generate all of them explicitly.

We have implemented all analyses using SOOT's intraprocedural dataflow analysis framework and experimentally evaluated four of them according to their performance and memory characteristics on five qualitatively different SPLs. On our benchmarks, the combined analysis strategy is up to almost eight times faster than the brute-force approach.

References

1 Claus Brabrand, Márcio Ribeiro, Társis Tolêdo, and Paulo Borba. Intraprocedural Dataflow Analysis for Software Product Lines. In *Proceedings of the 11th International Conference on Aspect- oriented Software Development (AOSD 2012)*, pages 13–24. ACM, Potsdam, Germany, 2012.

2 Claus Brabrand, Márcio Ribeiro, Társis Tolêdo, Johnni Winther, and Paulo Borba. Intraprocedural Dataflow Analysis for Software Product Lines. In *Transactions on Aspect-Oriented Software Development X*, vol. 10, pages 73–108, Springer, 2013.

3.7 If not interfaces, then views

Dave Clarke (KU Leuven, BE)

Features tend to cross-cut a software product line's code base, and thus providing an interface for them is difficult, if not impossible. For many applications, such as design and modelling of SPLs and some analysis tasks, an alternative is makes more sense. The idea is to provide a view of a software product line by abstracting away irrelevant details. Numerous formalisms for SPLs are based on so-called super-imposed variants, wherein all products of the SPL are captured within the same semantic model. Notions of abstraction for these models should be used to produce views of features or feature combinations. This talk advocated a comprehensive study of views, where no sensible notion of interface exists, in order to simplify reasoning about SPLs.

3.8 On a Feature-Oriented Characterization of Exception Flows in Software Product Lines

Roberta Coelho (Federal University of Rio Grande do Norte, BR)

Joint work of Coelho, Roberta; Melo, Hugo; Kulesza, Uira
Main reference H. Melo, R. Coelho, U. Kulesza, "On a Feature-Oriented Characterization of Exception Flows in Software Product Lines," in Proc. of the 26th Brazilian Symp. on Software Engineering (SBES'12), IEEE, 2012.
URL http://dx.doi.org/10.1109/SBES.2012.15

The Exception Handling (EH) is a widely used mechanism for building robust systems. In Software Product Line (SPL) context it is not different. As EH mechanisms are embedded in most of mainstream programming languages, we can find exception signalers and handlers spread over code assets associated to common and variable SPL features. When exception signalers and handlers are added to an SPL in an unplanned way, one of the possible consequences is the generation of faulty products (i.e., products on which common or variable features signal exceptions that are mistakenly caught inside the system). This talk reports a first systematic study, based on manual inspection and static code analysis, in order to (i) categorize the possible ways exceptions flow in SPLs, and (ii) analyze its consequences. Fault-prone exception flows were consistently detected during this study; such as flows on which a variable feature signaled an exception a different and unrelated variable feature handled it. The talk is based on some previous publications [1, 2, 3].

References

1 Hugo Melo, Roberta Coelho, Uirá Kulesza On a Feature-Oriented Characterization of Exception Flows in Software Product Lines. 26th Brazilian Symposium on Software Engineering (SBES), August 2012.

2 Roberta Coelho, Awais Rashid, Uirá Kulesza, Arndt and von Staa, Carlos Lucena Unveiling and taming liabilities of aspects in the presence of exceptions: A static analysis based approach. In *Information Sciences*, 181, no. 13, pages 2700–2720. 2011

3 Roberta Coelho, Awais Rashid, Alessandro Garcia, Fabiano Ferrari, Nélio Cacho, Uirá Kulesza, Arndt and von Staa, Carlos Lucena Assessing the impact of aspects on exception flows: An exploratory study. In *Proceedings of ECOOP 2008– European Conference on Object-Oriented Programming*, pages 207–234. 2008.

3.9 ProVeLines: a Product Line of Model Checkers for Software Product Lines and some of the Theory behind it

Maxime Cordy (University of Namur, BE)

Joint work of Cordy, Maxime; Classen, Andreas; Heymans, Patrick; Schobbens, Pierre-Yves; Legay, Axel
Main reference A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, J.-F. Raskin, “Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and their Application to LTL Model Checking,” IEEE Transactions on Software Engineering, 2013.
URL http://dx.doi.org/10.1109/TSE.2012.86
URL http://info.fundp.ac.be/fts

Software Product Lines (SPLs) are families of similar software products built from a common set of features. As the number of products of an SPL is potentially exponential in the number of its features, the model checking problem is harder than for single software. A practical way to face this exponential blow-up is to reuse common behaviour between products. We previously introduced Featured Transition Systems, a mathematical model from which serves as a basis for efficient SPL model checking techniques. Here, we present ProVeLines, a product line of verifiers for SPLs that incorporates the results of over three years of research on formal verification of SPLs. Being itself a product line, our tool is flexible and extensible, and offers a wide range of solutions for SPL modelling and verification.

3.10 Toward the Systematic Derivation of Variational Program Analyses

Martin Erwig (Oregon State University, US)

Joint work of Erwig, Martin; Walkingshaw, Eric
Main reference M. Erwig, E. Walkingshaw, “Variation Programming with the Choice Calculus”, in Proc. of the Int’l Summer School on Generative and Transformational Techniques in Software Engineering IV (GTTSE’11), LNCS, Vol. 7680, pp. 55–100, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-642-35992-7_2
URL http://eecs.oregonstate.edu/ erwig/papers/abstracts.html#GTTSE12

In this presentation I will illustrate some basic principles for the systematic derivation of variational program analyses from non-variational ones. The approach is based on the view of an analysis as a function f that maps programs from an object language L to elements of some type T that represents the possible results of the analysis. The method proceeds in a number of small steps. First, the syntax of L is extended to a language VL that can represent variational programs. Then, in a similar way, the result type T is extended to a type VT to represent variational results. The language extension is realized through the choice calculus, which provides a generic representation for variation in (tree-structured) software artifacts. Because of its generic structure, the choice calculus can essentially be used as a variational type constructor V, and this makes the first two language extension steps systematic. Finally, f is lifted into a variational analysis by employing a number of simple transformations. Here it is the fact that the variational type constructor V is a monad that makes much of the lifting systematic, because the complexity involved in lifting f can be captured in many cases through an application of the monadic bind operation.

3.11 Topologically configurable systems as product families

Alessandro Fantechi (University of Firenze, IT)

Main reference A. Fantechi, "Topologically configurable systems as product families", to appear in the Proc. of the 17th Int'l Software Product Line Conf., Tokyo, August 2013.

We address a category of systems whose deployment requires a configuration according to topological information. Actually, this study is inspired by the case of railway interlocking systems, but gives a general definition of topologically configurable control systems. We consider the application of product line engineering principles to the development of these systems, e.g. by discussing the adoption of different approaches to achieve a flexible configuration of products, that allow factorising most of the design effort, as typical in a product line approach.

Things become more complex when there is a need of analysing the behaviour of such systems, either by testing or by formal verification: the intricate relations between the actual topology controlled by a product and its functional requirements may prevent any attempt to factorise analysis activities.

Indeed, in the cited case of the interlocking systems, the heavy verification and certification activities required by the safety regulations make a large part of the software development costs. Every newly configured product needs to undergo such certification activities, and little is saved from certifications made for previously deployed systems, with significant costs for each new installation.

We will discuss how a product line approach can help, with special focus on formal verification, showing that several research issues are still to be investigated in this direction.

3.12 Patterns in Configuration Dependence

Brady J. Garvin (University of Nebraska – Lincoln, US)

Joint work of Garvin, Brady J.; Cohen, Myra B.; Dwyer, Matthew B.

For configuration-aware testing and analysis techniques to effectively exploit whitebox knowledge, it is essential that the mapping of configurability to source code be precise. Unfortunately, as systems grow, less and less of the mapping is syntactically explicit; configuration information may flow through non- configuration data and control dependencies, ultimately rendering code dead in seemingly unrelated places. These dependency chains can grow quite long, and the heavyweight analyses needed to track the relevant properties may not scale to the whole-program level. I will give some examples in my talk.

Using a combination of static and dynamic analysis, we have established the configuration dependence in some small subjects exactly, in order to look for common patterns. Following earlier work, we express reachability of a block either as a CNF formula over atoms that represent configuration options, or else the complement of such a formula, and, thus far, the clauses tend to be extremely short (usually one literal, or, very rarely, two), with unique clauses across all basic blocks being few. In addition to some illustrative data, I will also overview results from previous studies that corroborate these findings.

If these observations hold up in other settings, and we can reasonably assume these patterns in larger systems, then we can drastically shrink the family of possible configuration dependence formulae and therefore the analysis effort needed to identify the formula for a particular basic block.

3.13 Making Software Product Line Evolution Safer

Rohit Gheyi (Federal University of Campina Grande, BR)

Developers evolve software product lines (SPLs) manually or using typical program refactoring tools. However, when evolving a product line to introduce new features or to improve its design, it is important to make sure that the behavior of existing products is not affected. Typical program refactorings cannot guarantee that because the SPL context goes beyond code and other kinds of core assets, and involves additional artifacts such as feature models and configuration knowledge. Besides that, in a SPL we typically have to deal with a set of possibly alternative assets that do not constitute a well-formed program. As a result, manual changes and existing program refactoring tools may introduce behavioral changes or invalidate existing product configurations. To avoid that, we propose approaches and implement tools for making product line evolution safer; these tools check whether SPL transformations are refinements in the sense that they preserve the behavior of the original SPL products. This talk is based on some previous publications [1, 2].

References

1 Paulo Borba, Leopoldo Teixeira, and Rohit Gheyi. A theory of software product line refinement. *Theoretical Computer Science*, 455:2 – 30, 2012.

2 Gustavo Soares, Rohit Gheyi, Tiago Massoni. Automated behavioral testing of refactoring engines. *IEEE Transactions on Software Engineering*, 39: 147–162, 2013.

3.14 Reuse of Formal Verification Proofs By Abstract Method Calls

Reiner Hähnle (TU Darmstadt, DE)

Joint work of Hähnle, Reiner; Schaefer, Ina; Bubel, Richard
Main reference R. Hähnle, I. Schaefer, R. Bubel, "Reuse in Software Verification by Abstract Method Calls," in Proc. of the 24th Conf. on Automated Deduction (CADE'13), LNCS, Vol. 7898, pp. 300–314, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-642-38574-2_21

Modern software tends to undergo frequent requirement changes and typically is deployed in many different scenarios. This poses significant challenges to formal software verification, because it is not feasible to verify a software product from scratch after each change. It is essential to perform verification in a modular fashion instead. The goal must be to reuse not merely software artifacts, but also specification and verification effort.

In our setting code reuse is realized by delta-oriented programming, an approach where a core program is gradually transformed by code "deltas" each of which corresponds to a product feature. The delta-oriented paradigm is then extended to contract-based formal specifications and to verification proofs. As a next step towards modular verification we transpose Liskov's behavioural subtyping principle to the delta world. Finally, based on the resulting theory, we perform a syntactic analysis of contract deltas that permits to automatically factor out those parts of a verification proof that stays valid after applying a code delta. This is achieved by a novel verification paradigma called "abstract verification".

3.15 Complete and Reproducible Applications of SPL Testing

Martin Fagereng Johansen (University of Oslo, NO)

Joint work of Johansen, Martin Fagereng; Haugen, Øystein; Fleurey, Franck; Carlson, Erik; Endresen, Jan; Wien, Tormod
Main reference M.F. Johansen, Ø. Haugen, F. Fleurey, E. Carlson, J. Endresen, T. Wien, "A Technique for Agile and Automatic Interaction Testing for Product Lines", in Proc. of the 24th IFIP WG 6.1 Int'l Conf. on Testing Software and Systems (ICTSS'12), LNCS, Vol. 7641, pp. 39–54, Springer, 2012.
URL http://dx.doi.org/10.1007/978-3-642-34691-0_5

Product line engineering is an inherently large scale effort; thus, most product lines are closed source, proprietary and not free. Understanding and verifying the performance of product line testing techniques benefits from a complete application of it that can be reproduced freely and easily. As resource material for a recent paper, we did provide two such complete and reproducible applications, available freely; one small application using CVL and model driven engineering, and a second large application with the Eclipse IDEs using the Eclipse plug-in system and ordinary textual programming. A quick overview of these two examples will be given with views on the benefits of the availability of complete and reproducible applications of SPL testing.

3.16 Analyzing the #ifdef hell with TypeChef – Or the quest for realistic subjects in product-line analysis

Christian Kästner (Carnegie Mellon University – Pittsburgh, US)

Joint work of Kästner, Christian; Apel, Sven; Berger, Thorsten; Erdweg, Sebastian; Giarrusso, Paolo; Liebig, Joerg; Ostermann, Klaus; von Rhein, Alexander; Rendel, Tillmann

In recent years, work on analysis of configurable systems has exploded. I and many others have investigated how we can make analysis of entire product lines faster, for example type check all configurations of a program annotated with features. The community has come up with many approaches to speed up analyses (type checking, model checking, static analysis, parsing, and others) by orders of magnitude compared to a brute-force approach [20].

We have claimed that analyses of entire product lines are necessary, because there is an exponentially exploding number of configurations. We have claimed that one could not feasibly check every configuration in isolation. We have claimed that industrial product lines typically have hundreds or thousands of configuration options and more configurations than there are atoms in the universe. We have claimed that analysis is critical, because otherwise users, who configure the systems, run into problems late in the development process when problems are expensive to fix.

When we proposed analysis mechanisms, our evaluations did not really align with our claims. For example, in our work on type checking in CIDE [7], we implemented the checks in a research environment and checked three systems with less than 15 configuration options and one system with 42 configuration options—far away from “more than atoms in the universe”. Given the fact that many of our arguments are empirical (SAT solvers, sharing in typical applications), results may not immediately generalize. In fact, analyzable subject systems are rare. The few systems we had, we shared. The graph product line [15] has served the community well as a canonical example, but it’s tiny. MobileMedia [6] is a common candidate but also rather limited in size and generalizability. The collected applications in CIDE, FeatureHouse [1], and the collected feature models in SPLOT [16] are a great start, but all relatively small and mostly stemming from student projects. We knew that there are lot’s of large industrial product lines, but we could not get our hands on them.

With envy, we have been looking at evaluation subjects from the testing community. Real-world large-scale systems, such as MySQL [23] and GCC [4, 22] with hundreds of configuration options tested and bugs found. For feature modeling, the Kconfig model of the Linux kernel turned out to be a great source for insights [17, 3].

To demonstrate that our analyses can scale to real-world problems and find real-world bugs in actual product lines, we searched for larger subject systems. Even though potentially not really product lines, we found that many C systems are highly configurable at compile time through their use of #ifdef directives and the C preprocessor. And the good news was that there are many openly available C systems with active developer communities, who might care about our results.

In initial studies, we found that most open-source systems that we looked at contain a massive amount of variability through the preprocessor [12]. Unfortunately, the way that the preprocessor is used (not always but often enough) makes it hard to parse the code without preprocessing it [13]. That meant, if we wanted to analyze that code in a precise way, and after all we intended to perform at least type checking without running into many false negatives, we would need to build our own infrastructure.

We decided to go after the Linux kernel, which was previously already investigated in the community regarding it’s feature model and dead code [17, 3, 19, 18]. That was the birth of the TypeChef project (initially short for “type checking #ifdef code”). Over the following years, we developed a lexer and parser that could actually process unpreprocessed C code in a sound and complete way [9]. On top, we have built various analyses, which by now form an interesting ecoystem. As of writing, TypeChef includes parsers for GNU C and Java, a type system for C, linker checks, control-flow graphs, and intra-procedural data-flow analysis, first steps toward refactoring engines and interpreters and sampling algorithms; and several more are currently planned and part of ongoing work.

TypeChef is now able to handle the x86 architecture of the Linux kernel (9 million lines of code, 6000 configuration options), Busybox (250 thousand lines of code, 800 configuration options) and we are currently looking into several other systems including OpenSSL, openVPN, BerkeleyDB, Apache, ChibiOS, and vim. We have found and reported several configuration-specific type errors already. We share the necessary scaffolding for these projects as well, which makes it easy to use and extend TypeChef and build other analyses.

While ecosystem and community around TypeChef is thriving, TypeChef is not without limitations. As any analysis of C code it struggles with C dialects and extensions. The parser is unnecessarily slow for mere technical reasons and we do not offer unparsing yet. Setting up analysis for a new system is difficult, because build paths, build system, and feature model need to be reverse engineered—a task where the community has helped us a lot with

Linux [2, 17, 18] and a reason why we publish all our scaffolding to setups easier for others.

Overall, I have seen a great spirit in this community that has frequently shared subject systems such as the graph product line [15], MobileMedia [6], and ArgoUML [5], and I have tried to contribute my own with CIDE and now TypeChef. As a community, we should foster and expand this sharing. Project collecting subject systems, such as first SPLOT for feature models and now SPL2go for product-line implementations are great. Currently, I'd be interested in such a repository for product lines with specifications or test cases...

Acknowledgments. TypeChef was only possible through great collaborations across several research groups, including direct contributors to the project and many colleagues who have helped with the infrastructure around it. I deeply appreciate the help of Sven Apel, Thorsten Berger, Sebastian Erdweg, Paolo G. Giarrusso, Steffen Haase, Andy Kenner, Joerg Liebig, Sarah Nadi, Klaus Ostermann, Alexander von Rhein, Tillmann Rendel, and Reinhard Tartler.

Further Reading on TypeChef. An in-depth discussion of the parsing approach and our experience with parsing Linux was published at OOPSLA 2011 [9]. In the context of a variability-aware module system, we discussed type checking and linker checks on the example of Busybox, published at OOPSLA 2012 [10]. A more detailed discussion of the variability-aware lexer (or partial preprocessor) was presented at VaMoS 2011 [8]. More information on the performance of our type system and data-flow analysis can be found in a technical report [14]. A simple variability-aware interpreter for executing test cases was build on top of TypeChef and published at FOSD 2012 [11]. For an overview of variability-aware analysis in general, please refer to the corresponding reports [20, 21] on the following webpage http://fosd.net/spl-strategies.

References

1 S. Apel, C. Kästner, and C. Lengauer. Language-independent and automated software composition: The FeatureHouse experience. *IEEE Transactions on Software Engineering (TSE)*, 2012. in press.

2 T. Berger, S. She, K. Czarnecki, and A. Wąsowski. Feature-to-code mapping in two large product lines. In *Proc. Int'l Software Product Line Conference (SPLC)*, pages 498–499. Springer-Verlag, 2010.

3 T. Berger, S. She, R. Lotufo, A. Wąsowski, and K. Czarnecki. Variability modeling in the real: A perspective from the operating systems domain. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*, pages 73–82. ACM Press, 2010.

4 M. B. Cohen, M. B. Dwyer, and J. Shi. Interaction testing of highly-configurable systems in the presence of constraints. In *Proc. Int'l Symp. Software Testing and Analysis (ISSTA)*, pages 129–139. ACM Press, 2007.

5 M. V. Couto, M. T. Valente, and E. Figueiredo. Extracting software product lines: A case study using conditional compilation. In *Proc. European Conf. on Software Maintenance and Reengineering (CSMR)*, pages 191–200. IEEE Computer Society, 2011.

6 E. Figueiredo et al. Evolving software product lines with aspects: An empirical study on design stability. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 261–270. ACM Press, 2008.

7 C. Kästner, S. Apel, T. Thüm, and G. Saake. Type checking annotation-based product lines. *ACM Trans. Softw. Eng. Methodol. (TOSEM)*, 21(3):Article 14, 2012.

8 C. Kästner, P. G. Giarrusso, and K. Ostermann. Partial preprocessing of C code for variability analysis. In *Proc. Int'l Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 137–140. ACM Press, 2011.

9 C. Kästner, P. G. Giarrusso, T. Rendel, S. Erdweg, K. Ostermann, and T. Berger. Variability-aware parsing in the presence of lexical macros and conditional compilation.

In *Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, pages 805–824. ACM Press, Oct. 2011.

10 C. Kästner, K. Ostermann, and S. Erdweg. A variability-aware module system. In *Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. ACM Press, 2012.

11 C. Kästner, A. von Rhein, S. Erdweg, J. Pusch, S. Apel, T. Rendel, and K. Ostermann. Toward variability-aware testing. In *Proc. GPCE Workshop on Feature-Oriented Software Development (FOSD)*, pages 1–8, 2012.

12 J. Liebig, S. Apel, C. Lengauer, C. Kästner, and M. Schulze. An analysis of the variability in forty preprocessor-based software product lines. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 105–114. ACM Press, 2010.

13 J. Liebig, C. Kästner, and S. Apel. Analyzing the discipline of preprocessor annotations in 30 million lines of C code. In *Proc. Int'l Conf. Aspect-Oriented Software Development (AOSD)*, pages 191–202. ACM Press, 2011.

14 J. Liebig, A. von Rhein, C. Kästner, S. Apel, J. Dörre, and C. Lengauer. Large-scale variability-aware type checking and dataflow analysis. Technical report, Department of Informatics and Mathematics, University of Passau, 2012.

15 R. Lopez-Herrejon and D. Batory. A standard problem for evaluating product-line methodologies. In *Proc. Int'l Conf. Generative and Component-Based Software Engineering (GCSE)*, volume 2186 of *Lecture Notes in Computer Science*, pages 10–24. Springer-Verlag, 2001.

16 M. Mendonca, M. Branco, and D. Cowan. S.p.l.o.t.: Software product lines online tools. In *Proc. Int'l Conf. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, pages 761–762. ACM Press, 2009.

17 S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki. The variability model of the Linux kernel. In *Proc. Int'l Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 45–51. University of Duisburg-Essen, 2010.

18 R. Tartler, D. Lohmann, J. Sincero, and W. Schröder-Preikschat. Feature consistency in compile-time-configurable system software: Facing the Linux 10,000 feature problem. In *Proc. European Conference on Computer Systems (EuroSys)*, pages 47–60. ACM Press, 2011.

19 R. Tartler, J. Sincero, W. Schröder-Preikschat, and D. Lohmann. Dead or alive: Finding zombie features in the Linux kernel. In *Proc. GPCE Workshop on Feature-Oriented Software Development (FOSD)*, pages 81–86. ACM Press, 2009.

20 T. Thüm, S. Apel, C. Kästner, M. Kuhlemann, I. Schaefer, and G. Saake. Analysis strategies for software product lines. Technical Report FIN-004-2012, School of Computer Science, University of Magdeburg, Apr. 2012.

21 A. von Rhein, S. Apel, C. Kästner, T. Thüm, and I. Schaefer. The pla model: On the combination of product-line analyses. In *Proceedings of the 7th Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, pages 14:1–14:8, 1 2013.

22 X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in C compilers. In *Proc. Conf. Programming Language Design and Implementation (PLDI)*, pages 283–294. ACM Press, 2011.

23 C. Yilmaz. Test case-aware combinatorial interaction testing. *IEEE Trans. Softw. Eng. (TSE)*, PP(99):1–1, 2012.

3.17 Is medical underwriting knowledge a field for verification in the presence of variability?

Kim Lauenroth (adesso AG – Dortmund, DE)

Medical underwriting is a part of the application process and refers to the assessment of an applicant for insurance coverage (e.g. life or health insurances). Many insurance companies use software systems to automate the medical underwriting process.

The necessary assessment knowledge for these systems is specified by:

(a) a large set of medical variables including weight, height, body mass index (bmi), type of diabetes;
(b) different questions to elicit medical variables, *e.g.* "What is your height (cm)?" or "What is your weight (kg)?";
(c) rules that trigger additional questions, *e.g.* if (height > 200) then ask "Do you suffer from back pain?";
(d) rules that lead to a risk decision (if (type of diabetes == 2 AND bmi >30) then reject application).

Variables, rules, and questions vary in several dimensions, for example, between different insurance products, and between countries or regions of the world. Quality assurance of this assessment knowledge is a constant challenge for insurance companies, since the knowledge is modified regularly because of new medical research results and new insurance products.

Insurance companies typically perform the quality assurance of their assessment knowledge based on different sample test cases but do not use automated techniques such as model checking. This talk has two goals:

1. Give an introduction into the field of medical underwriting and into the structure of the assessment knowledge;
2. Discuss possible strategies for the automated quality assurance of assessment knowledge with the seminar participants.

3.18 Scientific Workflows: Eternal Components, Changing Interfaces, Varying Compositions

Tiziana Margaria (Universität Potsdam, DE)

Joint work of Margaria, Tiziana; Lamprecht, Anna-Lena
Main reference A.-L. Lamprecht, T. Margaria, "Scientific workflows: eternal components, changing interfaces, varying compositions," in Proc. of the 5th Int'l Conf. on Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change – Part I (ISoLA'12), LNCS, Vol. 7609, pp. 47–63, Springer, 2012.
URL http://dx.doi.org/10.1007/978-3-642-34026-0_5

We describe how scientific application domains are characterized by the long-term availability of the basic computational components, and how software systems for managing the actual scientific workflows must deal with changing service interfaces and varying service compositions. In this light, we explain how rigorous technical and semantic abstraction, which is key to dealing with huge and heterogeneous application domains in an "extreme model driven design" framework like the jABC, supports the management of workflow evolution. We illustrate the different aspects by means of examples and experiences from the application of the framework in different scientific application domains.

3.19 Compositional Verification of Software Product Lines

Jean-Vivien Millo (INRIA Sophia Antipolis – Méditerranée, FR)

License Creative Commons BY 3.0 Unported license
© Jean-Vivien Millo
Joint work of Millo, Jean-Vivien; Ramesh, S; Sankara Narayanan, Krishna; Narwane Kandhu, Ganesh
Main reference J.-V. Millo, S. Ramesh, K. Sankara Narayanan, G. Narwane Kandhu, "Compositional Verification of Software Product Lines," in Proc. of the 10th Int'l Conf. on Integrated Formal Methods (IFM'13), LNCS, Vol. 7940, pp. 109–123, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-642-38613-8_8
URL http://hal.archives-ouvertes.fr/hal-00747533/

This work presents a novel approach to the design verification of Software Product Lines (SPL). The proposed approach assumes that the requirements and designs at the feature level are modeled as finite state machines with variability information. The variability information at the requirement and design levels are expressed differently and at different levels of abstraction. Also the proposed approach supports verification of SPL in which new features and variability may be added incrementally. Given the design and requirements of an SPL, the proposed design verification method ensures that every product at the design level behaviourally conforms to a product at the requirement level. The conformance procedure is compositional in the sense that the verification of an entire SPL consisting of multiple features is reduced to the verification of the individual features. The method has been implemented and demonstrated in a prototype tool SPLEnD (SPL Engine for Design Verification) on a couple of fairly large case studies.

3.20 Feature Maintenance with Emergent Interfaces

Marcio Ribeiro (Federal University of Pernambuco – Recife, BR)

License Creative Commons BY 3.0 Unported license
© Marcio Ribeiro
Joint work of Ribeiro, Marcio; Pacheco, Humberto; Teixeira, Leopoldo; Borba, Paulo
Main reference M. Ribeiro, H. Pacheco, L. Teixeira, P. Borba. "Emergent Feature Modularization," in In Onward!, affiliated with ACM SIGPLAN Int'l Conf. on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH'10), pp. 11–18, ACM, 2012.
URL http://dx.doi.org/10.1145/1869542.1869545

Hidden code dependencies are responsible for many complications in maintenance tasks. With the introduction of variable features in product lines, dependencies may even cross feature boundaries and related problems are prone to be detected late. Many current implementation techniques for product lines lack proper interfaces, which could make such dependencies explicit. As alternative to changing the implementation approach, we introduce a tool-based solution to support developers in recognizing and dealing with feature dependencies: emergent interfaces. Emergent interfaces are computed on demand, based on feature-sensitive interprocedural dataflow analysis. They emerge in the IDE and emulate benefits of modularity not available in the host language.

References

1 M. Ribeiro, H. Pacheco, L. Teixeira, and P. Borba. Emergent Feature Modularization. In *Onward!, affiliated with ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH)*, pages 11–18, New York, NY, USA, 2010. ACM.

2 M. Ribeiro, F. Queiroz, P. Borba, T. Tolêdo, C. Brabrand, and S. Soares. On the impact of feature dependencies when maintaining preprocessor-based software product lines. In *Proc. of the 10th ACM International Conference on Generative Programming and Component Engineering (GPCE)*, pages 23–32, Portland, Oregon, USA, 2011. ACM.

3 C. Brabrand, M. Ribeiro, T. Tolêdo, J. Winther, and P. Borba. Intraprocedural dataflow analysis for software product lines. *Transactions on Aspect-Oriented Software Development (TAOSD)*, 10:73–108, 2013.

3.21 Delta-oriented Regression-based Testing of Software Product Lines

Ina Schaefer (TU Braunschweig, DE)

Joint work of Schaefer, Ina; Malte Lochau; Sascha Lity
Main reference M. Lochau, I. Schaefer, J. Kamischke, S. Lity, "Incremental Model-Based Testing of Delta-Oriented Software Product Lines," in Proc. of the 6th Int'l Conf. on Tests and Proofs (TAP'12), LNCS, Vol. 7305, pp. 67–82, Springer, 2012.
URL http://dx.doi.org/10.1007/978-3-642-30473-6_7

Software architecture specifications are of growing importance for coping with the complexity of large-scale systems. They provide an abstract view on the high-level structural system entities together with their explicit dependencies and build the basis for ensuring behavioral conformance of component implementations and interactions, e.g., using model-based integration testing. The increasing inherent diversity of such large-scale variant-rich systems further complicates quality assurance. In this article, we present a combination of architecture-driven model-based testing principles and regression-inspired testing strategies for efficient, yet comprehensive variability-aware conformance testing of variant-rich systems. We propose an integrated delta-oriented architectural test modeling and testing approach for component as well as integration testing that allows the generation and reuse of test artifacts among different system variants. Furthermore, an automated derivation of retesting obligations based on accurate delta-oriented architectural change impact analysis is provided. Based on a formal conceptual framework that guarantees stable test coverage for every system variant, we present a sample implementation of our approach and an evaluation of the validity and efficiency by means of a case study from the automotive domain.

3.22 Model-Based Testing of Software Product Lines

Holger Schlingloff (HU Berlin, DE)

We give an introduction to software product lines as defined in the literature, share our view on model-based testing, and survey some recent work in the model-based testing of software product lines.

3.23 Reuse of Test Cases for Model-Based Development of Software Product Lines

Holger Schlingloff (HU Berlin, DE)

Main reference T. Kahsai, M. Roggenbach, B.-H. Schlingloff, "Specification-Based Testing for Software Product Lines," in Proc. of the 6th IEEE Int'l Conf. on Software Engineering and Formal Methods (SEFM'08), pp. 149–158, IEEE CS, 2008.
URL http://dx.doi.org/10.1109/SEFM.2008.38
URL http://www2.informatik.hu-berlin.de/~hs/Publikationen/2008_SEFM_Kahsai-Roggenbach-Schlingloff_Specification-based-Testing-of-Product-Lines.pdf

In a model-based development process of a software product line, the feature model determines which functions are present in each instance. The functional model elaborates the features to be transformed into an implementation. Both feature model and functional model are constantly enhanced and extended to allow for the incorparation of new features and functionalities. A frequent problem in this process is to decide which test cases can be reused from one instance to the next one. New features can extend the existing ones, or be in conflict with them. Thus, some test cases remain valid for the new product, while others have to be reworked. In this talk, we formalize these notions and show how to decide which parts of a test suite can be reused in a controlled product line development.

3.24 Test Case Prioritization Criteria for Software Product Lines

Sergio Segura (University of Sevilla, ES)

Testing all the products of a software product line is usually unfeasible, there are simply too many. To address this problem, contributions in the last years has focused on obtaining a representative subset of the products to be tested. But, once we have selected a set of products, is the order in which they are tested relevant? We think so. This is what test case prioritization techniques are about. For example, testers may wish to order their test cases in order to achieve code coverage at the fastest rate possible, employ features in order of expected frequency of use or increase the rate of fault detection of test cases. Our preliminary results show that the order in which test cases are run may have a significant impact in the rate of fault detection.

3.25 Composing Variable Components Considering Interaction Behavior – A Problem Statement

Vanessa Stricker (Universität Duisburg – Essen, DE)

Current approaches for integrating variability into hierarchical component structures neglect the behavioral specification of the component hierarchy. Though, in single system development it is acknowledged that early verification, ensuring that a component composition behaves as intended, is crucial. Especially the interaction between the components might result in unexpected deviations and inconsistencies between the actual and intended behavior.

Detecting these inconsistencies in composed component behavior is even more important, but also more complicated, in the presence of variability. The analysis of composed behavior has to identify if variable components can safely be composed as well as how the components have to be configured (i.e. binding of variability) in a composition in order to behave as intended. This is in particular important since a component's behavior might be restricted by its relations to other components. The influence of variability onto these behavior restrictions needs to be analyzed carefully and cannot be considered in isolation — especially if the variability of the components is described decentralized in multiple variability models, e.g. for 3rd party components or supplier product lines. To enable a rigorous analysis and reasoning about the behavioral correctness of variable component compositions, existing component-oriented design and analysis methods need to be extended to be able to consider variability. Using existing approaches of the formal methods community in an engineering method for composing variable components has to address certain challenges: There is a large set of existing formal methods in single system development and software product line engineering, which focus on verifying specific properties of component behaviors while having certain assumptions and restrictions. Therefore, the question arises whether these approaches can be used in an engineering approach, how they might have to be adapted and combined and how the results can be interpreted and exploited.

3.26 Safe Evolution of Software Product Lines and Communities

Leopoldo Teixeira (Federal University of Pernambuco – Recife, BR)

Joint work of Teixeira, Leopoldo; Borba, Paulo; Gheyi, Rohit
Main reference P. Borba, L. Teixeira, R. Gheyi, "A theory of software product line refinement," Theoretical Computer Science, Vol. 455, pp. 2–30, 2010.
URL http://dx.doi.org/10.1016/j.tcs.2012.01.031

Software Product Line evolution can benefit from refactorings with formal basis, to ensure correctness by construction. In this talk, we present a language independent theory of product line refinement, establishing refinement properties that justify stepwise and compositional product line evolution [1]. Instead of dealing directly with the stronger notion of refactoring, we focus on refinement, which also captures behavior preservation but abstracts quality improvement. We take the broader view of refinement as a relation that preserves properties necessary to assure safe evolution. The theory also supports reasoning about sets of product lines that define communities. To illustrate one of the practical applications of the theory, we introduce and prove soundness of a number of refinement transformation templates [2], that range from evolving individual artifacts to the product line as a whole, and sets of product lines. These templates can be used as the basis to derive comprehensive product line refinement catalogues. We use the Prototype Verification System to encode and prove soundness of the theory and associated properties and templates.

References

1 Paulo Borba, Leopoldo Teixeira, and Rohit Gheyi. A theory of software product line refinement. *Theoretical Computer Science*, 455:2–30, 2012.

2 Laís Neves, Leopoldo Teixeira, Demóstenes Sena, Vander Alves, Uirá Kulezsa, and Paulo Borba. Investigating the safe evolution of software product lines. In *Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering (GPCE'11)*, pages 33–42, 2011.

3.27 Product-Line Verification with Contracts

Thomas Thüm (Universität Magdeburg, DE)

Main reference T. Thüm, "Verification of Software Product Lines Using Contracts," in Doktorandentagung Magdeburger-Informatik-Tage (MIT), pp. 75–82, University of Magdeburg, 2012.
URL http://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen/ps/auto/Th:MIT12.pdf

Software product lines challenge existing specification and verification techniques known from single-system engineering. Specifying and verifying each product separately involves redundant effort and is usually not feasible. A promising technique is to specify and verify product lines based on design by contract. We apply generative programming to contracts, and generate contracts for each product based on domain artifacts. Furthermore, we analyze whether all products fulfill their contracts using model checking, static analysis, and deductive verification. Finally, we discuss pitfalls and benefits of design by contract for product-line verification.

References

1 S. Apel, A. von Rhein, T. Thüm, and C. Kästner. Feature-Interaction Detection based on Feature-Based Specifications. *Computer Networks*, 2013. To appear.
2 F. Benduhn. Contract-Aware Feature Composition. Bachelor's thesis, University of Magdeburg, Germany, 2012.
3 W. Scholz, T. Thüm, S. Apel, and C. Lengauer. Automatic Detection of Feature Interactions using the Java Modeling Language: An Experience Report. In *Proc. Int'l Workshop Feature-Oriented Software Development (FOSD)*, pages 7:1–7:8. ACM, 2011.
4 T. Thüm. Verification of Software Product Lines Using Contracts. In *Doktorandentagung Magdeburger-Informatik-Tage (MIT)*, pages 75–82. University of Magdeburg, 2012.
5 T. Thüm, S. Apel, C. Kästner, M. Kuhlemann, I. Schaefer, and G. Saake. Analysis Strategies for Software Product Lines. Technical Report FIN-004-2012, School of Computer Science, University of Magdeburg, Germany, 2012.
6 T. Thüm, I. Schaefer, S. Apel, and M. Hentschel. Family-Based Deductive Verification of Software Product Lines. In *Proc. Int'l Conf. Generative Programming and Component Engineering (GPCE)*, pages 11–20. ACM, 2012.
7 T. Thüm, I. Schaefer, M. Kuhlemann, and S. Apel. Proof Composition for Deductive Verification of Software Product Lines. In *Proc. Int'l Workshop Variability-intensive Systems Testing, Validation and Verification (VAST)*, pages 270–277. IEEE, 2011.
8 T. Thüm, I. Schaefer, M. Kuhlemann, S. Apel, and G. Saake. Applying Design by Contract to Feature-Oriented Programming. In *Proc. Int'l Conf. Fundamental Approaches to Software Engineering (FASE)*, volume 7212 of *LNCS*, pages 255–269. Springer, 2012.

3.28 Evaluating Dataflow Analysis for Software Product Lines

Tarsis Tolêdo (Federal University of Pernambuco – Recife, BR)

Joint work of Brabrand, Claus; Ribeiro, Márcio; Tolêdo, Tarsis; Borba, Paulo
Main reference C. Brabrand, M. Ribeiro, T. Tolêdo, P. Borba, "Intraprocedural dataflow analysis for software product lines," in Proc. of the 11th Annual Int'l Conf. on Aspect-oriented Software Development (AOSD'12), pp. 13–24, ACM, 2012.
URL http://dx.doi.org/10.1145/2162049.2162052

Brabrand et al. [1] proposed four different approaches to perform feature-sensitive intraprocedural dataflow analysis, which avoids having to explicitly generate each variant of a method before analyzing it. We evaluated these four approaches on four different software product lines and learned that each approach behaves differently depending on different characteristics of each method, like method size, size of lattice domain, number of confluence points; and also on different characteristics of the variability constructs, such as number, size and position of the #ifdef statements. In this talk, we discuss the confounding factors and possible strategies for a better evaluation of the approaches.

References

1 Claus Brabrand, Márcio Ribeiro, Társis Tolêdo, and Paulo Borba. Intraprocedural Dataflow Analysis for Software Product Lines. In *Proceedings of the 11th International Conference on Aspect- oriented Software Development (AOSD 2012)*, pages 13–24. ACM, Potsdam, Germany, 2012.

3.29 Inferring Variational Types for Variational Programs

Eric Walkingshaw (Oregon State University, US)

Joint work of Chen, Sheng; Erwig, Martin; Walkingshaw, Eric
Main reference S. Chen, M. Erwig, E. Walkingshaw, "An Error-Tolerant Type System for Variational Lambda Calculus," in Proc. of the ACM SIGPLAN Int'l Conf. on Functional Programming (ICFP'12), pp. 29–40, ACM, 2012.
URL http://dx.doi.org/10.1145/2364527.2364535
URL http://eecs.oregonstate.edu/~erwig/ToSC/VLC-TypeSystem.pdf

I presented a type system and type inference algorithm for variational lambda calculus (VLC). VLC extends the lambda calculus with a simple construct for representing variation points as choices between alternatives. Each choice is associated with a dimension and all choices in the same dimension are synchronized. The type of a VLC expression is a correspondingly variational type. VLC and variational types are two different instantiations of the choice calculus, making the extension of the Damas-Milner algorithm systematic, and giving us many components of the type system and inference algorithm "for free".

References

1 S. Chen, M. Erwig, and E. Walkingshaw. Extending Type Inference to Variational Programs. (Journal paper, in minor revision). 2013.
2 S. Chen, M. Erwig, and E. Walkingshaw. An Error- Tolerant Type System for Variational Lambda Calculus. *ACM SIGPLAN Int. Conf. on Functional Programming*, pages 29–40, 2012.

3.30 Combinatorial Interaction Testing

Cemal Yilmaz (Sabanci University – Istanbul, TR)

Main reference C. Yilmaz, "Test Case-Aware Combinatorial Interaction Testing", IEEE Transactions on Software Engineering, Vol. 39, No. 5, pp. 684–706, 2013.
URL http://dx.doi.org/10.1109/TSE.2012.65

The configuration spaces of modern software systems are too large to test exhaustively. Combinatorial interaction testing (CIT) approaches, such as covering arrays, systematically sample the configuration space and test only the selected configurations. Given a configuration space model, a t-way covering array is a set of configurations in which each valid combination of option settings for every combination of t options appears at least once. This talk has two parts. In the first part, I provide a brief survey of CIT approaches, including t-way covering arrays, system-wide inter-option constraints, seeding mechanisms, and variable strength covering arrays. In the second part, I introduce two novel combinatorial objects for testing, namely test case-aware covering arrays and cost-aware covering arrays. Traditional covering arrays, while taking system-wide inter-option constraints into account, do not provide a systematic way of handling test case-specific inter-option constraints. Thus they suffer from masking effects – unaccounted test case- specific constraints perturb program executions so as to prevent some option- related behaviors from being exercised. On the other hand, test case-aware covering arrays account for test case-specific constraints while constructing covering sets. A t-way test case-aware covering array is not just a set of configurations as is the case in traditional covering arrays, but a set of configurations, each of which is associated with a set of test cases, such that all system-wide and test case-specific constraints are satisfied and that, for each test case, each valid combination of option settings for every combination of t options appears at least once in the set of configurations that the test case is associated with. Another downside of traditional covering arrays is that they simply assume that every configuration costs the same. We, however, argue that this is often not the case in practice. Cost aware-covering arrays take actual cost of testing into account when computing covering sets. A t-way cost-aware covering array is a t- way covering array that minimizes a given cost function.

3.31 Effective Test Execution for Software Product Lines

Sabrina de Figueirêdo Souto (Federal University of Pernambuco – Recife, BR)

Joint work of de Figueirêdo Souto, Sabrina; d'Amorim, Marcelo
URL http://www.cin.ufpe.br/~sfs/souto-dagstuhl2013-talk.pdf

Software Product Lines (SPLs) have gained significant attention recently as an approach to improve productivity in development of software families. Considering that the number of valid products in one SPLs can be very high, one important practical problem is to speedup testing. This paper evaluates Symbolic Execution of Features (SEF) to address this problem. It is widely known that general symbolic execution is expensive compared to concrete execution. The efficiency of SEF comes from its ability to partition the state in two parts: one symbolic and one concrete. The symbolic part models the features which need to be enabled in the system along the analysis of one path while the concrete part models the entire program state. In SEF, the symbolic data does not flow to the stores associated to the concrete parts of the state; hence, operations that elaborate the program state perform efficiently as they only manipulate concrete data. Feasibility checking of paths is achieved with efficient incremental sat solving of feature constraints.

3.32 Modelling, analysing and verifying variability by means of Modal Transition Systems

Maurice H. ter Beek (CNR – Pisa, IT)

Joint work of ter Beek, Maurice H.; Fantechi, Alessandro; Gnesi, Stefania; Mazzanti, Franco; Asirelli, Patrizia
Main reference P. Asirelli, M.H. ter Beek, S. Gnesi, A.Fantechi, "Formal Description of Variability in Product Families," in Proc. of the 15th Int'l Software Product Line Conference (SPLC'11), pp. 130–139, IEEE, 2011.
URL http://dx.doi.org/10.1109/SPLC.2011.34

Our aim is a unifying logical framework for modelling, analysing and verifying behavioural variability in product families, with tool support for formal verification. We use Modal Transition Systems (MTSs) for modelling the behaviour and v-ACTL, a suitable action-based branching-time temporal logic interpreted over MTSs, for the necessary additional constraints. Our tool (VMC) accepts a product family specified as an MTS, possibly with additional variability constraints. Subsequently, VMC can be used for the derivation, exploration and analysis (from 'family-based' to 'product-based' analyses) of the family and its products, including the efficient verification of temporal logic properties through on-the-fly model checking. This talk is based on a number of previous publications [1, 2, 3].

References

1 Patrizia Asirelli, Maurice H. ter Beek, Alessandro Fantechi, and Stefania Gnesi. *A Logical Framework to Deal with Variability*. In Proceedings 8th International Conference on Integrated Formal Methods (IFM 2010). LNCS 6396, Springer, 2010, 43–58.
2 Patrizia Asirelli, Maurice H. ter Beek, Alessandro Fantechi, and Stefania Gnesi. *Formal Description of Variability in Product Families*. In Proceedings 15th International Software Product Line Conference (SPLC 2011). IEEE, 2011, 130–139.
3 Maurice H. ter Beek, Franco Mazzanti, and Aldi Sulova. *VMC: A Tool for Product Variability Analysis*. In Proceedings 18th International Symposium on Formal Methods (FM 2012). LNCS 7436, Springer, 2012, 450–454.

Participants

- Vander Alves
University of Brasilia, BR
- Sven Apel
Universität Passau, DE
- Joanne M. Atlee
University of Waterloo, CA
- Kacper Bąk
University of Waterloo, CA
- Don Batory
University of Texas at Austin, US
- Thorsten Berger
IT Univ. of Copenhagen, DK
- Eric Bodden
TU Darmstadt, DE
- Paulo Borba
University of Pernambuco – Recife, BR
- Claus Brabrand
IT Univ. of Copenhagen, DK
- Dave Clarke
KU Leuven, BE
- Andreas Classen
University of Namur, BE
- Roberta Coelho
Federal University of Rio Grande do Norte, BR
- Myra Cohen
Univ. of Nebraska – Lincoln, US
- Maxime Cordy
Facultés Universitaires Notre-Dame de la Paix, BE
- Krzysztof Czarnecki
University of Waterloo, CA
- Sabrina de Figueirêdo Souto
University of Pernambuco – Recife, BR
- Martin Erwig
Oregon State University, US
- Alessandro Fantechi
University of Firenze, IT
- Brady J. Garvin
Univ. of Nebraska – Lincoln, US
- Rohit Gheyi
Federal University of Campina Grande, BR
- Stefania Gnesi
CNR – Pisa, IT
- Reiner Hähnle
TU Darmstadt, DE
- Øystein Haugen
SINTEF – Oslo, NO
- Martin Fagereng Johansen
University of Oslo, NO
- Christian Kästner
Carnegie Mellon University – Pittsburgh, US
- Shriram Krishnamurthi
Brown University, US
- Kim Lauenroth
adesso AG – Dortmund, DE
- Axel Legay
INRIA Bretagne Atlantique – Rennes, FR
- Martin Leucker
Universität Lübeck, DE
- Tiziana Margaria
Universität Potsdam, DE
- Dusica Marijan
Simula Reseach Laboratory – Lysaker, NO
- Jean-Vivien Millo
INRIA Sophia Antipolis – Méditerranée, FR
- Gilles Perrouin
University of Namur, BE
- Márcio Ribeiro
University of Pernambuco – Recife, BR
- Ina Schaefer
TU Braunschweig, DE
- Holger Schlingloff
HU Berlin, DE
- Sergio Segura
University of Sevilla, ES
- Vanessa Stricker
Univ. Duisburg–Essen, DE
- Leopoldo Teixeira
University of Pernambuco – Recife, BR
- Maurice H. ter Beek
CNR – Pisa, IT
- Thomas Thüm
Universität Magdeburg, DE
- Társis Tolêdo
University of Pernambuco – Recife, BR
- Salvador Trujillo
Ikerlan Research Centre – Arrasate-Mondragón, ES
- Eric Walkingshaw
Oregon State University, US
- Andrzej Wąsowski
IT Univ. of Copenhagen, DK
- Cemal Yilmaz
Sabanci Univ. – Istanbul, TR