Report from Dagstuhl Seminar 19191

Software Evolution in Time and Space: Unifying Version and Variability Management

Edited by

Thorsten Berger[1], Marsha Chechik[2], Timo Kehrer[3], and Manuel Wimmer[4]

1 **Chalmers and University of Gothenburg, SE,** thorsten.berger@chalmers.se
2 **University of Toronto, CA,** chechik@cs.toronto.edu
3 **HU Berlin, DE,** timo.kehrer@informatik.hu-berlin.de
4 **Johannes Kepler Universität Linz, AT,** manuel.wimmer@jku.at

Abstract

Effectively managing versions and variants of software systems are among the main challenges of software engineering. Over the last decades, two large research fields, Software Configuration Management (SCM) and Software Product Line Engineering (SPLE), have focused on addressing the version and the variant management, respectively. Yet, large-scale systems require addressing both challenges in a unified way. The SCM community regularly faces the need to support variants, while SPLE needs versioning support. However, neither community has been successful in producing unified version and variant management techniques that are effective in practice. This seminar aimed at establishing a body of knowledge of version and variant management techniques. Together with industrial practitioners, we invited researchers from both fields to conceive an ontology of SCM and SPLE concepts, to identify open problems, and to elicit and synthesize practitioners' challenges and requirements. These outcomes provided the basis to create a research agenda, research infrastructure, and working groups, and finally, to establish a benchmark for evaluating future research results. As such, the seminar enabled research on enhanced version and variant management techniques that will ultimately be adopted in practice.

Seminar May 5–10, 2019 – http://www.dagstuhl.de/19191

2012 ACM Subject Classification Software and its engineering → Software configuration management and version control systems, Software and its engineering → Software product lines, Software and its engineering → Empirical software validation

Keywords and phrases software configuration management, versioning, variability management, software product lines, empirical evaluation

Digital Object Identifier 10.4230/DagRep.9.5.1

Edited in cooperation with Wardah Mahmood

Software Evolution in Time and Space: Unifying Version and Variability Management, *Dagstuhl Reports*, Vol. 9, Issue 5, pp. 1–31
Editors: Thorsten Berger, Marsha Chechik, Timo Kehrer, and Manuel Wimmer

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Executive Summary

Thorsten Berger
Marsha Chechik
Timo Kehrer
Manuel Wimmer

Overview and Motivation

Modern software systems evolve rapidly and often need to exist in many variants. Consider the Linux kernel with its uncountable number of variants. Each variant addresses different requirements, such as runtime environments ranging from Android phones to large super-computers and server farms. At the same time, the Linux kernel frequently boasts new versions, managed by thousands of developers. Yet, software versions–resulting from evolution in time–and variants–resulting from evolution in space–are managed radically differently. Version management relies on a version control system (Git) and sophisticated workflows–concepts that have been developed for decades in the field of software configuration management (SCM) [12, 24, 23]. Variant management in the Linux kernel relies on techniques known from the field of software product line engineering (SPLE) [27, 11, 13], such as an integrated software platform, a variant-aware build system [7], an interactive configurator tool [30], and a model-based representation [9, 8, 17, 1] of all kernel features [4, 28]. The Linux kernel is exemplary for many large-scale, variant-rich, and rapidly evolving software systems in industry [5, 3, 32], especially in the domains of embedded, cyber-physical, automotive, and avionics control systems.

Despite decades of research in both fields, the effective evolution of variant-rich systems is still an open problem. Three main challenges exist. First, while version control systems are well-integrated into development processes, product-line engineering requires investment into additional tooling and different processes that are difficult to adopt. In fact, organizations rarely adopt product line engineering from scratch [6], but rather use readily available version control systems with their branching and forking facilities–a strategy known as clone & own [15, 10]. While this strategy is simple, it does not scale with the number of variants, and then requires evolving (i.e., re-engineering) cloned variants into a product-line platform [2]. Second, evolving product-line platforms is substantially more complex than evolving single variants, mainly since developers need to work on all variants at the same time [25]. Third, the granularity of tracking versions of variants is still unclear. While the whole platform can be versioned, ideally, versioning at the level of features should be supported.

In summary, SCM and SPLE are two widely established, yet actively researched software engineering disciplines offering a variety of concepts to deal with software versions and variants [16, 14, 18, 21]. Yet, despite various attempts [22, 34, 33, 20], none of the two disciplines has been successful in establishing unified solutions addressing both problems at the same time–mainly due to the isolation of both communities and due to the absence of realistic and widely accepted requirements on how to evaluate the effectiveness of techniques for managing both versions and variants.

Goals of the Seminar

This Dagstuhl Seminar aimed at establishing a body of knowledge on unified version and variant management. We invited leading practitioners and researchers from both disciplines to discuss each other's challenges, solutions, and experiences. The seminar's goals were to: (i) survey state-of-the-art SCM and SPLE concepts and map both areas' terminologies and open problems, (ii) gather industrial and academic challenges and requirements on integrated version and variant management, (iii) survey and assess existing evaluation approaches, and (iv) stablish a research agenda, research infrastructure, and working groups. To guide future research, the participants also discussed the basis to work on improved evaluation approaches–as benchmarks for new version and variant management techniques. As such, the long-term goal of the seminar was to enable the development and evaluation of enhanced version and variant management techniques that will be adopted in practice.

Week Overview

Monday. After an introduction of all participants, the seminar started off with general talks on versioning and variability. Bernhard Westfechtel set the stage with an introduction into version management concepts and workflows, which already illustrated some overlap with variability management concepts. For instance, directed deltas are conceptually similar to compositional variation mechanisms (e.g., feature modules or delta modules), and the construction of versions in intensional versioning can be related to the configuration-based derivation of individual variants from a product-line platform. The seminar continued with a talk by Don Batory, who discussed the integration of version control systems, variability management techniques, and integrated development environments (IDEs) based on ideas centering around a better representation and execution of program refactorings in versioned and variant-rich software systems. The talk by Thorsten Berger (actually given on Tuesday, since the introduction round and discussions for the other talks took more time) followed up on the concepts introduced in the previous talks and presented a survey on variation control systems, which support developers managing variant-rich systems in terms of features. Such variation control systems go back to the end of the 1970s with concepts and prototypes developed in the SCM community, but never made it into the mainstream. The talk surveyed their concepts and discussed problems likely prohibiting their adoption. Thereafter, we enjoyed three talks on industrial perspectives given by our industrial practitioners: Henrik Lönn (Volvo), Danilo Beuche (pure::systems), and Ramesh S. (General Motors; talk also given on Tuesday for timing reasons), confirming and explaining the gaps between academia and industry.

Tuesday. The day started with an introduction into the prospective breakout groups for the afternoon, followed by the talk of Christoph Seidl on versioning of product lines relying on a representation of feature versions in a new dialect of feature models, called Hyper Feature Models. Thereafter, the breakout sessions on four relevant topics took place, specifically: on a conceptual model to map SPLE and SCM concepts, on operations for managing versions and variants, on analyses of versions and variants, on workflows for managing versions and variants, and on first-class support of variability and versioning in programming languages. A benchmarking group was discussed, but abandoned in favor of first working on the foundations before discussing benchmarking techniques to evaluate prospective unified techniques for versioning and variability. The breakout group discussions continued until the afternoon,

before the remaining talks from Monday were given (Thorsten Berger and Ramesh Sethu), followed by lightning talks from Shurui Zhou and Sandro Schulze. Shurui discussed the relevance of version and variability management in the domain of engineering AI-based systems, where models and large dataset need to be managed. Sandro proposed a round-trip-engineering process relying on unified management of versioning and variability, relying on automated extraction of variability information from cloned variants (which should be integrated into a platform in a round-trip-engineering manner).

Wednesday. We started the day with a talk by Daniel Strüber on benchmarking scenarios and a survey of existing benchmarks. In fact, it is a common consensus of the community that the lack of strong, landmark benchmarks hinders the progress in both communities (SCM and SPLE). Thereafter, Yi Li presented his work on slicing of the history of software codebases along features, where features are represented by test cases to help identifying the relevant code in a longitudinal manner. Thomas Thüm then presented a vision on the–ideally automated–synchronization of cloned variants as followed by the VariantSync research project which is led by Thomas and Timo Kehrer. Thomas also presented a very first prototypical implementation of the VariantSync tool. The approach shares, based on audience feedback, ideas with the Virtual Platform, proposed by researchers in 2014 [1]. In the afternoon, the majority of the participants continued their discussion on their group trip to the city of Trier and a dinner at a local winery.

Thursday. The day began with a talk by Gabriele Taentzer, presenting a generalizing framework for transformations of software product lines, relying on the formalism of category theory. Another talk was given by Julia Rubin on equivalence checking of variants based on behavior instead of structural characteristics of changes. Thereafter, the breakout groups continued their discussions until the later afternoon, where the results were presented to the other seminar participants. After dinner, two lightning talks were given by Paulo Borba and Iris Reinhartz-Berger. Paulo discussed the detection of semantic merge conflicts in the light of avoiding unwanted feature interactions, and Iris presented insights from two research projects on behavior-derived variability analysis and mechanisms recommendation.

Friday. The last day of the seminar started with a talk by Lukas Linsbauer on his work towards a feature-oriented and distributed version-control system, relying on the variant-integration tooling ECCO. We then had a closing discussion, re-iterating the main challenges we identified throughout the seminar, as well as discussing future work.

Outcome of the Seminar

The seminar established breakout groups who continued their discussion after the seminar and already published two papers [7, 1] at the VariVolution workshop, hosted at the Systems and Software Product Line Conference (SPLC). In addition, a paper accepted at the main track of SPLC on benchmarking, relying on input from the seminar participants via a survey [31], and providing an initial infrastructure for community-oriented benchmark creation,[1] can be seen as a core outcome of the seminar.

A core topic of the final discussion was the teaching of SPLE and SCM concepts–an important means to eventually improve the handling of versions and variants in practice. One

[1] https://bitbucket.org/easelab/evobench

of the problems identified is that, while SCM is covered sufficiently, the relevant variability-management concepts are not taught at the Bachelor's level in the majority of universities. However, the discussants believe that practicing feature-oriented analysis and design early in the curriculum would be beneficial, where currently object-oriented analysis and design is dominating. Interestingly, based on the experience of the discussants, SPLE is still seen as something rather futuristic by students, which is somewhat surprising, given that building highly configurable systems and software platforms are established practices, so perhaps there is a perception and awareness problem that teaching needs to address. Naturally, a course teaching SPLE at the Bachelor's level should also teach the relevant SCM concepts. A closely related topic discussed is that of teaching architectures, especially those of product lines, which is not really in the focus of current software architecture courses. Of course, it is generally difficult to talk to students about software architecture, since, as a discussant explains, a relevant abstract concept that students do not immediately perceive as relevant in the course of the studies. In contrast, with compilers and databases, students obtain some hands-on experience, which allows them to relate more closely to, especially with respect to a future job in the industry. This calls for close collaboration with industry in SPLE courses.

Establishing benchmarks turned out to be a more difficult problem than expected. Benchmarking was prominently discussed, as well as input elicited for a set of 11 high-level benchmarking scenarios defined by some of the seminar participants and organizers before the seminar. The participants plan to follow-up on creating concrete benchmarks upon the infrastructure created.[1]One idea is to build a web application to contribute specific benchmark data (e.g., code integration examples, comprising the original code variants and the final result as a ground truth) to establish a community benchmark. Another interesting comment was that the currently published case studies and experience reports about variability management and product lines are relatively old and do not provide sufficient technical details. Furthermore, they also do not highlight the problems associated with clone & own and the need for product-line migration techniques adequately. This discussion is a call to arms for improving the benchmarking situation in the SCM and SPLE community.

Last but not least, an important outcome of the final discussion session of the seminar is the need for a commonly agreed set of core concepts, mechanisms and practices–a well-documented Body of Knowledge (BOK) of our discipline. Currently, only some aspects of versioning in time and space are partially covered by the Software Engineering BOK (SWEBOK). However, for promoting a consistent view of our discipline worldwide and beyond our discipline borders as well as for having a foundation for a consistent curriculum development, a dedicated BOK or an extension of the SWEBOK may be necessary as a community effort.

References

1 Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines*. Springer, Berlin Heidelberg, 2013.

2 Wesley K. G. Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering*, 22(6):2972–3016, 2017.

3 Jonatas Ferreira Bastos, Paulo Anselmo da Mota Silveira Neto, Padraig O'Leary, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. Software product lines adoption in small organizations. *Journal of Systems and Software*, 131(Supplement C):112–128, 2017.

4 Thorsten Berger, Daniela Lettner, Julia Rubin, Paul Grünbacher, Adeline Silva, Martin Becker, Marsha Chechik, and Krzysztof Czarnecki. What is a Feature? A Qualitative

Study of Features in Industrial Software Product Lines. In *SPLC*, 2015.

5 Thorsten Berger, Divya Nair, Ralf Rublack, Joanne M. Atlee, Krzysztof Czarnecki, and Andrzej Wasowski. Three cases of feature-based variability modeling in industry. In *MODELS*, 2014.

6 Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. A Survey of Variability Modeling in Industrial Practice. In *VaMoS*, 2013.

7 Thorsten Berger, Steven She, Krzysztof Czarnecki, and Andrzej Wąsowski. Feature-to-Code mapping in two large product lines. In *SPLC*, 2010.

8 Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki. Variability modeling in the real: A perspective from the operating systems domain. In *ASE*, 2010.

9 Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wąsowski, and Krzysztof Czarnecki. A Study of Variability Models and Languages in the Systems Software Domain. *IEEE Transactions of Software Engineering*, 39(12):1611–1640, 2013.

10 John Businge, Openja Moses, Sarah Nadi, Engineer Bainomugisha, and Thorsten Berger. Clone-based variability management in the Android ecosystem. In *ICSME*, 2018.

11 Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, 2001.

12 Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Comput. Surv.*, 30(2):232–282, 1998.

13 Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston, MA, 2000.

14 Danny Dig, Kashif Manzoor, Ralph Johnson, and Tien N Nguyen. Refactoring-aware configuration management for object-oriented programs. In *ICSE*, 2007.

15 Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. An exploratory study of cloning in industrial software product lines. In *CSMR*, 2013.

16 Jacky Estublier, David Leblang, André van der Hoek, Reidar Conradi, Geoffrey Clemm, Walter Tichy, and Darcy Wiborg-Weber. Impact of software engineering research on the practice of software configuration management. *ACM Transactions on Software Engineering and Methodology*, 14(4):383–430, 2005.

17 Kyo Kang, Sholom Cohen, James Hess, William Nowak, and Spencer Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report SEI-90-TR-21, CMU, 1990.

18 Timo Kehrer, Udo Kelter, and Gabriele Taentzer. A rule-based approach to the semantic lifting of model differences in the context of model versioning. In *ASE*, 2011.

19 Jacob Krueger, Wanzi Gu, Hui Shen, Mukelabai Mukelabai, Regina Hebig, and Thorsten Berger. Towards a better understanding of software features and their characteristics: A case study of Marlin. In *VaMoS*, 2018.

20 Vincent J. Kruskal. Managing multi-version programs with an editor. *IBM Journal of Research and Development*, 28(1):74–81, 1984.

21 Philip Langer, Manuel Wimmer, Petra Brosch, Markus Herrmannsdörfer, Martina Seidl, Konrad Wieland, and Gerti Kappel. A posteriori operation detection in evolving software models. *Journal of Systems and Software*, 86(2):551–566, 2013.

22 Lukas Linsbauer, Thorsten Berger, and Paul Grünbacher. A classification of variation control systems. In *GPCE*, 2017.

23 Stephen A. MacKay. The state of the art in concurrent, distributed configuration management. In *SCM-4 and SCM-5*, 1995.

24 Axel Mahler. Configuration management. Chapter Variants: Keeping Things Together and Telling Them Apart. Wiley, 1995.

25 Jean Melo, Claus Brabrand, and Andrzej Wąsowski. How does the degree of variability affect bug finding? In *ICSE*, ACM.

26 Mukelabai Mukelabai, Damir Nešić, Salome Maro, Thorsten Berger, and Jan-Philipp Steghöfer. Tackling combinatorial explosion: A study of industrial needs and practices for analyzing highly configurable systems. In *ASE*, 2018.

27 David Parnas. On the design and development of program families. *IEEE Transactions on Software Engineering*, 2(1):1–9, 1976.

28 Leonardo Passos, Jesus Padilla, Thorsten Berger, Sven Apel, Krzysztof Czarnecki, and Marco Tulio Valente. Feature scattering in the large: A longitudinal study of Linux kernel device drivers. In *MODULARITY*, 2015.

29 Christopher Pietsch, Timo Kehrer, Udo Kelter, Dennis Reuling, and Manuel Ohrndorf. SiPL–A Delta-Based Modeling Framework for Software Product Line Engineering. In *ASE*, 2015.

30 Julio Sincero, Horst Schirmeier, Wolfgang Schröder-Preikschat, and Olaf Spinczyk. Is the Linux kernel a software product line. In *Workshop on Open Source Software and Product Lines*, 2007.

31 Daniel Strueber, Mukelabai Mukelabai, Jacob Krueger, Stefan Fischer, Lukas Linsbauer, Jabier Martinez, and Thorsten Berger. Facing the truth: Benchmarking the techniques for the evolution of variant-rich systems. In *SPLC*, 2019.

32 Christer Thörn. Current state and potential of variability management practices in software-intensive SMEs: Results from a regional industrial survey. *Information and Software Technology*, 52(4):411–421, 2010.

33 Eric Walkingshaw and Klaus Ostermann. Projectional editing of variational software. In *GPCE*, 2014.

34 Bernhard Westfechtel, Bjørn P. Munch, and Reidar Conradi. A layered architecture for uniform version management. *IEEE Transactions of Software Engineering*, 27(12):1111–1133, 2001.

2 Table of Contents

Executive Summary
Thorsten Berger, Marsha Chechik, Timo Kehrer, and Manuel Wimmer 2

Overview of Talks

Version Models for Software Configuration Management
Bernhard Westfechtel 10

Roadmap to Revolutionize IDE SPL Technology
Don Batory 10

Closing The Gap
Danilo Beuche 11

Variants and Versions in a Vehicle Integration Context
Henrik Lönn 11

Versioning & Product Lining is Automotive ECS: Challenges and Strategies
Ramesh Sethu 12

Managing Variability in Space and Time in Software Families
Christoph Seidl 12

A Classification of Variation Control Systems
Thorsten Berger 13

Facing the Truth: Benchmarking the Techniques for the Evolution of Variant-Rich Systems
Daniel Strüber 14

Semantic Slicing of Software Version Histories
Yi Li 15

VariantSync – Automating the Synchronisation of Software Variants
Thomas Thüm 15

Transformations of Software Product Lines: A Generalizing Framework Based on Category Theory
Gabriele Taentzer 16

Client-Specific Equivalence Checking
Julia Rubin 16

Lightning Talks

Versioning ML Models & Data in Time and Space
Shurui Zhou 17

Towards Variability Mining Across Artifacts with Round-Trip Engineering
Sandro Schulze 17

Behavior-Derived Variability Analysis and Mechanisms Recommendation
Iris Reinhartz-Berger 18

Checking Feature Interaction and Code Integration Conflicts
Paulo Borba 18

Towards a Feature-Oriented and Distributed Version Control System
Lukas Linsbauer 19

Breakout Groups

Analysis Group 19

Conceptual Modeling Group 21

Workflow Group 25

Languages Group 27

Operations Group 29

Participants 30

3 Overview of Talks

3.1 Version Models for Software Configuration Management

Bernhard Westfechtel (Universität Bayreuth, DE)

Joint work of Reidar Conradi, Bernhard Westfechtel
Main reference Reidar Conradi, Bernhard Westfechtel: "Version Models for Software Configuration Management", ACM Comput. Surv., Vol. 30(2), pp. 232–282, 1998.
URL https://doi.org/10.1145/280277.280280

This talk focuses on the version models underlying both commercial systems and research prototypes of software configuration management systems. It introduced the notion of a version, which is a state of an evolving item. According to different dimensions of evolution, versioning is classified into temporal, logical, and cooperative versioning. In particular, revisions and variants are versions evolving in time and space, respectively. In the case of state-based versioning, a version is specified in terms of the states of versioned items. In the case of change-based versioning, a version is specified in terms of changes being applied to a base version. The version space is a structure which describes the organization of versions, e.g., by version graphs consisting of versions and successor relationships. Finally, we may distinguish between extensional and intensional versioning. In the former case, a version set is defined by explicit enumeration of its members; in the latter case, a version is described by a predicate specifying its desired properties. This distinction is crucial: SCM systems based on extensional versioning focus on the reconstruction of versions which were submitted to the repository; SCM systems based on intensional versioning need to construct versions which may have never been created before, such that specified version properties hold. While traditional SCM systems focus on extensional, temporal, and cooperative versioning, they do provide limited support for variants, which may be represented as branches in a version graphs. However, this approach breaks down in the case of multi-dimensional variations, due to the combinatorial explosion of the number of branches and the multiple maintenance problem of repeating changes on each affected branch. To solve this problem, a number of SCM systems were developed which do support multi-dimensional variation. It turns out that these systems are based on similar concepts, as they were developed in the context of software product line engineering.

3.2 Roadmap to Revolutionize IDE SPL Technology

Don Batory (The University of Texas at Austin, US)

Integrating variability–a.k.a., software product lines (SPLs)–with version control is an exciting and largely unexplored research topic. I encountered this topic in 2007 when visiting a US government facility that created SPLs for a fleet of ships. In following this lead, I encountered yet another technical problem that was similar, but (I thought) more critical to explore–the integration of variability (SPLs) with refactoring–and the need to modernize (Java or OO) IDE support for building SPLs, so that modern OO program development practices can be applied to BOTH SPL's one-of-a-kind programs and SPLs.

In this talk, I explain key ideas my colleagues and I discovered in integrating variability with OO refactoring, and how the challenges are very similar to that of integrating variability with version control. I believe it will be useful to be aware of these similarities as research with version control proceeds, because ultimately what modern SPL tooling requires is an integration of variability with version control AND refactorings.

References

1 Jongwook Kim, Don Batory and Danny Dig, "Refactoring Java Software Product Lines," In *SPLC*, 2017.
2 Don Batory, Invited Presentation: "Program Refactoring, Program Synthesis, and Model-Driven Development," In *ETAPS*, 2007.
3 Danny Dig, PhD Thesis: "Automated Upgrading of Component-Based Applications," 2007.

3.3 Closing The Gap

Danilo Beuche (pure::systems GmbH, DE)

This talk discusses whether there is a gap between the current "main-stream" product line engineering (PLE) research and the challenges in the industrial application of PLE. To that end, it presents typical usage scenarios of the commercial product-line engineering tool pure::variants and shows if and how it deals with some of the challenges encountered. The first challenge is the need to investigate if versioning is a suitable alternative of PLE, and more importantly if it is in fact cheaper than investing in a product-line architecture. The second challenge is the co-evolution of the product line and how to address it. This involves change propagation and merging in multiple variants, and upward propagation of the changes in the variants to the product line base code. The third challenge comes out of the question of whether product-line owners and product-line engineers should be consulted to get their viewpoints. The challenge is to establish if the right people are working in the industry. Finally, a couple of discussion points for the seminar are raised.

3.4 Variants and Versions in a Vehicle Integration Context

Henrik Lönn (Volvo Group Trucks Technology, SE)

Automotive embedded systems are increasingly critical and capable. For this reason, rigorous verification is necessary. Simulation based techniques multiply the test velocity and allow rare and dangerous situations to be exercised. The rich variability of control software and mechanical configurations calls for systematic variability management to secure verification confidence and completeness. This talk will address a model-based approach where software, electronics, and mechanics is represented for the purpose of both simulation and software development towards target execution. Variability is a key aspect, in order to jointly configure the embedded system and its models. Development is performed iteratively and incrementally with continuous integration, and simulations are therefore generated automatically including the variability resolution phase.

3.5 Versioning & Product Lining is Automotive ECS: Challenges and Strategies

Ramesh Sethu (General Motors R&D, US)

There is an ever increasing demand for Automotive Electronics, Control and Software (ECS) Assets in a vehicle: Today's advanced driver assistance systems in automotive systems, like Adaptive Cruise Controllers are giving rise to more complex and safety-critical Level 2 and beyond, features. Potential safety violations and security vulnerabilities are plaguing the industry. To meet the demands of rigorous development of software, more emphasis is being placed on additional life cycle artefacts like requirements, design models, simulation and test results which significantly impacts the resource requirements for building these systems. Reuse and product lining are some of the techniques used in the industry to amortize the costs of developing these systems over larger product portfolio.

The talk highlights and emphasizes the need for versioning and product line mechanisms extended to all life cycle assets, besides software. Software is reasonably structured and the standard techniques used in the software industry for a product lining and versioning may not be easily extended to unstructured and higher dimensional assets like textual requirements and simulation results. There is also a strong need for tracing the relationship between the artefacts and their versions and variants which would help in error analysis and resolution. The amount of information available across the life cycle is enormous and it would be interesting to see whether advances in data analysis can be extended to solve these problems. Efficient management of the large amount of unstructured data is also required. To reduce the overhead of managing large amount of data, opportunities for reducing or containing the revisions and variants are also very important. The last but not the least aspect is the development of appropriate tools for managing the life cycle assets across their revisions and variants.

3.6 Managing Variability in Space and Time in Software Families

Christoph Seidl (Technische Universität Braunschweig, DE)

Main reference Christoph Seidl, Ina Schaefer, Uwe Aßmann: "Integrated management of variability in space and time in software families", in Proc. of the 18th International Software Product Line Conference, SPLC'14, Florence, Italy, September 15-19, 2014, pp. 22–31, ACM, 2014.
URL https://doi.org/10.1145/2648511.2648514

Software product lines (SPLs) and software ecosystems (SECOs) encompass a family of closely related software systems in terms of common and variable assets that are configured to concrete products (variability in space). Over the course of time, variable assets of SPLs and especially SECOs are subject to change in order to meet new requirements as part of software evolution (variability in time). In many cases, both dimensions of variability have to be handled simultaneously, e.g., as not all customers upgrade their respective products immediately or completely. In this presentation, we introduce an integrated approach to manage variability in space and time in software families using Hyper Feature Models (HFMs) with feature versions and combine them with an extension of the transformational variability

realization mechanism delta modeling. This allows derivation of concrete software systems from an SPL or SECO configuring both functionality (features) as well as versions.

References

1 Seidl, Aßmann: Towards Modeling and Analyzing Variability in Evolving Software Ecosystems, In *VaMoS*, 2013.
2 Seidl, Schaefer, Aßmann: Capturing Variability in Space and Time with Hyper Feature Models, In *VaMoS*, 2014.

3.7 A Classification of Variation Control Systems

Thorsten Berger (Chalmers | University of Gothenburg, SE)

Joint work of Lukas Linsbauer, Thorsten Berger, Paul Grünbacher
Main reference Lukas Linsbauer, Thorsten Berger, Paul Grünbacher: "A classification of variation control systems", in Proc. of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, GPCE'17, Vancouver, BC, Canada, October 23-24, 2017, pp. 49–62, ACM, 2017.
URL https://doi.org/10.1145/3136040.3136054

Version control systems are an integral part of today's software and systems development processes. They facilitate the management of revisions (sequential versions) and variants (concurrent versions) of a system under development and enable collaboration between developers. Revisions are commonly maintained either per file or for the whole system. Variants are supported via branching or forking mechanisms that conceptually clone the whole system under development. It is known that such cloning practices come with disadvantages. In fact, while short-lived branches for isolated development of new functionality (a.k.a., feature branches) are well supported, dealing with long-term and fine-grained system variants currently requires employing additional mechanisms, such as preprocessors, build systems or custom configuration tools. Interestingly, the literature describes a number of variation control systems, which provide a richer set of capabilities for handling fine-grained system variants compared to the version control systems widely used today. In this paper we present a classification and comparison of selected variation control systems to get an understanding of their capabilities and the advantages they can offer. We discuss problems of variation control systems, which may explain their comparably low popularity. We also propose research activities we regard as important to change this situation.

References

1 Stefan Stanciulescu, Thorsten Berger, Eric Walkingshaw, Andrzej Wasowski. Concepts, Operations, and Feasibility of a Projection-Based Variation Control System. In ICSME, 2016.
2 Stefan Fischer, Lukas Linsbauer, Roberto E. Lopez-Herrejon, Alexander Egyed. The ECCO tool: Extraction and composition for clone-and-own. In ICSE, 2015.
3 Felix Schwägerl, Bernhard Westfechtel. SuperMod: tool support for collaborative filtered model-driven software product line engineering. In ASE, 2016.
4 Bjørn Gulla, Even-André Karlsson, and Dashing Yeh. Change-oriented Version Descriptions in EPOS. Softw. Eng. J., 6(6):378–386, 1991.
5 Vincent J. Kruskal. Managing multi-version programs with an editor. IBM Journal of Research and Development, 28(1):74–81, 1984.

6 Vincent J. Kruskal. A blast from the past: Using p-edit for multidimensional editing. In Workshop on Multi-Dimensional Separation of Concerns in Software Engineering, 2000.
7 Anund Lie, Reidar Conradi, Tor Didriksen, Even-Andre Karlsson. Change oriented versioning in a software engineering database. SIGSOFT Softw. Eng. Notes, 14(7):56–65, 1989.
8 Bjørn P. Munch, Jens-Otto Larsen, Bjørn Gulla, Reidar Conradi. Uniform versioning: The Change-Oriented Model. Norwegian Institute of Technology, Trondheim, Norway, 1993.

3.8 Facing the Truth: Benchmarking the Techniques for the Evolution of Variant-Rich Systems

Daniel Strüber (Chalmers & University of Gothenburg, SE)

Joint work of Daniel Strüber, Mukelabai Mukelabai, Jacob Krüger, Stefan Fischer, Lukas Linsbauer, Jabier Martinez, Thorsten Berger
Main reference Daniel Strüber, Mukelabai Mukelabai, Jacob Krüger, Stefan Fischer, Lukas Linsbauer, Jabier Martinez, Thorsten Berger: "Facing the Truth: Benchmarking the Techniques for the Evolution of Variant-Rich Systems," in 23rd International Systems and Software Product Line Conference – Volume A (SPLC'19), September 9–13, 2019, Paris, France. ACM, New York, NY, USA, 12 pages.
URL https://doi.org/10.1145/3336294.3336302

The evolution of software systems in general, and of variant-rich systems in particular, is a challenging task. Many techniques have been proposed in the literature to support developers during software evolution. To advance such techniques and support their adoption, it is crucial to evaluate them against realistic baselines, ideally in the form of generally accessible benchmarks. To this end, we need to improve our empirical understanding of typical evolution scenarios for variant-rich systems and their relevance for benchmarking, and identify gaps in the existing benchmarking landscape. In this work, we establish eleven evolution scenarios in which benchmarks would be beneficial. Our scenarios cover typical lifecycles of variant-rich system, ranging from clone & own to adopting and evolving a configurable product-line platform. For each scenario, we formulate requirements for benchmarking the corresponding techniques. To assess the clarity and relevance of our scenarios, we conducted a community survey with software variability and evolution experts. We also surveyed the existing benchmarking landscape, identifying synergies and gaps with respect to our scenarios. We observed that most scenarios, despite being perceived as important by researchers, are only partially or not at all supported by existing benchmarks–a call to arms for building community benchmarks upon our requirements. We hope that our work raises awareness for benchmarking as a means to advance techniques for evolving variant-rich systems, and that it will lead to a benchmarking initiative in our community.

3.9 Semantic Slicing of Software Version Histories

Yi Li (Nanyang Technological University, SG)

Joint work of Yi Li, Chenguang Zhu, Julia Rubin, Marsha Chechik
Main reference Yi Li, Chenguang Zhu, Julia Rubin, Marsha Chechik: "Semantic Slicing of Software Version Histories", IEEE Trans. Software Eng., Vol. 44(2), pp. 182–201, 2018.
URL http://dx.doi.org/10.1109/TSE.2017.2664824

Software developers often need to transfer functionality, e.g., a set of commits implementing a new feature or a bug fix, from one branch of a configuration management system to another. That can be a challenging task as the existing configuration management tools lack support for matching high-level, semantic functionality with low-level version histories. The developer thus has to either manually identify the exact set of semantically-related commits implementing the functionality of interest or sequentially port a segment of the change history, "inheriting" additional, unwanted functionality. In this talk, we tackle this problem by providing automated support for identifying the set of semantically-related commits implementing a particular functionality, which is defined by a set of tests. We present two approaches, CSER and DEFINER, in a specific implementation for Java projects managed in Git and evaluate its correctness and effectiveness on a set of open-source software repositories. We show that it allows to identify subsets of change histories that maintain the functionality of interest but are substantially smaller than the original ones.

3.10 VariantSync – Automating the Synchronisation of Software Variants

Thomas Thüm (Technische Universität Braunschweig, DE)

Joint work of Thomas Thüm, Timo Kehrer

Today's software is often released in multiple variants to meet all customer requirements. Software product lines have the potential to decrease development costs and time-to-market, and have been actively researched for more than two decades. Nevertheless, practitioners frequently rely on ad hoc reuse based on a principle which is known as clone & own, where new variants of a software family are created by copying and adapting an existing variant. However, if a critical number of variants is reached, their maintenance and evolution becomes impractical, if not impossible, and the migration to a product line is often infeasible. With the research conducted in VariantSync, we aim to enable a fundamentally new development approach which bridges the gap between clone & own and product lines, combining the minimal overhead of clone & own with the systematic handling of variability of software product lines in a highly flexible methodology. The key idea is to transparently integrate the central product-line concept of a feature with variant management facilities known from version control systems in order to automatically synchronize a set of evolving variants. Lifting the underlying techniques employed by version control systems to the abstraction level of features which are shared among variants is an open problem and the main research challenge addressed in VariantSync. We believe that our research results have the potential to effectively change the way how practitioners will develop multi-variant software systems for which it is hard to foresee which variants will be added or released in the future.

3.11 Transformations of Software Product Lines: A Generalizing Framework Based on Category Theory

Gabriele Taentzer (Philipps-University Marburg, DE)

Joint work of Gabriele Taentzer, Rick Salay, Daniel Strüber, Marsha Chechik
Main reference Gabriele Taentzer, Rick Salay, Daniel Strüber, Marsha Chechik: "Transformations of Software Product Lines: A Generalizing Framework Based on Category Theory", in Proc. of the 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2017, Austin, TX, USA, September 17-22, 2017, pp. 101–111, IEEE Computer Society, 2017.
URL https://doi.org/10.1109/MODELS.2017.22

Software product lines are used to manage the development of highly complex software with many variants. In the literature, various forms of rule-based product line modifications have been considered. However, when considered in isolation, their expressiveness for specifying combined modifications of feature models and domain models is limited. In this talk a formal framework for product line transformations is presented that is able to combine several kinds of product line modifications presented in the literature. Moreover, it defines new forms of product line modifications supporting various forms of product lines and transformation rules. Our formalization of product line transformations is based on category theory, and concentrates on properties of product line relations instead of their single elements. This framework provides improved expressiveness and flexibility of software product line transformations while abstracting from the considered type of model.

3.12 Client-Specific Equivalence Checking

Julia Rubin (University of British Columbia, CA)

Joint work of Federico Mora, Yi Li, Julia Rubin, Marsha Chechik
Main reference Federico Mora, Yi Li, Julia Rubin, Marsha Chechik: "Client-specific equivalence checking", in Proc. of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018, pp. 441–451, ACM, 2018.
URL https://doi.org/10.1145/3238147.3238178

Software is often built by integrating components created by different teams or even different organizations. With little understanding of changes in dependent components, it is challenging to maintain correctness and robustness of the entire system. In this talk, we discuss the effect of component changes on the behavior of their clients. We show that changes in a component are often irrelevant to a particular client and thus can be adopted without any delays or negative effects. Following this observation, we formulate the notion of client-specific equivalence checking and develop an automated technique optimized for checking such equivalence. We evaluate our technique on a set of benchmarks, including those from the existing literature on equivalence checking, and show its applicability and effectiveness.

4 Lightning Talks

4.1 Versioning ML Models & Data in Time and Space

Shurui Zhou (Carnegie Mellon University, US)

Machine Learning (ML) technology has been widely used in a great number of applications. It is common that data scientists test different ML models, hyperparameters, configuration options, and so on, aiming to find the best configurations for the ML task. However, in order to easily track the process and compare results among different experimental settings, data scientists need to manually log different versions of the ML models, configurations, and so on, which is inefficient and error-prone. Besides, current version control techniques in the software-engineering domain, such as git, could not fulfill the requirement for ML-related tasks because of the scale and formats of data beyond only source code. Therefore, we aim to understand the problems in the ML domain and how versioning and variation mechanisms can support ML-related tasks and help data scientists to work and collaborate more efficiently in the future.

4.2 Towards Variability Mining Across Artifacts with Round-Trip Engineering

Sandro Schulze (Otto-von-Guericke University Magdeburg, DE)

This talk discusses some ideas about unifying the information of variability mining from different kind of artifacts. The typical semantics of clone & own are discussed along with the resulting challenges during development and maintenance. These challenges include lack of support for bug fix propagation to different variants, decentralization of information, risk of loss of information for new features, variation points not being expit and redundancy of efforts. A potential solution is to mine variability related information from artifacts of interest and extracting features and variability from those artifacts.These artifacts can be related to code, design or requirements etc. In addition, requirements can be used to create links between the other artifacts. The benefit of this is that knowledge on variability can be propagated. However, the challenge is that requirements are specified in natural language, which might be prone to inaccuracies and in-consistencies. This talk addresses this problem by proposing a similarity based natural language processing technique for variability mining in various software artifacts.

References

1 Anh Nguyen Duc, Audris Mockus, Randy L. Hackbarth, and John Douglas Palframan, “Forking and Coordination in Multi-platform Development: A Case Study”, In *ESEM*, 2014.
2 Thorsten Berger, Divya Nair, Ralf Rublack, Joanne M. Atlee, Krzysztof Czarnecki, and Andrzej Wasowski, “Three Cases of Feature-Based Variability Modeling in Industry”, In *MODELS*, 2014.

3 Stefan Stanciulescu, Sandro Schulze, and Andrzej Wasowski, "Forked and Integrated Variants In An Open-Source Firmware Project", In *ICSME*, 2015.

4.3 Behavior-Derived Variability Analysis and Mechanisms Recommendation

Iris Reinhartz-Berger (University of Haifa, IL)

Software reuse, the use of existing artifacts in order to produce new software, has many benefits, including increased productivity, reduced costs and time-to-market, and improved quality. In cases of similarly behaving systems, reuse is more challenging, but yet profitable. In this presentation, two projects were introduced. SOVA–Semantic and Ontological Variability Analysis–proposes to combine semantic and ontological considerations that reflect system behavior, rather than implementation, as manifested in different development artifacts (e.g., requirements, test cases, and code). The approach includes behavior extraction, similarity calculation, and variability analysis, producing feature diagrams which depict the perceived variability in the system behaviors. More about SOVA can be found online [1]. VarMeR–Variability Mechanisms Recommendation–supports the analysis, visualization, and recommendation of behavior-derived software reuse based on existing product artifacts. This is done by considering polymorphism-inspired variability mechanisms: parametric, subtyping, and overloading. The outcomes are represented as a multi-layer similarity graph, where the nodes are reusable subjects (products, packages, classes, behaviors), and the edges recommend on potential reuse options in the form of the considered variability mechanisms. More about VarMeR can be found online [2].

References

1 SOVA, https://sites.google.com/is.haifa.ac.il/sova
2 VarMeR, https://sites.google.com/is.haifa.ac.il/varmer

4.4 Checking Feature Interaction and Code Integration Conflicts

Paulo Borba (Federal University of Pernambuco, BR)

Joint work of Paulo Borba, Roberto Barros, Léuson Silva, Rodrigo Bonifácio, Guilherme Cavalcanti, Rafael Mota, João Pedro Moisakis

In this talk we explore the similarities between semantic code integration conflicts and unintended, application-specific feature interaction. By relating the notions of feature and code contribution (changes submitted by a developer), and of interaction and interference, we propose to leverage semantic conflict detection techniques for checking feature interaction. As an initial step, we study the potential of using information flow analysis for detecting semantic conflicts when integrating developers contributions. The overall idea is to detect information flow between developers' contributions. We discuss initial promising results and a number of challenges involved.

4.5 Towards a Feature-Oriented and Distributed Version Control System

Lukas Linsbauer (Johannes Kepler University Linz, AT)

This work relates to several areas of research that use different concepts to achieve similar goals, such as Version Control Sytems (VCS) and Software Configuration Management, Clone & own and Product Lines, and Traceability. VCS keep track of revisions (sequential versions, such as bug fixes) and facilitate collaboration among developers. It is also the responsibility of VCS to manage variants (concurrent versions, such as customer-specific variants). However, current VCS provide insufficiently coarse mechanisms (such as branches, which are essentially clones). Fine-grained variant management (based on individual features that trace to specific implementation artifacts) requires additional (often artifact type specific) mechanisms (such as simple preprocessors for text files or more complex product line platforms). We argue that feature-based variant management is needed to not only retrieve versions that have been explicitly stored (extensional versioning), but also to configure a system based on features and retrieve combinations of features that have not explicitly been stored (intensional versioning). We believe that such feature-based development is even better suited for distributed development than branch-based development (as is evidenced by the extensive use of feature-branches in current practice). Moreover, we observe that systems usually consist of different types of implementation artifacts in addition to source code which is why a generic variability mechanism is needed that is not limited to a specific type of artifacts. This work aims to unify related concepts of the above mentioned areas of research to combine their advantages while avoiding their drawbacks. The goal is to enable a novel feature-oriented and distributed development workflow. We therefore design, implement and evaluate a Feature-Oriented and Distributed Version Control System that supports both extensional and intensional versioning via revision and variant management mechanisms based on features for heterogeneous types of implementation artifacts.

References

1 ECCO, https://jku-isse.github.io/ecco.

5 Breakout Groups

5.1 Analysis Group

Members

Goetz Botterweck, Timo Kehrer, Mukelabai Mukelabai, Ina Schaefer, Klaus Schmid, Leopoldo Teixeira, Thomas Thüm, Mahsa Varshosaz, and Eric Walkingshaw.

5.1.1 Motivation and Goals

There are numerous analyses to cope with variation in space (i.e., product-line analyses) and others that cope with variation in time (i.e., regression analyses). While both kinds of techniques have developed largely independent of each other, the common idea is to exploit the similarities between variants (product-line analysis) and versions (regression analysis) in order to save analysis efforts and to avoid doing entire analysis from scratch. To that

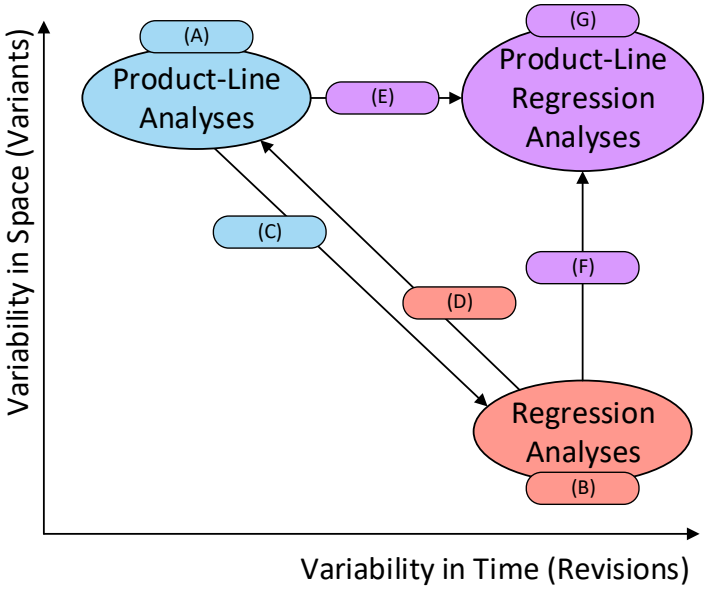

Figure 1 Efficient analyses for two dimensions of variability.

end, in this breakout group, we discussed to which extent product-line analyses can be applied to revisions and, conversely, where regression analyses can be applied to variants. In addition, we discussed challenges related to the combination of product-line and regression analyses. The overall goal is to increase the efficiency of analyses by exploiting the inherent commonality between variants and revisions.

5.1.2 Summary of Results and Open Challenges

An overview of the course of our discussions during the seminar and the corresponding results is shown in Figure 1. It structures analysis techniques along the two dimensions of variability, i.e., variability in time (leading to revisions) and variability in space (leading to variants).

Our starting point was to recall the basic nature of classical product-line analysis (A), devoted to the analysis of variants [9, 8, 6], and regression analysis (B), designed to efficiently analyze revisions [10, 4, 2, 1, 3]. Next, we discussed the application of traditional techniques in both directions, that is, (C) the application of product-line analyses to revisions and (D) the application of regression analyses to variants. In fact, there is a lot of potential for re-using product-line analyses for revisions and for re-using regression analyses for variants, some of which has been already exploited [5, 4]. Finally, we discussed how product-line analyses and regression analyses can be applied to both dimensions of variability. As shown in Figure 1, such product-line regression analyses can be realized by (E) applying product-line analyses to revisions of variants, (F) applying regression analyses to revisions of variants, or (G) combinations of product-line analyses and regression analyses.

As a result of our survey of existing analysis techniques, we identified a couple of challenges and promising directions for future research. In particular, the application of product-line analyses to variation in time seems to be a new application area for many existing product-line analyses. That is, research results of the product-line community could be reused by communities working on regression analyses. While regression analyses have often been applied to variation in space, we also summarized common challenges for their application to

variants. With product-line regression analysis, we denote analyses that cope with variation in both dimensions, namely time and space. For that purpose, two-dimensional lifting of traditional analyses is necessary with respect to both dimensions of variation. We believe that it requires a community effort to identify which strategies for lifting lead to the most efficient analyses, perhaps also paving the way for an integration of multiple strategies.

5.1.3 Further Reading

A more thorough discussion of our literature survey as well as the identified challenges towards efficient analysis of variation in time and space can be found in Thüm et al. [7], a paper that has been written by the breakout group members after the Dagstuhl seminar and that has been recently accepted at the VariVolution workshop at SPLC 2019.

References

1 Steven Arzt and Eric Bodden. Reviser: Efficiently Updating IDE-/IFDS-based Data-flow Analyses in Response to Incremental Program Changes. In *ICSE*, 2014.
2 Larissa Braz, Rohit Gheyi, Melina Mongiovi, Márcio Ribeiro, Flávio Medeiros, Leopoldo Teixeira, and Sabrina Souto. A Change-Aware Per-File Analysis to Compile Configurable Systems with #ifdefs. *Computer Languages, Systems & Structures*, 54:427–450, 2018.
3 Benny Godlin and Ofer Strichman. Regression Verification. In *DAC*, 2009.
4 Wolfgang Heider, Rick Rabiser, Paul Grünbacher, and Daniela Lettner. Using Regression Testing to Analyze the Impact of Changes to Variability Models on Products. In *SPLC*, 2012.
5 Sascha Lity, Malte Lochau, Ina Schaefer, and Ursula Goltz. Delta-oriented model-based SPL regression testing. In *PLEASE*, 2012.
6 Mukelabai Mukelabai, Damir Nešić, Salome Maro, Thorsten Berger, and Jan-Philipp Steghöfer. Tackling combinatorial explosion: A study of industrial needs and practices for analyzing highly configurable systems. In *ASE*, 2018.
7 Thomas Thüm, Leopoldo Teixeira, Klaus Schmid, Eric Walkingshaw, Mukelabai Mukelabai, Mahsa Varshosaz, Goetz Botterweck, Ina Schaefer, and Timo Kehrer. Towards efficient analysis of variation in time and space. In *VaMoS*, 2019.
8 Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. A Classification and Survey of Analysis Strategies for Software Product Lines. ACM Comput. Surv. 47(1):6:1–6:45, 2014.
9 Alexander von Rhein, Sven Apel, Christian Kästner, Thomas Thüm, and Ina Schaefer. The PLA Model: On the Combination of Product-Line Analyses. In *VaMoS*, 2013.
10 Shin Yoo and Mark Harman. Regression Testing Minimization, Selection and Prioritization: A Survey. Softw. Test., Verif. Reliab. 22(2):67–120, 2012.

5.2 Conceptual Modeling Group

Members

Sofia Ananieva, Thorsten Berger, Andreas Burger, Timo Kehrer, Heiko Klare, Anne Koziolek, Henrik Lönn, Ramesh Sethu, Gabriele Taentzer, and Bernhard Westfechtel.

5.2.1 Motivation and Goals

SCM and SPLE are two widely established yet actively researched software engineering disciplines offering a variety of concepts to deal with software variability in time and space.

Research on SCM has proposed versioning models which define the artifacts to be versioned as well as the way in which these artifacts are organized, identified and composed to configurations [4]. Nowadays version control systems such as Subversion [9] or Git [7] are file-based, organizing versions of files in a directed acyclic version graph. Variants of a software artifact or an entire software system are represented by parallel development branches, where each of these branches has its own chronological evolution. Instead of managing variants (a.k.a. products) as clones in parallel branches, SPLE advocates to create a product-line platform that integrates all the product features and contains explicit variation points realized using variability mechanisms such as conditional compilation or element exclusion [5, 3]. However, neither research community has been successful in producing unified management techniques that are effective in practice.

As a step towards overcoming this unfortunate situation, the goal of this breakout group was to conceive a conceptual yet integrated model of SCM and SPLE concepts. This provides discussion grounds for a wider exploration of a unified methodology supporting software evolution in both time and space. The value and possible usage scenarios of a conceptual model are twofold. It may be instantiated to characterize and classify existing approaches, to structure the state-of-the-art and to map and align both communities' core concepts. It may also pinpoint open issues and serve as a vehicle for evaluating different integration strategies on a high-level of abstraction.

5.2.2 Summary of Results

In Figure 2, we present a basic conceptual model of variability in time and space, which was developed during the seminar. It is composed of three differently colored parts corresponding to (i) concepts for variability in time (blue), (ii) concepts for variability in space (green), and (iii) concepts common to both (red). Clearly, although the SCM and SPLE communities have developed largely independently of each other, they share a set of common concepts, notably the idea of composing a system from fragments which serve as units of versioning and as re-usable assets, respectively. These common concepts of system descriptions served as a starting point of our discussion on a conceptual model, before we explored those concepts which we consider to be specific to one of the disciplines. As for variability in time, the key concept of a revision is applied to each fragment, and a sequence of revisions related through predecessor and successor relationships represents the chronological evolution of a fragment. As for variability in space, the key idea is to define a set of explicit variation points which may be bound to concrete fragments in order to instantiate a product.

Finally, we elaborated on an idea of how those concepts could be combined for managing variability in time and space. Figure 3 represents an extension to the introduced model and depicts the proposed integration of variability in space and time. In essence, integration is achieved through the concept of a versioned item, which represents a higher-level versioning of the introduced concepts by putting them under revision control. In this sense, the versioned item acts as a super class for the fragment, for the variation point and for the product line itself.

5.2.3 Discussion and Future Work

To validate that our model is *general* and *appropriate* in the sense that we are able to map its elements to actual approaches for describing such variability, we will apply the model to existing approaches, such as Ecco [6], SuperMod [10], DeltaEcore [11] or SiPL [8] in future work. Several design decisions in the conceptual model were subject to intensive discussion

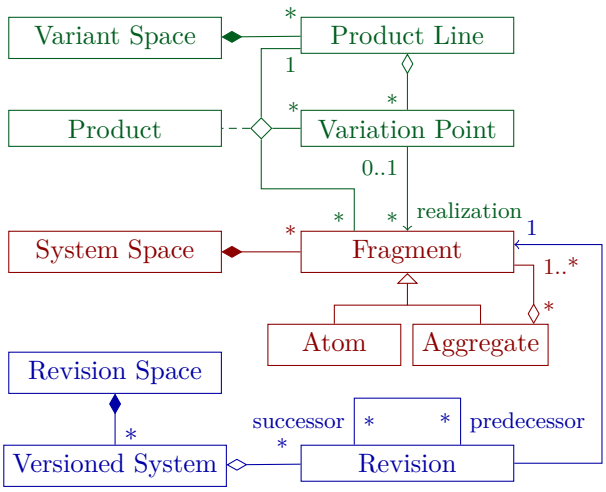

Figure 2 A basic conceptual model of variability in time (blue), variability in space (green), and shared concepts (red).

and may be validated when instantiating the model in future work.

One central subject of discussion is whether branches in revision control systems are a concept of variability in time to support temporary divergence for concurrent development, or whether they represent a realization of variability in space, as they support the existence of products at the same point in time. For the time being, we chose to follow the former option.

Another subject of discussion which requires future validation is whether or not to consider the product a subclass of the *versioned item*. According to Antkiewicz et al. [2], product derivation is either fully automated or followed by manual post-processing (corresponding to the so-called *governance levels* L5 and L6). In the case of fully automated product derivation (L6), a product represents a fully derived artifact for which revision control becomes superfluous since the product line is already put under revision control in the extended model. When manual post-processing takes place (L5), a product does not represent a fully derived artifact anymore for which revision control becomes reasonable again.

Additionally, the semantics of several concepts is only defined through the mechanisms that operate on them. For example, for the configuration of a product from a product line, variation points and fragments are expressed in our model, but constraints that define which variation points and fragments may be selected have to be ensured by a configuration mechanism. The same applies to the unifying concept of our extended model. To define what the relations between revisions of product lines, variation points and fragments are, a mechanism that defines how they can be combined has to be defined. Designing such a mechanism, based on the presented model, should be the next step towards a unifying concept for variability in space and time.

5.2.4 Further Reading

A more detailed description of our conceptual model can be found in Ananieva et al. [1], a paper which has been written by the breakout group members after the Dagstuhl seminar and which has been recently accepted at the VariVolution workshop at SPLC 2019. The paper also provides an example instantiation of our conceptual model and a more thorough discussion of related work.

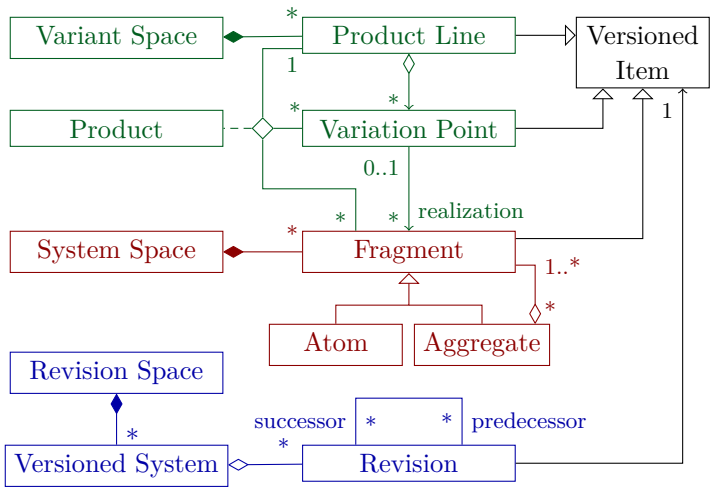

Figure 3 An extended conceptual model (w.r.t. Figure 2) for combining concepts of variability in space and time.

References

1 Sofia Ananieva, Timo Kehrer, Heiko Klare, Anne Koziolek, Henrik Lönn, Ramesh Sethu, Andreas Burger, Gabriele Taentzer, and Bernhard Westfechtel. Towards a conceptual model for unifying variability in space and time. In *VaMOS*, 2019.

2 Michał Antkiewicz, Wenbin Ji, Thorsten Berger, Krzysztof Czarnecki, Thomas Schmorleiz, Ralf Lämmel, Stefan Stănciulescu, Andrzej Wąsowski, and Ina Schaefer. Flexible product line engineering with a virtual platform. In *ICSE*, 2014.

3 Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

4 Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Comput. Surv.*, 30(2):232–282, June 1998.

5 Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

6 Stefan Fischer, Lukas Linsbauer, Roberto E. Lopez-Herrejon, and Alexander Egyed. The ECCO tool: Extraction and composition for clone-and-own. In *ICSE*, 2015.

7 Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media, Inc., 2012.

8 Christopher Pietsch, Timo Kehrer, Udo Kelter, Dennis Reuling, and Manuel Ohrndorf. SIPL–a delta-based modeling framework for software product line engineering. In *ASE*, 2015.

9 C. Michael Pilato, Ben Collins-Sussman, and Brian W. Fitzpatrick. *Version Control with Subversion: Next Generation Open Source Version Control*. O'Reilly Media, Inc., 2008.

10 Felix Schwägerl and Bernhard Westfechtel. Supermod: Tool support for collaborative filtered model-driven software product line engineering. In *ASE*, 2016.

11 Christoph Seidl, Ina Schaefer, and Uwe Aßmann. Integrated management of variability in space and time in software families. In *SPLC*, 2014.

5.3 Workflow Group

Members

Don Batory, Danilo Beuche, Paulo Borba, Paul Grünbacher, Jacob Krüger, Ralf Lämmel, Lukas Linsbauer, Sarah Nadi, Iris Reinhartz-Berger, Sandro Schulze, Stefan Stanciulescu, Daniel Strüber, Shurui Zhou, and Andrzej Wasowski.

5.3.1 Scope

The task of the group was to investigate the role of workflows in the context of the general seminar theme of unifying version and variability management. An initial discussion revealed that there are many different workflows, and defining a general workflow that would work in different domains would not be useful. For instance, several participants discussed the differences between open source software (OSS) and software development processes in industry. Also, participants pointed out the need for workflows supporting both developers/teams working on single platforms as well as workflows guiding the interactions and exchange between multiple development platforms, for instance, in the context of software ecosystems. The discussion also showed that there are many good reasons to create forks ("clone & own engineering"), which also requires specific workflows. For instance, in such a context, awareness becomes particularly important to understand what's happening in the different clones.

However, despite these diverse characteristics, the discussion showed that the different workflows share common elements ("operations"). The group continued with identifying candidate operations as elements of workflow. Examples are: Create Clone, Add Feature to Clone, Create Pull Request, Accept Pull Request, Modify Feature Implementation, or Checkout Configuration to name but a few.

A better understanding of these operations as building blocks for workflows is essential, so the group proceeded with a scenario-based approach.

5.3.2 Scenario-Based Investigation of Two Workflows

The group investigated two real-world workflows more closely by defining them in terms of scenarios to better understand their characteristics. The first scenario covers an OSS workflow about the typical development process of a new feature in Marlin. It was defined based on a paper by workgroup member Jacob Krüger et al [5]. The second scenario describes a typical industrial workflow. It was defined based on the seminar talk and additional information by workgroup member Danilo Beuche. For instance, here are examples of steps from the OSS workflow scenario:

- Alice clones the platform (not a variant) to create a feature
- Alice implements the feature in the clone. She creates a number of commits in her fork. The changes are marked as belonging to the feature.
- Alice creates a pull request, which checks the integration by checking for feature interactions, dead code, feature model inconsistencies, and so on. During the check, a "conflict" appears.
- Alice syncs with upstream (that is, she pulls) and resolves the conflict. This may include resolving inconsistencies in the feature model, make all code alive again, and so on.
- Bob reviews the pull request. He confirms that the commit satisfies the given requirements and that the feature has the right level of granularity, whether it is placed in the right place in the model, whether it has correct dependencies.

- An automated test is performed by the continuous integration (CI) system. Tests are run on the maximum and minimum configuration with the feature enabled and disabled. Cynthia assesses Bob's review and the CI results. She decides to merge the pull request in.

After defining the scenarios the team mapped the candidate operations to the different scenario steps. The purpose of this mapping process was to better understand the scope and purpose of the different operations and to discuss their properties with respect to the seminar topic of integrating version and variability management. In particular, the operations were refined and "feature-ized" by defining signatures. For example: CreateClone (in: existing platform; out: copy of the existing platform), AddFeature (in: featureName, in: featureDependencies), CommitToFeature (in: commits; in: presCond; in: repository).

5.3.3 Challenges and Next Steps

Finally, the group discussed how these workflows and operations could possibly be implemented. The group explored alternatives such as extending an existing version control system like Git; extending a Variation Control System (ECCO, VTS, and so on); or the development of a new system from scratch.

The discussion about future work already covered confirming and evaluating the workflows and operations. There was an agreement to study more industrial workflows, different OSS workflows, and workflows from other domains. In this regard participants raised the issue of the high cognitive complexity of the envisioned workflows. Finally, the team discussed possible benchmarks and case study systems. TurtleBot (TUB) and the Pick-and-Place Unit (TUM, JKU) were named as examples.

For further readings, we refer to the following list of references.

References

1 Michal Antkiewicz, Wenbin Ji, Thorsten Berger, Krzysztof Czarnecki, Thomas Schmorleiz, Ralf Lämmel, Stefan Stanciulescu, Andrzej Wąsowski, and Ina Schaefer. Flexible product line engineering with a virtual platform. In *ICSE*, 2014.
2 Daniel Hinterreiter, Lukas Linsbauer, Florian Reisinger, Herbert Prähofer, Paul Grünbacher, and Alexander Egyed. Feature-oriented evolution of automation software systems in industrial software ecosystems. In *ETFA*, 2018.
3 Wenbin Ji, Thorsten Berger, Michal Antkiewicz, and Krzysztof Czarnecki. Maintaining feature traceability with embedded annotations. In *SPLC*, 2015.
4 Jing Jiang, David Lo, Jiahuan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. Why and how developers fork what from whom in github. *Empirical Software Engineering*, 22(1):547–578, 2017.
5 Jacob Krüger, Mukelabai Mukelabai, Wanzi Gu, Hui Shen, Regina Hebig, and Thorsten Berger. Where is my feature and what is it about? A case study on recovering feature facets. *Journal of Systems and Software*, 152:239–253, 2019.
6 Lukas Linsbauer, Thorsten Berger, and Paul Grünbacher. A classification of variation control systems. In *GPCE*, 2017.
7 Lukas Linsbauer, Alexander Egyed, and Roberto Erick Lopez-Herrejon. A Variability-Aware Configuration Management and Revision Control Platform. In *ICSE*, 2016.
8 Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. Variability extraction and modeling for product variants. *Software and System Modeling*, 16(4):1179–1199, 2017.
9 Rachel Potvin and Josh Levenberg. Why google stores billions of lines of code in a single repository. *Commun. ACM*, 59(7):78–87, 2016.

10 Baishakhi Ray and Miryung Kim. A case study of cross-system porting in forked projects. In *FSE*, 2012.
11 Baishakhi Ray, Christopher Wiley, and Miryung Kim. REPERTOIRE: a cross-system porting analysis tool for forked software projects. In *FSE*, 2012.
12 Luyao Ren, Shurui Zhou, Christian Kästner, and Andrzej Wąsowski. Identifying redundancies in fork-based development. In *SANER*, 2019.
13 Thomas Schmorleiz and Ralf Lämmel. Similarity management of 'cloned and owned' variants. In *SAC*, 2016.
14 Stefan Stanciulescu, Thorsten Berger, Eric Walkingshaw, and Andrzej Wasowski. Concepts, operations, and feasibility of a projection-based variation control system. In *ICSME*, 2016.
15 Stefan Stanciulescu, Sandro Schulze, and Andrzej Wasowski. Forked and integrated variants in an open-source firmware project. In *ICSME*, 2015.
16 Shurui Zhou, Ştefan Stănciulescu, Olaf Leßenich, Yingfei Xiong, Andrzej Wasowski, and Christian Kästner. Identifying features in forks. In *ICSE*, 2018.

5.4 Languages Group

Members

Don Batory, Marsha Chechik, Shahar Maoz, Julia Rubin, Christoph Seidl, and Manuel Wimmer.

5.4.1 Motivation and Goals

This breakout group investigated the idea of having first-class language support for features and variants as well as for versioning. Currently, these aspects are mostly managed outside the base programming and modeling languages, which may come with several drawbacks. For instance, the intention of a feature is likely to be lost, e.g., by generic preprocessor directive, tools that want to perform analyses or changes have to spend a lot of time on deciphering the (company/project specific) conventions how a feature is represented within or for a particular language.

Having first-class language support may allow for immediate feedback of feature effects as well as may simplify tool building if there is a unique and formally defined concept provided by the base language.

5.4.2 Summary of Results and Open Challenges

Language support for Features/Variants. First, the group investigated existing work which goes into the direction of having first-class language support for features and variants. For instance, Matlab/Simulink provides the modeling concept "Variant Subsystems" which is somewhat similar to features where the code generator may disregard parts of the model.

Second, the group investigated different possibilities of feature concepts for programming languages where a specific focus was on Java as example language, but also other options going beyond Java have been considered to be more general. In the following, a summary of different options is provided.

- With preprocessor-like comments such as supported by Antenna, one is very flexible which elements are affected by a feature, but the intention of the feature is lost.
- With if statements, one is limited which elements a feature may affect. Again, the intention of the feature is lost and the variability is pushed to runtime.

- Java annotations would allow to maintain the intention of a feature, but may still be somewhat limited in which elements they can affect (although in newer versions of Java, several improvements have been provided with respect to annotation targets).
- Partial classes as provided for instance in C#, would allow to define 150% models. The solution would be flexible in which elements it can affect, however, the intention of a feature is not explicitly maintained. Furthermore, the solution would work well for optional features, but alternatives would be harder to realize as all partial classes are combined by the compiler.
- Syntax and compiler extensions as provided by Scala or Groovy would allow to build conservative extensions of languages in order to be backward compatible. New keywords may be introduced for defining features and variants which may be used in a flexible manner with respect to which elements they can affect. The intention of a feature is maintained without having to maintain a dialect of the base language. However, it seems politically complicated to get such extensions into the core language specifications in near future.

As a potential future work, investigating the usage of annotations for introducing feature and variant concepts to existing languages seems promising as a first step. A concrete outcome should be the creation and population of a library of annotations that can be enforced, analyzed, and applied by existing IDEs. As a second step, interactions with the community of a language may be necessary to reach consensus about the concepts and their integration in the base language such as assembling Java Specification Request (JSR) in the case of Java.

Finally, we discussed if there should be a "required interface" for features, which specifies the feature realized in a language element. For instance, if a class is realizing two features, one may provide the following explicit syntax:

```
class A realizes FeatureA, FeatureB
```

Language Support for Versioning. The discussion group also investigated first-class language support for versioning of artifacts. Several use cases came up during the discussions: (*i*) trace changes for later manual inspection, (*ii*) suggest changes that need approval, (*iii*) maintain backward compatibility if it is of utmost importance, and (*iv*) negotiate "contracts" between service provider/user.

The group also investigated existing work which already proposes first-class language support for versioning. For instance, AutomationML comes with integrated versioning support [1]. Change-oriented programming by Peter Ebraert [2] advocates tracing changes to language artifacts but has no version support directly integrated with languages.

Finally, the group concluded–based on the discussion and previous experiences of the group members–that the use cases for programming languages are not so clear or pressing. However, the use cases for (system) modeling languages seem more relevant and here clear needs could be identified. Thus, future work in this area is anticipated.

References

1 Stefan Biffl, Emanuel Mätzler, Manuel Wimmer, Arndt Lüder, and Nicole Schmidt. Linking and versioning support for automationml: A model-driven engineering perspective. In *INDIN*, 2015.

2 Peter Ebraert, Jorge Vallejos, Pascal Costanza, Ellen Van Paesschen, and Theo D'Hondt. Change-oriented software engineering. In *ICDL*, 2007.

5.5 Operations Group

Members

Don Batory, Paulo Borba, Yi Li, Wardah Mahmood, Shahar Maoz, Sarah Nadi, Julia Rubin, Klaus Schmid, Christoph Seidl, and Manuel Wimmer.

5.5.1 Motivation and Goals

For building, using, and combining tools in order to deal with versions and variants of systems, a set of well-established and formalized operators is required. Previous work on database integration and evolution focused on the establishment of model management operators [1], which have a well-defined signature as well as clearly defined pre- and postconditions. In addition, a kernel set of artifact types on which they operate is defined.

For instance, let us introduce two different operators which may be of frequent use in versioning and variant scenarios and which build on each other:

```
Differencing Operator: diff(m1, m2) : diffModel
Patching Operator: patch(m2, diff(m0, m1)) : model
```

Based on such a set of operators, more complicated scenarios can be supported by orchestrating the operators to so-called model management scripts. Thus, workflows can be defined based on a set of basic building blocks. Similar support seems beneficial to manage artifacts in versioning and variant management systems or even a combination of it (cf. also the report of the workflow group for similar discussions).

5.5.2 Summary of Results and Open Challenges

When reasoning about operators for versioning and variant management, it becomes evident that such operators may act differently for a file (which is currently the main artifact type in versioning systems) than for a feature. Furthermore, current scenarios such as building a product that combines feature X in v1.1 and feature Y in v2.0 may be hard to achieve with the current version control system operations. Thus, the question came up if there are operators needed for variant management that need fundamental changes to version control systems. The discussion group agreed that this is not required, but instead there should be a layer on top of version control systems for variant management.

Another important aspect for variant management is that one wants to abstract from the file level. Imagine a user wants to look at the history in terms of feature edits (e.g., these three commits are actually changing feature X) instead of tracking changes in individual files. Another scenario is when checking-in, the version control system may ask if this change is related to feature A. Similar, when modifying feature A, one may want to know that no other features are affected.

The group also agreed that it is hard to define operations in isolation, because they highly depend on the variation mechanism and the concrete notion of feature as well as on the specific workflows one may want to achieve. Thus, as future work, the operations should be derived from the common workflows as discussed and reported by the workflow group.

References

1 Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: A programming platform for generic model management. In *SIGMOD*, 2003.

Participants

- Sofia Ananieva
FZI – Berlin, DE
- Sven Apel
Universität des Saarlandes, DE
- Don Batory
University of Texas – Austin, US
- Thorsten Berger
Chalmers and University of Gothenburg, SE
- Danilo Beuche
pure-systems GmbH – Magdeburg, DE
- Paulo Borba
Federal University of Pernambuco – Recife, BR
- Götz Botterweck
University of Limerick, IE
- Andreas Burger
ABB – Ladenburg, DE
- Marsha Chechik
University of Toronto, CA
- Paul Grünbacher
Johannes Kepler Universität Linz, AT
- Timo Kehrer
HU Berlin, DE
- Heiko Klare
KIT – Karlsruher Institut für Technologie, DE
- Anne Koziolek
KIT – Karlsruher Institut für Technologie, DE
- Jacob Krüger
Universität Magdeburg, DE
- Ralf Lämmel
Facebook – London, GB
- Yi Li
Nanyang TU – Singapore, SG
- Lukas Linsbauer
Johannes Kepler Universität Linz, AT
- Henrik Lönn
Volvo – Göteborg, SE
- Wardah Mahmood
Chalmers University of Technology – Göteborg, SE
- Shahar Maoz
Tel Aviv University, IL
- Mukelabai Mukelabai
University of Gothenburg, SE
- Sarah Nadi
University of Alberta – Edmonton, CA
- Iris Reinhartz-Berger
Haifa University, IL
- Julia Rubin
University of British Columbia – Vancouver, CA
- Ina Schaefer
TU Braunschweig, DE
- Klaus Schmid
Universität Hildesheim, DE
- Sandro Schulze
Universität Magdeburg, DE
- Christoph Seidl
TU Braunschweig, DE
- Ramesh Sethu
General Motors – Warren, US
- Stefan Stanciulescu
ABB – Baden-Dättwil, CH
- Daniel Strüber
Chalmers University of Technology – Göteborg, SE
- Gabriele Taentzer
Universität Marburg, DE
- Leopoldo Teixeira
Federal University of Pernambuco – Recife, BR
- Thomas Thüm
TU Braunschweig, DE
- Mahsa Varshosaz
IT University of Copenhagen, DK
- Eric Walkingshaw
Oregon State University – Corvallis, US
- Andrzej Wasowski
IT University of Copenhagen, DK
- Bernhard Westfechtel
Universität Bayreuth, DE
- Manuel Wimmer
Johannes Kepler Universität Linz, AT
- Shurui Zhou
Carnegie Mellon University – Pittsburgh, US