Report from Dagstuhl Seminar 11021

Feature-Oriented Software Development (FOSD)

Edited by

Sven Apel[1], William Cook[2], Krzysztof Czarnecki[3], and Oscar Nierstrasz[4]

1 **Universität Passau, DE,** apel@uni-passau.de
2 **University of Texas - Austin, US,** wcook@cs.utexas.edu
3 **University of Waterloo, CA,** czarnecki@acm.org
4 **Universität Bern, CH,** oscar@iam.unibe.ch

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 11021 "Feature-Oriented Software Development (FOSD)". FOSD is an emerging paradigm that aims at increasing the level of automation, reuse, and variation in software development. The main goal of the Dagstuhl Seminar on FOSD was to gather researchers and practitioners who are active in different communities to discuss the roots, state of the art, and future directions of FOSD research and practice. Additional goals were to strengthen the identity of the feature orientation community and to relate FOSD to other software development paradigms. The report contains an executive summary, abstracts of the talks held during the seminar, and summaries of special sessions.

Seminar 09.–14. January, 2011 – www.dagstuhl.de/11021
1998 ACM Subject Classification D.2.13 Reusable Software
Keywords and phrases FOSD, automation, software family
Digital Object Identifier 10.4230/DagRep.1.1.27
Edited in cooperation with Kacper Bąk

1 Executive Summary

Sven Apel
William R. Cook
Krzysztof Czarnecki
Oscar M. Nierstrasz

License Creative Commons BY-NC-ND 3.0 Unported license
© Sven Apel, William R. Cook, Krzysztof Czarnecki, Oscar M. Nierstrasz

Seminar Motivation

Feature orientation is an emerging paradigm of software development. It supports partially or completely automated generation of software systems in a domain based on features—units of functionality covering the domain. The key idea of feature orientation is to emphasize the similarities of a family of software systems for a given application domain (e.g., database systems, banking software, and text processing systems) by treating features as first-class entities throughout the entire development lifecycle and across all the software artifacts, with the goal of reusing software artifacts among the family members. For example, features of a database system could be transaction management, query optimization, and multi-user operation, and those of a text processing system could be printing, spell checking, and document format conversions.

Except where otherwise noted, content of this report is licensed under a Creative Commons BY-NC-ND 3.0 Unported license
Feature-Oriented Software Development (FOSD), *Dagstuhl Reports*, Vol. 1, Issue 1, pp. 27–41
Editors: Sven Apel, William Cook, Krzysztof Czarnecki, and Oscar Nierstrasz
Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A key software engineering challenge is that a feature does not necessarily map cleanly to an isolated module of code. Rather, it may affect ("cut across") many components of a modular software system. For example, the feature transaction management would affect many parts of a database system, e.g., query processing, logical and physical optimization, and buffer and storage management.

The concept of feature orientation is still in its infancy. However, a growing community of researchers have been working on it for years, and there are related, well-known concepts of software engineering with well-populated research communities, e.g., software product lines, aspect-oriented software development, service-oriented architecture, and model-driven engineering. The main goal of the Dagstuhl seminar on FOSD was to gather researchers and practitioners who are active in these different communities to discuss the roots, state of the art, and future directions of FOSD research and practice and to strengthen the identity of the feature orientation community. We think that this seminar met this goal. A overview of the seminar organization and a summary of results are given below.

Seminar Organization

As a warm-up for the seminar we conducted a survey on FOSD. The idea was to ask the emerging community what they think FOSD was about. We asked the following seven questions:

1. What do you think are the distinguishing concepts and ideas of FOSD?
2. What do you think are the major challenges in FOSD?
3. Which success stories of FOSD do you know?
4. What is missing in FOSD to adopt it in industry?
5. Is FOSD sufficiently visible in the software engineering community?
6. What do you expect to get out of the week?
7. What format and what kind of activities are you interested in (tutorials, demos, talks, breakout groups, brainstorming, social events, etc.)?

Based on the responses of 27 participants (available at the seminar's website), we prepared an introductory presentation on FOSD that aimed at "synchronizing" the participants, which is especially important in a field that is still in its infancy. After the self-introductions of all of the 49 participants and the introductory presentation, we allocated slots for the "hot" topics in the field of FOSD. On Monday, we had a discussion session of feature modularity. Tuesday was dedicated entirely to feature interactions. On Thursday, we had a mix of discussions sessions on industry adoption, the relation of FOSD to other development paradigms, as well as automatic product generation based on FOSD. On Tuesday and Thursday, we had demo sessions in the evening; on Wednesday, we had breakout sessions and a social event. Finally, on Friday, we had two wrap-up sessions, one concluding the individual discussions of the breakout groups and one summarizing the seminar and discussing results and further action items.

Seminar Results

From the organizers' perspective, the seminar was successful, although the large number of participants pushed the Dagstuhl concept to its limits. The topic attracted a lot of

interest (the seminar was fully booked), and during the seminar there were many very lively and sometimes quite controversial discussions. Many participants contributed actively by organizing mini-tutorials, discussion sessions, and breakout groups. The results of the discussion sessions and the breakout groups are available at the seminar's website.

The participants used the seminar as an opportunity to learn about each others work and to establish collaborations, which will bear fruit in the years to come. As a first tangible outcome, we would like to point out the list of resources that the seminar's participants developed in a team effort:

- Key FOSD papers
- Annotated bibliographies in the portal researchr.org
- A suite of benchmark problems
- Teaching material on FOSD

The details of this list are described on the seminar's website. Further discussion points were how to promote FOSD in the future, how to further strengthen the community, and how to collaborate in an efficient manner.

In summary, we conclude that the seminar was constructive and largely met its goals. Dagstuhl provided a productive and interactive atmosphere. It was certainly a key event in the maturation of the FOSD community.

Acknowledgement

The editors would like to thank Kacper Bąk for his help in collecting the summary material from the participants and compiling this report.

2 Table of Contents

Executive Summary
Sven Apel, William R. Cook, Krzysztof Czarnecki, Oscar M. Nierstrasz 27

Overview of Talks

Feature-Aware Verification
Sven Apel 32

Feature Interactions: the Good, the Bad, and the Ugly
Joanne Atlee 32

Clafer Demo
Kacper Bąk 32

FOSD—A Science of Software Design: A Personal and Historical Perspective
Don Batory 33

Model checking feature-based specifications with SNIP
Andreas Classen 33

Context-oriented Programming: First-class layers at runtime
Pascal Costanza 34

Choice Calculus Mini Tutorial
Martin Erwig 34

Re-Thinking Product Line Verification as a Constraints Problem
Kathi Fisler 35

Type Checking entire Product Lines
Christian Kästner 35

Modularity vs. Virtual Separation of Concerns in Feature-Oriented Implementations
Christian Kästner 36

A Survey of Product Family Algebra
Bernhard Möller 36

Testing Software Product Lines
Sebastian Oster 36

Event-B and Rodin Overview
Michael Poppleton 37

FOSD Adoption in Industry (Plenary Session Summary)
Rick Rabiser 37

Feature-Oriented Requirements Engineering—An Integrated Modeling, Aspect and Decision Based Approach
Reinhard Stoiber 38

Working Groups

Variability Representation (Breakout Session Summary)
Kacper Bąk 38

Non-functional versus Function Properties in Feature-oriented Product Line Generation (Breakout Session Summary)
Charles Krueger 39

Feature Location (Breakout Session Summary)
Christian Kästner 39

Participants 41

3 Overview of Talks

3.1 Feature-Aware Verification

Sven Apel (University of Passau, DE)

License Creative Commons BY-NC-ND 3.0 Unported license
© Sven Apel
Joint work of Apel, Sven; Speidel, Hendrik; Boxleitner, Stefan; von Rhein, Alex; Wendler, Philipp; Beyer, Dirk

The verification of a software system is challenging. Verifying a software product line adds two aspects to the challenge. First, a software product line represents a set of software systems, and thus we have to verify that all products (distinguished in terms of their features) fulfill their specifications. Second, we have to make sure that the features of a product work properly together. Feature interactions —situations in which the combination of features leads to emergent and possibly critical behavior— are a major source of failures in software product lines. We propose feature-aware verification, a verification approach that is aware of features and their combinations. It supports the specification of feature properties along with the features in separate and composable units. We encode the verification problem (i.e., checking for critical feature interactions) such that we can apply off-the-shelf model-checking technology. Furthermore, we integrate the technique of variability encoding, which allows us to verify a product line, without generating and checking a possibly exponential number of feature combinations. To demonstrate the feasibility of our approach, we implemented feature-aware verification based on standard model-checking and applied it to a case study that incorporates the domain knowledge of AT&T on e-mail systems.

3.2 Feature Interactions: the Good, the Bad, and the Ugly

Joanne Atlee (University of Waterloo, CA)

License Creative Commons BY-NC-ND 3.0 Unported license
© Joanne Atlee

This talk will be an overview of the research on feature interactions from the perspective of the FI community. It will include the "feature interaction problem" (which is NOT that features sometimes interact), general approaches to detecting and resolving interactions, and the trend toward architectural solutions.

3.3 Clafer Demo

Kacper Bąk (University of Waterloo, CA)

License Creative Commons BY-NC-ND 3.0 Unported license
© Kacper Bąk
Joint work of Bąk, Kacper; Czarnecki, Krzysztof; Wąsowski, Andrzej
Main reference Kacper Bąk, Krzysztof Czarnecki und Andrzej Wąsowski, "Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled," pp. 102–122, Software Language Engineering, LNCS 6563, Springer-Verlag, 2011.
URL http://dx.doi.org/10.1007/978-3-642-19440-5_7

In the demo we present Clafer, a meta-modeling language with first-class support for feature modeling. We designed Clafer as a concise notation for meta-models, feature models, mixtures

of meta- and feature models (such as components with options), and models that couple feature models and meta-models via constraints (such as mapping feature configurations to component configurations or model templates). Clafer also allows arranging models into multiple specialization and extension layers via constraints and inheritance. We show how to model a car telematics product line in Clafer, and how to generate an instance of it in the Alloy analyzer. Clafer models are translated to Alloy via clafer2alloy.

3.4 FOSD—A Science of Software Design: A Personal and Historical Perspective

Don Batory (University of Texas at Austin, US)

License Creative Commons BY-NC-ND 3.0 Unported license
© Don Batory

I present a summary of my involvement with FOSD for the last 30 years, from a tutorial perspective.

3.5 Model checking feature-based specifications with SNIP

Andreas Classen (University of Namur, BE)

License Creative Commons BY-NC-ND 3.0 Unported license
© Andreas Classen

In software product line engineering, systems are developed in families and differences between systems of a product line are expressed in terms of features. The model checking problem for product lines is more difficult than for single systems because a product line with n features yields up to 2^n individual systems to verify.

We present SNIP, a tool for model checking product lines against temporal properties. Contrary to existing approaches, SNIP relies on an efficient mathematical structure for product line behaviour, that exploits similarities and represents the behaviour of all systems in a compact manner. This structure is used to model check all systems of the product line in a single step.

The tool comes together with an intuitive specification language based on Promela. Variability in the behaviour is specified by guarding statements with features. A guarded statement is only executed in products that contain the features of its guard. When checking properties, SNIP takes these guards and the feature model into account. For this, it uses the TVL feature modelling language.

SNIP puts the theoretical results of our ICSE 2010 paper "Model Checking Lots of Systems" into practice and makes them available to engineers through an intuitive specification language for behaviour and for feature models.

3.6 Context-oriented Programming: First-class layers at runtime

Pascal Costanza (Vrije Universiteit Brussel, BE)

License Creative Commons BY-NC-ND 3.0 Unported license
© Pascal Costanza
Main reference Robert Hirschfeld, Pascal Costanza, Oscar Nierstrasz: "Context-oriented Programming", in Journal of Object Technology, vol. 7, no. 3, March-April 2008, pp. 125–151.
URL http://dx.doi.org/10.5381/jot.2008.7.3.a4

Context-oriented Programming is strongly related to Feature-oriented Programming in that both approaches use layers as abstractions for expressing feature increments and behavioral variations of a software system. However, in Context-oriented Programming, layers are available as first-class entities at runtime that can be activated and deactivated with well-defined scopes, in order to enable a software system to change its behavior depending on the context of use.

I have presented a short overview of the notion of Context-oriented Programming, its essential elements, and a concrete realization in the form of the programming language ContextL.

3.7 Choice Calculus Mini Tutorial

Martin Erwig (Oregon State University, US)

License Creative Commons BY-NC-ND 3.0 Unported license
© Martin Erwig
Joint work of Erwig, Martin; Walkinshaw, Eric
Main reference The Choice Calculus: A Representation for Software Variation, ACM Transactions on Software Engineering and Methodology, 2011, to appear
URL http://web.engr.oregonstate.edu/~erwig/papers/abstracts.html#TOSEM11

We describe the basic elements of the *choice calculus*, a formal representation for software variation that can serve as a common, underlying representation for variation research, playing a similar role that lambda calculus plays in programming language research. We will sketch the syntax and semantics of the choice calculus and present several applications.

At the core of the choice calculus are *choices*, which represent different alternatives that can be selected. Choices are annotated by names, which group choices into *dimensions*. Dimensions provide a structuring and scoping mechanism for choices. Moreover, each dimension introduces the number of alternatives each choice in it must have and tags for selecting those alternatives. The semantics of the choice calculus is defined via repeated elimination of dimensions and their associated choices through the selection of a tag defined by that dimension.

The choice calculus obeys a rich set of laws that give rise to a number of normal forms and allow the flexible restructuring of variation representations to adjust to the needs of different applications.

Among the potential applications of the choice calculus are feature modeling, change pattern detection, property preservation, and the development of change IDEs.

3.8 Re-Thinking Product Line Verification as a Constraints Problem

Kathi Fisler (Worcester Polytechnic Institute, US)

License Creative Commons BY-NC-ND 3.0 Unported license
© Kathi Fisler
Main reference Colin Blundell, Kathi Fisler, Shriram Krishnamurthi, Pascal Van Hentenryck, "Parameterized Interfaces for Open System Verification of Product Lines," pp. 258–267, Proceedings of the 19th IEEE International Conference on Automated Software Engineering, 2004.
URL http://dx.doi.org/10.1109/ASE.2004.53

Software product-lines view systems as compositions of features. Each component corresponds to an individual feature, and a composition of features yields a product. Feature-oriented verification must be able to analyze individual features and to compose the results into results on products. Since features interact through shared data, verifying individual features entails open system verification concerns. To verify temporal properties, features must be open to both propositional and temporal information from the remainder of the composed product. This talk argues that we can handle both forms of openness by viewing product-line verification as a two-phase process of constraint generation and discharge, rather than as a conventional verification problem.

3.9 Type Checking entire Product Lines

Christian Kästner (Universität Marburg, DE)

License Creative Commons BY-NC-ND 3.0 Unported license
© Christian Kästner
Joint work of Kästner, Christian; Apel, Sven

I propose to give a very brief (6-10 minute) overview of concept of checking the product line instead of checking the derived products and how this can be applied to type systems. The idea is to highlight a common research trend and stipulate discussions about adopting all kinds of checks to check the entire product line.

In a nutshell, the idea of a variability-aware type system to check the product-line implementation containing all variability (before product derivation) instead of checking each (of potentially billions of) derived product in isolation. Although the number of potential products may be exponential, variability-aware type systems exploit knowledge about variability implementation and can separate intrinsic complexity from accidental complexity. We are currently trying to scale such type system to type check the Linux kernel with over 8000 features. We believe that moving analysis from derived products to the product-line implementation is a common theme that can and should also be applied to many other approaches.

3.10 Modularity vs. Virtual Separation of Concerns in Feature-Oriented Implementations

Christian Kästner (Universität Marburg, DE)

License Creative Commons BY-NC-ND 3.0 Unported license
© Christian Kästner

When implementing features, the question of modularity arises. Interactions between feature (intended and accidental) at domain level and implementation level pose serious questions of how to divide a system into modules and what benefits a strict modularization brings. At the same time, we argue that tool support can provide many benefits of modularity also for other forms of implementation, such as the notorious conditional compilation used frequently by practitioners.

In this talk, I like to give a brief overview of different implementation strategies, but mostly focus on open questions regarding modularity to initiate a discussion. As one potential solution I discuss some results of our experience with virtual separation of concerns and related approaches.

3.11 A Survey of Product Family Algebra

Bernhard Möller (Universität Augsburg, DE)

License Creative Commons BY-NC-ND 3.0 Unported license
© Bernhard Möller
Main reference Peter Höfner, Ridha Khédri, Bernhard Möller, "Supplementing Product Families with Behaviour." Accepted for publication in the International Journal of Software and Informatics.

Using the well-known concept of a semiring we develop an algebra of product families with + standing for union and . for composition, like e.g. in regular expressions. The distributive semiring laws allow factoring out commonality and, conversely, algebraically calculating all possible products in a family. The algebra is enriched by a relation expressing that presence of one feature necessitates or excludes that of another one. Finally the "syntactic" view working only with feature names is supplemented by a semantic one, in which features are programs with a soundly defined semantics, in such a way that the axioms of product family algebra are still satisfied.

3.12 Testing Software Product Lines

Sebastian Oster (TU Darmstadt, DE)

License Creative Commons BY-NC-ND 3.0 Unported license
© Sebastian Oster
Main reference Sebastian Oster, Florian Markert, Philipp Ritter, "Automated incremental pairwise testing of software product lines," Proceedings of the 14th International Conference on Software product lines: going beyond, pp. 196–210, Jeju Island, South Korea, Springer-Verlag, Berlin, Heidelberg, 2010.
URL http://dx.doi.org/10.1007/978-3-642-15579-6_14

Testing Software Product Lines is very challenging due to a high degree of variability leading to an enormous number of possible products. The vast majority of today's testing approaches for Software Product Lines validate products individually using different kinds of reuse

techniques for testing. Due to the enormous number of possible products, individual product testing becomes more and more unfeasible.

Combinatorial testing offers one possibility to test a subset of all possible products. In this contribution we provide a detailed description of a methodology to apply combinatorial testing to a feature model of a Software Product Line. We combine graph transformation, combinatorial testing, and forward checking for that purpose. Additionally, our approach considers predefined sets of products.

3.13 Event-B and Rodin Overview

Michael Poppleton (University of Southampton, UK)

License Creative Commons BY-NC-ND 3.0 Unported license
© Michael Poppleton

We present the Event-B language for formal modelling, development and verification, and its supporting Rodin toolkit. This is now a leading state-based Formal Method, supported by a series of EU Framework STREP/IP grants. As introduction, the value added to a V-model is outlined. An example development is outlined to demonstrate the modelling, refinement, and V&V techniques, using the tools. Various Rodin plugin tools are described, those for decomposition/ composition being of particular interest (to a FOSD audience)

3.14 FOSD Adoption in Industry (Plenary Session Summary)

Rick Rabiser (Universität Linz, AT)

License Creative Commons BY-NC-ND 3.0 Unported license
© Rick Rabiser

In the discussion session *FOSD Adoption in Industry* three impulse talks were given about experiences of using FOSD and product line engineering in industry. More specifically, Christa Schwanninger reported experiences of feature-oriented development from Siemens AG; Charles Krueger presented experiences of introducing feature-oriented tools by BigLever Inc. in different large-scale organizations; and Paul Grünbacher reported project experiences of maturing a PLE research tool for industrial application within Siemens VAI.

The presenters discussed different industrial challenges motivating the use of FOSD. For example, the high number of configuration options in systems leading to long release durations; the lack of agreement on the scope of systems; or the competition forcing companies to perform explicit variability management. Main goals when applying FOSD in practice are to harness complexity, to improve the efficiency of production, and to reduce the time-to-market.

The session then focused on two particular areas for discussion, i.e., *What's already done in industrial practice?* and *What are the industrial challenges FOSD researchers should work on?*. The presentations and the discussion showed that FOSD is performed in different degrees in practice.

Variability management is done for diverse artifacts and using different tools ranging from simple spreadsheets to fully fledged PLE tool suites. Companies have their own and often very specific ways to perform FOSD. However, participants reported many challenges are remaining for FOSD researchers:

Handling of non-functional properties; feature-driven build management; standardization (e.g., of variability modeling, cf. CVL); simplification (esp. of user interfaces); integration with COTS apps and existing environments; support for industrial workflows; sharing of models and project management data; security; deployment support.

3.15 Feature-Oriented Requirements Engineering—An Integrated Modeling, Aspect and Decision Based Approach

Reinhard Stoiber (Universität Zürich, CH)

License Creative Commons BY-NC-ND 3.0 Unported license
© Reinhard Stoiber
Main reference Reinhard Stoiber, Martin Glinz, "Supporting Stepwise, Incremental Product Derivation in Product Line Requirements Engineering," pp. 77–84, Proceedings of VaMoS'10: Fourth International Workshop on Variability Modelling of Software-intensive Systems, Linz, Austria, 2010.
URL http://www.vamos-workshop.net/proceedings/VaMoS_2010_Proceedings.pdf

Considering variability modeling in requirements engineering is essential for developing, maintaining and evolving a software product line.

In this talk we introduce current techniques how practitioners and researchers typically deal with variability and point out open challenges in requirements engineering. Based on a real-world example, we demonstrate how our work on integrated requirements modeling, aspect-oriented modeling and Boolean decision modeling can help to overcome these problems. Our approach avoids information scattering of features over multiple diagrams, automates clerical as well as intellectual tasks for extracting and composing the variability, and automatically resolves and propagates variability constraints. This allows an easier creation of software product line models, improves the understanding of variability and its impact, and leads to an efficient and intuitive derivation of products that are correct by construction. Ongoing work addresses tool optimizations and a comprehensive validation.

4 Working Groups

4.1 Variability Representation (Breakout Session Summary)

Kacper Bąk (University of Waterloo, CA)

License Creative Commons BY-NC-ND 3.0 Unported license
© Kacper Bąk

The variability representation session focused on languages for modeling variability in source code and models. It started with an introduction to the choice calculus, a fundamental language for representing software variations. The choice calculus aspires to play a similar role for variation research as lambda calculus does in programming language research. The presentation covered basic ideas behind the choice calculus (choices and dimensions) and explained how choices get eliminated to derive a single product. The designers of the choice calculus were challenged to show how to translate feature models to the choice calculus. Even though the resulting models may be verbose, the translation is indeed possible.

Later the session turned to the Common Variability Language, which is the upcoming OMG standard for introducing variability into existing models. CVL is composed of several layers, of which two were particularly relevant to the session: variability realization and

variability abstraction. The former includes variation points that have concrete impact on the base model, while the latter shows variation at a higher level.

The main conclusion was that the choice calculus might be a suitable theory for variation points.

The last part of the session concerned static versus dynamic variability. Static variability is resolved at compile time, while dynamic features are loaded at the run-time. One question discussed was this: What should happen when user loads a dynamic feature which is in conflict with an existing feature? Participants discussed soft and hard constraints in the context of mobile applications. Soft constraints can be violated, while hard constraints must always hold. Mobile applications are user-centric, thus user preferences should also be taken into account when two features are in conflict. A proposed solution was to use probabilistic feature models to attach user preferences to features.

4.2 Non-functional versus Function Properties in Feature-oriented Product Line Generation (Breakout Session Summary)

Charles Krueger (BigLever, Austin, US)

License Creative Commons BY-NC-ND 3.0 Unported license
© Charles Krueger

With feature-oriented product line engineering, stakeholders exploit feature models to understand and manage functional properties during the generation of a system. Although functional behavior of a system is the primary concern, the associated non-functional properties – such as resource usage, performance, cost and perceived product value – are also important concerns in industry settings.

The common scenarios include (1) adjusting functionality in order to bring one or more non-functional properties with specified bounds, (2) expanding functionality to increase system value by taking advantage of underutilized capacity in a non-functional property. The relationship between features and non-functional properties may be hard or easy to measure and control, additive, binary, non-monotonic or chaotic.

How can feature-oriented approaches simultaneously manage the desired functional and non-functional properties of a system under generation? Measurements and statistical modeling of non-functional properties relative to feature selections can be incorporated into generators to provide guidance.

Adjacent fields that can contribute include operations research, AI and satisfaction solvers.

4.3 Feature Location (Breakout Session Summary)

Christian Kästner (Universität Marburg, DE)

License Creative Commons BY-NC-ND 3.0 Unported license
© Christian Kästner

In a breakout session on feature location, we assembled a group of people who pursue different location strategies. After introducing the respective concepts or tools, we soon realized that the goals differ significantly. We identified three main higher level goals: locating features for maintenance tasks, for design decisions, and for reuse. Some participants pursue mainly one of these goals, for example CodeX for maintenance, Portolio and CIDE for reuse, and

Feature Unweaving for Design, whereas several tools also pursued multiple goals at the same time, e.g., Linux Analysis and the Requirements to Test mappings.

We concluded that many approaches are complementary and we discussed possible integrations. A main outcome was the agreement to share case studies and the formulation of a vision of an entire process that spans feature-location tasks from design to implementation.

Participants

- Sven Apel
Universität Passau, DE
- Joanne Atlee
University of Waterloo, CA
- Kacper Bak
University of Waterloo, CA
- Don Batory
Univ. of Texas at Austin, US
- Thorsten Berger
Universität Leipzig, DE
- Götz Botterweck
University of Limerick, IE
- Andreas Classen
University of Namur, BE
- William R. Cook
University of Texas - Austin, US
- Pascal Costanza
Vrije Universiteit Brussel, BE
- Krzysztof Czarnecki
University of Waterloo, CA
- Ben Delaware
University of Texas - Austin, US
- Zinovy Diskin
University of Waterloo, CA
- Christoph Elsner
Universität Erlangen-Nürnberg, DE
- Martin Erwig
Oregon State University, US
- Kathi Fisler
Worcester Polytechnic Institute, US
- Martin Glinz
Universität Zürich, CH
- Gerhard Goos
KIT - Karlsruhe Institute of Technology, DE
- Mark Grechanik
Accenture Labs - Chicago, US
- Paul Grünbacher
Universität Linz, AT
- Florian Heidenreich
TU Dresden, DE
- Robert Hirschfeld
Hasso-Plattner-Institut - Potsdam, DE
- Christian Kästner
Universität Marburg, DE
- Kyo-Chul Kang
POSTECH - Pohang, KR
- Hans Körber
FH Kaiserslautern-Zweibrücken, DE
- Shriram Krishnamurthi
Brown University - Providence, US
- Charles Krueger
BigLever- Austin, US
- Ingolf Krüger
University of California - San Diego, US
- Ralf Lämmel
Universität Koblenz-Landau, DE
- Jaejoon Lee
Lancaster University, GB
- Thomas Leich
METOP GmbH - Magdeburg, DE
- Christian Lengauer
Universität Passau, DE
- Bernhard Möller
Universität Augsburg, DE
- Oscar M. Nierstrasz
Universität Bern, CH
- Sebastian Oster
TU Darmstadt, DE
- Holger Papajewski
pure-systems GmbH - Magdeburg, DE
- Joe Politz
Brown University - Providence, US
- Michael Poppleton
University of Southampton, GB
- Christian Prehofer
LMU München, DE
- Rick Rabiser
Universität Linz, AT
- Jorge Ressia
Universität Bern, CH
- Bernhard Rumpe
RWTH Aachen, DE
- Klaus Schmid
Universität Hildesheim, DE
- Christa Schwanninger
Siemens AG - Erlangen, DE
- Steven She
University of Waterloo, CA
- Olaf Spinczyk
TU Dortmund, DE
- Reinhard Stoiber
Universität Zürich, CH
- Salvador Trujillo
Ikerlan Research Centre - Arrasate-Mondragón, ES
- Eric Walkingshaw
Oregon State University, US
- Andrzej Wasowski
IT University of Copenhagen, DK