Design For FUTURE

Deutsche Forschungsgemeinschaft DFG

DFG SPP 1593

Kick-Off Workshop

Programme

September 11th – 14th, 2012

Goslar

DFG Priority Programme 1593
Design For Future - Managed Software Evolution

Coordination Board

Prof. Dr. Ursula Goltz
Coordinator

Prof. Dr. Ralf H. Reussner
Co-Coordinator

Prof. Dr. Michael Goedicke

Prof. Dr. Wilhelm Hasselbring

Prof. Dr.-Ing. Birgit Vogel-Heuser

Coordination Assistance

Dipl.-Wirt.-Inform. Lukas Märtin
Workshop Organisation

Table of Contents

	Di., 11.9.	Mi., 12.9.	Do., 13.9.	Fr., 14.9.	
09:00			Gemeinsame Aktivitäten (Märtin) (40 Min.)	Benchmarking, Stellenwert von Fallstudien (Hasselbring) (20 Min.)	
09:30		Eröffnung und Vorstellungsrunde (Goltz)	2 Projektvorstellungen (40 + 10 Min.) 1) Vogel-Heuser, Broy 2) Grunske, Tichy	Vorstellung einer IS-Fallstudie (Reussner) (20 Min.) Vorstellung einer AS-Fallstudie (Vogel-Heuser) (20 Min.)	
10:00				Allgemeine Diskussion zu behandelten Fallstudien	
10:30		Kaffeepause	Kaffeepause	Kaffeepause	
11:00		*4 Projektvorstellungen (80 + 10 Min.)* 1) Goltz, Schürr 2) Goedicke, Reussner 3) Beckert, Klebanov	*4 Projektvorstellungen (80 + 10 Min.)* 1) Apel 2) Fay, Lamersdorf 3) Koschke, Schmid	Parallel: Diskussionen zu Evolutionsszenarien in behandelten Fallstudien Informationssysteme (Reussner)	Automatisierungstechnik (Vogel-Heuser)
12:00		4) Jürjens, Schneider	4) Kelter, Taenzer	Fazit und Abschluss (Goltz, Reussner)	
12:30		Mittagessen	Mittagessen	Mittagessen (optional, Selbstzahler)	
13:30		*3 Projektvorstellungen (60 Min.)* 1) Hasselbring, Pohl, Reussner 2) Schaefer, Tribastone, Prehofer 3) Brügge, Paech	Motivation Schwerpunkt M&P - Methods and Processes (Hasselbring)		
14:00			Diskussion Schwerpunkt M&P		
14:30		Motivation Schwerpunkt KCS - Knowledge Carrying Software (Goedicke)			
15:30		Kaffeepause	Kaffeepause		
16:00		Diskussion Schwerpunkt KCS	Metaplan-Session (Hasselbring,Märtin)		
17:00					
18:00					
19:00	Abendessen	Abendessen	Abendessen		

Apel: Techniques and Prediction Models for Sustainable Product-Line Engineering

Software product line engineering has gained considerable momentum in recent years, both in industry and in academia. Companies and institutions such as NASA, Hewlett Packard, General Motors, Boeing, Nokia, and Philips apply product-line technology with great success to sustain their development by broadening their product portfolio, improving software quality, shorting time to market, and being able to react faster to market changes. However, pursuing a product-line approach implies often an up-front investment for future benefits. Product-line developers have to anticipate which features will be desired by customers in the future. So, prediction models play an important role to avoid uneconomic developments. However, contemporary prediction models largely ignore structural and behavioral properties of the architecture and implementation assets of a product line. For example, modifying the transaction management of a database system is by far more expensive and risky than modifying its command-line interface. We propose to rethink contemporary prediction models and to employ state-of-the-art analysis techniques to create a richer knowledge base for predictions based on implementation knowledge, including software metrics, static analysis, mining techniques, measurements of non-functional properties, and feature-interaction analysis.

Beckert, Klebanov: Regression Verification for Evolving Object-Oriented Software

The goal of this project is to leverage advances in deductive program verification to enable regression verification, i.e., proving formally that software remains correct through its evolution, and no new bugs are introduced. We aim to develop a regression verification methodology for a real object-oriented language (Java) that has the reach and power to be applied to real-world software. Even though building software with high quality from the beginning of the software lifecycle is crucial, it is not enough, since long-living software is adapted and evolves during its development and after its release. While traditional regression testing techniques are commonly used to give confidence in the reliability of evolving software, more powerful and reliable techniques are required. Over the last decade, there has been tremendous progress in the area of program verification. However, in the development of formal methods not enough attention has been given to software changes, which occur during software evolution. A very promising solution is to develop regression verification methods. They are a natural extension of regression testing. Given two programs that are both complex but similar to each other, much less effort is required to prove their equivalence than to prove that they satisfy a (complex) functional specification. The effort for proving equivalence mainly depends on the difference between the programs and not on their overall size and complexity. Our vision and goal is to develop regression verification methods powerful enough to be applicable to real-word software (and its evolution). We will develop regression verification methods for Java that can be used to prove that two Java programs are equivalent. We also address the problem of changing requirements by proving that two programs are not fully equivalent but differ in a formally specified way. We will implement these methods in a regression verifica-

tion system based on our Java verification tool KeY. Regression verification shows its power (and only makes sense) as part of software evolution. It is central to the success of the proposed project that we integrate our methods into the software development and adaptation process. In particular, we plan to integrate regression verification with refactoring and reengineering of software, with software product line techniques, and with test generation and test selection. Our project thus addresses the priority programme's guiding theme of methods and processes, contributing to the programme topics of (a) model-based and model-driven development of long-living systems, (b) traceability from requirements to architecture and code, and (c) continuous software system evolution under design (and to a lesser extent also runtime) control and management. We link to the application domains using case studies from the area of information systems, in particular the priority-programme-wide CoCoME case study.

Brügge, Paech: Usage- and Rationale-based Evolution Decision Support

For software evolution decisions developers need knowledge of the current and future deployment context as well as knowledge of the software and its development artifacts. Typically this knowledge is documented only partially and often only in unrelated fragments, and therefore it is not fully exploitable. In addition, the reasoning underlying the decisions made in previous releases can also change. Thus, it is an important challenge to ease the capture of this knowledge and to improve the decision process. The vision of the URES project is a continuous decision process over the whole software life-cycle where

• Developers reflect the actual user behavior in their evolution decisions. User behavior is automatically captured during operation and related to system models so that necessary changes to the software can be identified.

• Developers reflect system, project and operation knowledge in their long-term decisions and in particular the rationale of decisions made for previous releases. Links between system, project and operation knowledge are automatically captured and maintained. This allows providing consistent linkage between decisions and artifacts (incl. code) through rationale so that the impact of changes of the artifacts on the decisions and vice versa can be analyzed.

To empirically validate this vision we will develop corresponding methods and tools and apply them both, in a research environment and in an industrial case study, to a long-running system.

Broy, Vogel-Heuser: Model-Driven Evolution Management Framework for Automation Systems (MoDEMAS)

Automation systems constitute multi-disciplinary, software-intensive and long-living systems. System modifications to meet a constantly changing set of requirements (also called evolution steps in this proposal) regularly require long downtimes and critical test phases upon start-up. Hence, changes in the automation system have to be avoided whenever possible. As current and future markets demand higher flexibility regarding customer specific products (up to lot-size one concepts), the time between evolution steps needs to be decreased drastically. In fact, it is expected that in the future evolution of automation systems has to be managed in day-to-day routine. This project aims at enabling this routine through a model-driven engineering methodology including respective engineering process models and formalisms (1) to define and structure system requirements and (2) to describe and analyze system architectures and component structures/behavior.

Therefore, typical evolution scenarios and established project execution strategies and work practices are studied initially. The analysis results serve as the basis for developing a respective model-driven engineering method. Therefore, FOCUS – a well-elaborated model-based engineering methodology particularly suited for developing reactive software systems – is adapted substantially to meet the particular requirements of evolution in the automation system domain regarding both content and usability. In particular, the provided formalisms are extended to capture cross-discipline architectural, structural and behavioral system aspects that need to be preserved along the lifecycle of automation systems. Based on a rigorous system model complementary analysis techniques are studied, which allow verifying system properties and validating system designs and their respective model-based implementations early in the evolution process. Finally, a model

of the resulting engineering process and respective engineering methods is derived with the goal to facilitate establishing the proposed framework within organizations.

Fay, Lamersdorf: Forever Young Production Automation with Active Components (FYPA²C)

Software in productive use suffers from degeneration, caused e.g. by changing usage conditions or ad-hoc modifications not reflected in the latest software specifications. The FYPA²C project aims at addressing such software degeneration by an "anti-aging cycle" that perpetually reinforces the consistency of software specifications, their corresponding implementations, and their actual usage. Main contributions are (1) a combined meta-model for production automation software and verifiable requirements, (2) automated improvement of requirements specifications by learning from actual system behaviour, and (3) an infrastructure for online monitoring and simulation-based validation of requirements. The meta-model is based on "Active Components" and allows to specify the application, the underlying technical production process, and the requirements in ways in which they can be validated by corresponding test cases. Based on the knowledge of the system components and the production process, an online monitoring and learning infrastructure will continuously analyze and store system behaviour. When applying modifications to the software, it can be tested against all specified and learned requirements using the simulation-based validation. As a result, software developers and system operators are provided with an approach and tool suite for managing knowledge about software usage and requirements in order to keep such software "forever young".

Goltz, Schürr: Integrated Model-based Testing of Continuously Evolving Software Product Lines (IMoTEP)

Automation engineers are regularly faced with the problem to maintain long living safety critical process control software families. A promising paradigm for developing this type of software efficiently is (dynamic) software product line (SPL) engineering; it supports the systematic development of software product families of similar applications. However, existing SPL approaches offer only little support for integrated quality assurance and evolution of software over time. In particular, there exist only few approaches and no concise methodology for testing evolving SPLs and their applications until now. For automation systems, there is a particular need for such a support, since here modifications often have to be conducted under hard safety constraints—sometimes even in a running system. We hence propose an approach for managed dynamic SPL evolution in the automation engineering domain with a particular focus on efficient model-based testing techniques. SPL test suites needed for (1) systematic offline testing of representative sets of products in their development environments and (2) online testing of reconfigurating products in their runtime environments are generated and incrementally updated using a mixture of model checking, constraint solving, and model transformation techniques. Integrated general purpose feature modeling and domain-specific (test) modeling languages are used together with standard model- and new feature model/interaction-based coverage criteria for that purpose. The involved models and their metamodels as well as the associated model processing algorithms are used both at design and runtime. The overall aim is to support both predefined reconfigurations of products and unforeseen evolution of full product lines with a specific focus on the adaptation of their test suites.

Goedicke, Reussner: Concepts, methods and tools for architecture - and quality-centric evolution of long-living software systems

Nearly all aspects of our live are affected by long-living software systems. Software is aging when necessary changes are not performed to meet the requirements in its changing environment, or when the software is changed in a problematic way. These processes are part of the software evolution. The typical approach to meet evolution in current software projects is ad-hoc change of the implementation, often ignoring other development artifacts (i.e. requirement documents and design models), and without evaluating evolution alternatives. Software evolution can become problematic, because aging software is hard to maintain and it does not meet increasing external quality requirements, such as improved performance and increased reliability. For increasing the maintainability of software, the development artifacts need to be consistent with each other and up-to-date. To meet increasing external quality requirements and to account for changes in the environment, the system has to be changed regularly. The goal of this project is to develop concepts, methods, and tools for keeping the consistency between the development artifacts, and for systematically identifying and performing the necessary changes on a system to meet the quality requirements in a changing environment. With this means, the manageability of software evolution will be enhanced. Thus this work addresses the often negative interference between external (performance, reliability) and internal (maintainability) quality requirements. Our methods and tools will be evaluated using case studies from the information systems domain.

Grunske, Tichy: ENsurance of Software evolUtion by Run-time cErtification

Quality attributes play an important role in different classes of software systems, e.g. safety in embedded systems and performance in business information systems. Currently, quality requirements are typically checked at design time. For evolving systems with changing environmental conditions this leads to the problem that the system may behave differently with respect to quality attributes than analyzed at design time. We propose to address this problem by developing a holistic model-driven approach, which treats quality evaluation models as first class entities. This approach uses dedicated model transformations to evolve quality evaluation models with structural and behavioral models. Furthermore, the models will be continuously updated with statistical monitoring techniques to estimate model parameters like usage profiles and failure rates. As a result of this approach, we will be able to certify software evolution steps with consistent models.

Hasselbring, Pohl, Reussner: Integrated Observation and Modeling Techniques to Support Adaptation and Evolution of Software Systems

The increased adoption of service-oriented technologies and cloud computing creates new challenges for the adaptation and evolution of long-living software systems. Software services and cloud platforms are owned and maintained by independent parties. Software engineers and system operators of long-living software systems only have limited visibility and control over those third-party elements. Traditional monitoring provides software engineers and system operators with execution observation data which are used as basis to detect anomalies. If the services and the cloud platform are not owned and controlled by the engineers of the software systems, monitoring the execution of the software system is not straightforward.

The aim of the iObserve project is to develop and validate advanced techniques which empower the system engineers to observe and detect anomalies of the execution of software systems they do not fully own and control. It will extend and integrate previous work on adaptive monitoring, online testing and benchmarking and will use models@runtime as means to adjust the observation and anomaly detection techniques during system operation. To demonstrate the feasibility and potential benefits gained and for providing feedback to guide the research, the results will be continuously evaluated using an established research benchmark (CoCoME) as well as an industry-driven open-source application (Eclipse Skalli) that runs on a cloud platform.

Jürjens, Schneider: Beyond One-Shot Security: Keeping Information Systems Secure through Environment-Driven Knowledge Evolution (SecVolution)

Information systems are exposed to constantly changing environments which require constant updating. Software "ages" not by wearing out, but by failing to keep up-to-date with its environment. Security is an increasingly important quality aspect in modern information systems. At the same time, it is particularly affected by the above-mentioned risk of "software ageing". When an information system handles assets of a company or an organization, any security loophole can be exploited by attackers. Advances in knowledge and technology of attackers are part of the above-mentioned environment of a security-relevant information system. Outdated security precautions can, therefore, permit sudden and substantial losses. Security in long-living information systems, thus, requires an on-going and systematic evolution of knowledge and software for its protection. Our objective is to develop techniques, tools, and processes that support security requirements and design analysis techniques for evolving information systems in order to ensure "lifelong" compliance to security requirements. We will build on the security requirements & design approach SecReq developed in previous joint work. As a core feature, this approach supports reusing security engineering experience gained during the development of security-critical software and feeding it back into the development process. We will develop heuristic tools and techniques that support elicitation of relevant changes in the environment. Findings will be formalized for semi-automatic security updates. During the evolution of a long-living information system, changes in the environment will be monitored and translated to adaptations that preserve or restore its security level.

Kelter, Taentzer: Specifying and Recognizing Model Changes Based on Edit Operations

Model-based software development has become a widely accepted approach for embedded systems and in application domains where software must be maintainable and long-living. Models are subject to continuous change and have many versions during their lifetime. The specification and recognition of changes in models is the key to understand and manage the evolution of a model-based system. However, currently available versioning tools operate on low-level, sometimes tool-specific model representations. The resulting differences are often not understandable. It is a largely open problem how to use high-level edit operations, e.g. as offered by modern refactoring tools, to present and handle model differences and to analyse the evolution of models. The tight integration of editing and versioning tools requires consistent specifications of edit operations; this integration is another open problem. This project adresses both problems by consistently lifting model versioning concepts, techniques, and tools from low-level to high-level model changes. All concepts shall be formalized by graph transformation concepts in order to reason about complex model modifications and their interrelations. Our approach will be implemented and evaluated based on the widely used Eclipse Modeling Project.

Koschke, Schmid: Support for correct evolution of software product lines

Today software is often developed as a set of related products that are derived from a common infrastructure, a so-called software product line. A major issue in product line engineering is the continuous evolution of the product line as all products are intimately connected and the total lifetime of a product line is longer than that of any of its derived products.

In this project, we will study long-living software product lines and their continuous evolution with a specific focus on embedded systems and in particular industry automation systems. In these domains, variability is often implemented statically through preprocessor directives or dynamically through setting configuration variables at initialization time or later at runtime. Despite their relevance, these variability techniques are still only insufficiently researched.

We will develop techniques to check the integrity of a product line implementation whenever it is changed during its evolution. That is, as opposed to most other work on the analysis of product lines, the focus will not be on the analysis of one particular version of a product line, but on the difference introduced when modifying a product line. The project will combine reverse engineering, program analysis, and formal product line analysis to discover the introduction of flaws through evolutionary changes. These flaws often arise from unintended side effects of evolution activities related to a feature.

The project will take a comprehensive approach, taking into account maintenance actions related to the variability model as well as implementation changes and also from combinations of both.

Schaefer, Tribastone, Prehofer: Scalable design and performance analysis for long-living software families

Long-living software systems are typically available in a rich set of variants to deal with differing customer requirements and application contexts. Furthermore, users are often given the possibility to change to a different configuration online to dynamically adapt to varying environmental conditions. In addition to satisfying functional requirements, such changes are to preserve existing service-level agreements. The focus of this project is to define a methodology for expressing system variability and its impact on performance. Motivated by its widespread use in certain domains, we consider a model-driven approach based on behavioral models, using notions of delta modeling and feature composition enriched with information needed to automatically derive a performance model. We are concerned with the usually very large number of variations in models of realistic systems, which impede naive approaches based on exhaustive exploration for predictive purposes, especially at runtime when requirements on execution times are stringent. We offer a symbiotic approach which harnesses a structure of variability inferred by deltas to efficiently analyzing the whole configuration space, for instance by pruning certain subspaces with provably inferior estimated performance. The approach will be practically applied to the dynamic throughput optimization of a software-controlled automated assembly line, chosen as a representative case study of variant-rich long-lived software systems where changes are ideally applied online to avoid costly interruptions.