Design For FUTURE

DFG

DFG Priority Programme 1593
Design For Future - Managed Software Evolution

DFG Priority Programme 1593

Design For Future – Managed Software Evolution

Talk Abstracts

Design For FUTURE

DFG

DFG Priority Programme 1593
Design For Future - Managed Software Evolution

Table of Contents

Design For FUTURE
DFG Priority Programme 1593
Design For Future - Managed Software Evolution

Concepts, Methods and Tools for Architecture- and Quality-centric Evolution of Long-living Software Systems (ADVERT)

M. Goedicke, R. Reussner

The typical approach to meet software evolution is the ad-hoc change of the implementation, often ignoring other development artefacts (e.g., requirement documents and design models). In the ADVERT project we developed methods and tools for creating and keeping the consistency between documented requirements, architecture models, and program code. They support developers to derive alternative evolution paths for architectures from a set of requirements and existing program code. This includes a tool that proposes architectural alternatives based on architectural patterns and tools that keep architecture models and program code in a consistent state. The approach helps to manage the relationships between different development artefacts, and therefore contributes to the idea of knowledge-carrying code.

Continuous Usage- and Rationale-based Evolution Decision Support (CURES)

B. Paech, B. Brügge

Improving Continuous Software Engineering through Usage and Decision Knowledge

Continuous software engineering (CSE) evolved into a widely adopted process for developing and improving software systems. It is characterized by frequent and rapid development cycles and relies on the combination of different activities, such as agile development, continuous integration, or continuous delivery.

We see CSE as an opportunity to support stakeholders in capturing and exploiting usage and decision knowledge. Both knowledge types are essential for software evolution. In particular, usage knowledge provides insights into the users' acceptance of a software increment and decision knowledge covers the developers' knowledge about decisions and their justification that is necessary to understand and change the software system.

In this talk we present results from an interview study on CSE and its combination with usage and decision knowledge in current industry practices. We also describe our approach on how to integrate both knowledge types into CSE. Using an example workflow, we show how our tools CuuSE and ConDec support user understanding as well as decision documentation and exploitation for software evolution.

Design For FUTURE

DFG Priority Programme 1593
Design For Future - Managed Software Evolution

Scalable design and performance analysis for long-living software families (DAPS2)

I. Schaefer

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 behavioural models, using notions of delta modelling 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 analysing 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.

Declarative Performance Engineering (DECLARE)

A. van Hoorn, S. Kounev

During the life-cycle of a software system, performance analysts continuously need to provide answers to and act on performance-relevant concerns about response times, resource utilization, bottlenecks, trends, anomalies, etc. While there is a plethora of established methods, techniques, and tools for evaluating performance properties in various stages of a software system life-cycle, manifold challenges hinder the adoption of these techniques in practice. For example, it can be challenging to select and to parameterize a suitable solution strategy, as well as to filter and interpret the obtained results.

The goal of the DECLARE project was to address these challenges by introducing a Declarative Performance Engineering approach, thus bridging the abstraction gap between the level on which performance-relevant concerns are formulated and the level on which performance evaluations are actually executed.

In this talk, we summarize the main results, which are divided into the following four parts: First, we present our work on declarative languages, used for expressing these concerns. Then, we present approaches and tools for collecting the data required for answering the concerns, e.g., tools for extracting different models. Approaches and tools for the evaluation of stated concerns and their presentation are presented in the third part. In the fourth part, we present the infrastructure for the declarative approach itself, e.g., decision support for choosing the right evaluation approach. Finally, we will discuss lessons learned and our plans for the future work in the context of the project.

Domain-spanning Maintainability Estimation of Information and Manufacturing Automation Systems (DoMain)

B. Vogel-Heuser, R. Reussner

Information systems and manufacturing automation systems face very similar challenges with respect to evolution. Both struggle with changing requirements and architectures that have grown over time. Frequently, information systems as well as manufacturing automation systems are in operation over decades while they are continuously modified. Modifications may include corrections, improvements, or adaptations of the system to changes in its environment (e.g., hardware or software platform, technology, as well as user properties) and in requirements. Thus, maintainability is an important quality aspect, especially for long-living systems. Accordingly, researches in methods to improve maintainability are of scientific interest.

However, the effect of maintainability improving actions cannot be scientifically validated without the explicit use of case studies. There are two case studies involved in SPP 1593, namely Common Component Modelling Example (CoCoME) and Pick and Place Unit (PPU). In the context of the DoMain project, we integrated both case studies to an Industry 4.0 case study to provide support to projects in SPP 1593. Further, we developed a method to analyse the maintainability of software-intensive systems by deriving maintenance task lists, i.e. specific changes to be performed, from architecture descriptions. For assessing a system's architecture, the derived task list can be applied to make estimations about the maintainability of the proposed system design that are of high practical importance. Using the task lists, multiple architecture alternatives can be compared and, by that, change efforts can be identified.

The integration of these two case study demonstrators, CoCoME and PPU, does not only depict communication or the interaction simply between the two systems. The dynamic reconfigurable production has also been implemented to support various user orders with recipes. The related artefacts such as service-oriented architecture models and middleware to handle the connection in the both sides are provided to the community.

We proposed meta-models to reflect domain-specific constructs. Based on the meta-models, we provided a novel procedure to maintainability estimation by identifying change efforts for information systems and manufacturing automation systems. The procedure considers the interrelations between processes and the system as well as their effects on maintainability estimation while spanning over all life-phases of the system.

Ensurance of Software Evolution (ENSURE)

L. Grunske, M. Tichy

Software is an innovation driver; hence its quality needs to be ensured. ENSURE-II addresses this fact by a holistic model-driven approach. In this talk, we present the project results w.r.t. two challenges to developers during software evolution. First, the correct execution of the evolution and second efficient re-verification of quality attributes after the evolution.

First, we apply data mining techniques to model differences in order to find model transformations that are frequent during model evolution. This knowledge about previous evolutions can guide future evolutions via model repair techniques or through recommender systems for model transformations.

Second, probabilistic verification plays an important role for verifying the software's quality attributes such as reliability, safety and performance. Recently, incremental approaches have been found to be promising for the verification of evolving and self-* systems. However, substantial improvements have not yet been achieved for evaluating structural changes in the model. We describe formal foundations and a mathematical framework using regular expression trees to apply incremental verification.

Integrated Model-based Testing of Continuously Evolving Software Product Lines (IMoTEP 2)

I. Schaefer, M. Lochau

Mastering Variability in Space, over Time and at Runtime in Model-based Testing of Cyber-Physical Systems

The next generation of embedded software systems, frequently summarized under the term cyber-physical systems (CPS), may be described by three essential characteristics: CPS perform complex computations, CPS conduct control tasks involving continuous data- and signal-processing, and CPS are (parts of) distributed, and even mobile, communication systems. In addition, nowadays software systems have to cope with ever-growing extents of variability which, again, may come in three different flavors: variability in space by means of predefined configuration options (software product lines), variability at runtime by means of preplanned reconfigurations (runtime-adaptive systems), and variability over time by means of initially unforeseen updates to new versions (software evolution). Finally, depending on the particular application domain, CPS often constitute safety- and mission-critical parts of socio-technical systems. Thus, novel quality-assurance methodologies are required to systematically cope with the interplay between the different CPS characteristics and the different dimensions of variability. The talk gives an overview on state-of-the-art, ongoing research and open challenges in the specification and quality-assurance of CPS in the presence of variability. The research is illustrated using the PPU case study as well as a real-world medical-device software system. The talk focusses on evolving dynamic software product lines as engineering methodology and model-based testing as quality-assurance technique. Concerning variability in space, an automated test-generation approach based on symbolic model-checking is presented for efficiently covering highly-configurable source code. Extensions of these techniques addressed in the talk include reasoning about constraints on binding times of reconfigurable and runtime-adaptive software systems, freely configurable real-time behaviors using parametric timed automata as well as novel concepts for mutation-based testing of variable software. Finally, to handle unforeseen evolution, accompanying testing strategies comprise model-based techniques for reasoning about CPS artifact changes and their potential impact on crucial system properties, including model differencing and model merging of variable software, as well as delta-oriented model slicing and regression analysis.

Regression Verification in a User-Centered Software Development Process for Evolving Automated Production Systems (IMPROVE APS)

B. Beckert, M. Ulbrich, B. Vogel-Heuser

The vision for this project is to advance regression verification methods such that they can be applied to assess and increase the quality of automated production systems (aPS) during their evolution. APSs control complex manufacturing processes, such as mechanical, chemical and physical processes. They and their software are long-living systems subject to evolution. As mission-critical systems, they need to fulfil dependability and reliability criteria to avoid machine hazards or accidents.

The goal of regression verification is to formally prove that software remains correct through its evolution by showing that (1) changes have the desired effect, (2) new revisions show the same (partial) behaviour as old revisions, and (3) no new bugs are introduced. Regression verification avoids the main bottleneck for the practical use of formal verification, the need to write full functional specifications, and the main drawback of simulation, its lack of exhaustive coverage.

In the first project phase (IMPROVE) regression verification techniques for imperative, heap-based and object-oriented programs were developed and presented. In the second phase (IMPROVE APS), we extend and specialise the relational verification methods for automated production systems (aPS) as application domain. The research and results mainly focus on:

(1) Extending the reach and power of regression verification towards realistically sized production systems and change scenarios. A method to support the object-oriented extension of the standard IEC 61131-3 has been developed and implemented. The regression verification of larger software systems is supported using modularisation. One challenge is to find the right abstractions and translation concepts such that the verification engine succeeds on meaningful industrial case studies.
(2) A family of novel table-based specification languages that allow the application engineer to describe and analyse behavioural requirements in an expressible, comprehensible, and engineering-friendly manner. Besides defining the formal foundations, a verification engine and a user-friendly development environment for test tables have been developed.

Design For FUTURE

DFG

DFG Priority Programme 1593
Design For Future - Managed Software Evolution

Integrated Observation and Modeling Techniques to Support Adaptation and Evolution of Software Systems (iObserve)

W. Hasselbring, K. Pohl, R. Reussner

Cloud services promise many benefits, such as flexibility, scalability, reusability and economic use of resources, yet also lead to new software engineering challenges: (1) run-time changes of cloud services are not under the software engineers' control and, in most cases, cannot be anticipated during development; (2) software engineers have limited visibility into cloud services; (3) personal data may be distributed and decentralized across cloud services implying potential privacy concerns. This talk summarizes how iObserve addresses these challenges along the widely adopted MAPE (monitoring, analysis, planning, and execution) control loop model.

With respect to monitoring and analysis, iObserve introduced the notion of descriptive architectural runtime models to facilitate the automated analysis, but also human inspection of the cloud systems. In particular, iObserve focused on the detection of performance anomalies and privacy violations.
The architectural runtime models are derived from design time models or implementation code and updated via using dynamic monitoring data.

With respect to planning and execution, iObserve put emphasis on performing design space exploration to address multiple quality objectives (performance, privacy and cost), and supporting operator-in-the-loop adaptation. To this end, iObserve extended the role of architectural runtime models to also serve as prescriptive models for adaptation planning and execution.

To conceptually integrate the techniques for monitoring, analyzing, planning and executing adaptations, iObserve builds upon a holistic model-based approach. The CoCoME case study was extended and employed to evaluate the iObserve techniques, as well as a basis for collaboration with related SPP projects.

Linked Forever Young Production Automation with Active Components (LinkedFYPA²C)

A. Fay, W. Lamersdorf

Enhancing the evolution of previously undocumented systems by knowledge-carrying software

Automated Production Systems (aPS) should remain in operation over decades, despite changing requirements and frequent changes of their mechanical, electrical and software sub-systems. As these changes are carried out mostly undocumented, methods have been developed in the FYPA²C ("Forever Young Production Automation with Active Components") project (1st phase of the SPP) to observe runtime events in the aPS and thus to derive behavior models and to determine system properties, which can be stored and evaluated in a knowledge-carrying software. In the frame of the LinkedFYPA²C project (2nd phase of the SPP), the focus has been on the identification of evolution steps by analysis of similarities between behavior models and on exchanging evolution steps between similar aPS in a knowledge carrying network. Thus, the further evolution of an aPS is supported by providing model-based recommendations from its peers. A decentral form of peer-to-peer communication links the aPS and turns them into Cyber-Physical Production Systems (CPPS). In this network, evolution steps can be requested, exchanged, and evaluated between the CPPS. The concepts have been evaluated by means of the Pick-and-Place system.

MOCA

Specifying and Recognizing Model Changes in Co-Evolving Models (MOCA)

U. Kelter, G. Taentzer

Model-based software engineering has become a widespread paradigm for developing long-living software in various application domains. Being primary development artefacts, models are subjects to version management, notably comparison, patching and updating.

The main goal of this project was the lifting of version management services to a higher level of abstraction. Instead of specifying and recognizing model changes on the level of abstract syntax graphs (ASGs), changes are lifted to domain-specific edit operations as known from model editors.

The first phase of this project addressed the main theme for single, monolithic models. The second phase assumes that a system is modelled by several related models, i.e., a network of models, which represents different parts or stakeholder-specific views of the system. Developers often want to revise individual models of such a network independently. This requirement is not met by current approaches to model versioning. Established design styles for meta-models, as e.g. applied in the UML, aim at a redundancy-free, integrated modelling of all conceptual contents of all models (or views). As a result, a network of models becomes one integrated ASG and a monolithic unit of versioning.

The main goal of the second funding phase was to allow individual submodels or views to be independent units of versioning, i.e., to be checked out from a repository and later, after having been edited, to be checked in. From the developer's point of view all the submodel shall ideally appear as independent, co-evolving models. The solution strategy taken is to handle the check-out of a submodel as a specific model slicing task, and the later check-in - which can generally lead to conflicts and consistency violations - as a specific model repair task. These concepts and tools use the lifting to domain-specific edit operations developed in the first project phase.

To increase the usability of our tool support, a variety of meta-tools have been developed which support domain experts in configuring the services. The implemented tool environment has been and will be evaluated in case studies considering long-living systems in the fields of automation engineering and information systems.

Model-Driven Evolution Management for Microscopic Changes in Automation Systems (MoDEMMiCAS)

M. Broy, B. Vogel-Heuser

Automated Production Systems (aPS) constitute multi-disciplinary, software-intensive and long-living systems. System modifications to meet a constantly changing set of requirements (also called evolution) that regularly require long downtimes and critical test phases upon start-up. In the development of such systems, two different frequency scales of changes can be distinguished: the macroscopic and microscopic scales. Macroscopic scale evolutions occur in time frames of up to years (e.g. upgrade to new product generations), while evolutionary changes on a microscopic scale can happen as frequently as hourly (e.g., bug fix in source code). Enabling such a flexibility for system evolution is realized through a model-driven engineering methodology including respective engineering process models and formalisms to define and structure system requirements and to describe and analyze system architectures and component structures/behavior. In the first funding period of the priority program, the fundamental extensions to the modeling theory and model-checking techniques for verifying conformance of evolutionary changes in aPS were achieved. Based thereon, we further extended the modeling and verification framework in MoDEMMiCAS to also account for non-functional requirement of aPS (e.g., availability). Furthermore, we investigated the current state in aPS' development regarding the adaptation for fine-granular (microscopic) changes that fit a continuous integration (CI) and appropriate quality assurance approach for CI. In aPS software engineering mostly unit tests for general functions are rated as the means to achieve CI. Regression testing is promising approach to support CI for application specific functions as well as simulation of mechanical components in different intermediate modeling details before the final design is available. One of the key challenges is the best choice of the most appropriate level of complexity for CI, because to complex systems are not efficiently to simulate as they are customer specific and components are either already designed as mechatronic components and not challenging...

Techniques and Prediction Models for Sustainable Product-Line Engineering (Pythia)

S. Apel

Software product-line engineering has gained considerable momentum in recent years, both in industry and in academia. Companies and institutions (such as NASA and Hewlett Packard) apply product-line technology with great success to sustain their development by broadening their product portfolio, improving software quality, shortening time to market, and being able to react faster to market changes. However, contemporary prediction models for sustainable and economic development in software projects mostly ignore the structural and behavioural properties of the architecture and implementation assets of a product line. In the project PYTHIA, we propose to rethink contemporary prediction models by incorporation of product-line concepts and to employ a variety of state-of-the-art analysis techniques to create a rich prediction base relying on implementation specifics (including software metrics, static analysis, mining techniques, and employment of network analytics).

In more detail, we carried out three analysis types to gain insights into the structure and the evolution of software product-line projects: social analyses, technical analyses, and socio-technical analyses. We support each analysis with self-developed tools (i.e., Codeface, JDime, cppstats, and a generic library for network construction).
In our social analyses, we constructed developer networks based on the projects' means of communication and collaboration – as these are essential to any software project – to carry out network-analytic approaches: Identification of meaningful clusters of developers, measurement of the network structure and its evaluation, and observation of the networks over time to assess the projects' evolutionary trends in coordination.
In our technical analyses, we performed static measurements of feature scattering and tangling (including longitudinal studies on the Linux kernel) to understand product-line implementation patterns in more detail. Furthermore, to better understand and support the implementation workflow, we aimed at identifying indicators for merge conflicts in distributed version control systems (we found none!) and worked on the migration of existing software products into a product line.
In our socio-technical analyses, we consolidated data sources to examine specifically the fulfilment of coordination requirements on product-line implementation assets, i.e., on features: We extracted coordination requirements by mining concurrent changes to software implementation assets from the projects' history and aligned them with the projects' coordination to assess socio-technical congruence.

Overall, we gained helpful insights into the structure and evolution of software product lines (e.g., project coordination and feature implementation details) to support the sustainable and economic development of software product lines, which is also applicable for other software projects.

Beyond One-Shot Security: Requirements-driven Run-time Security Adaptation to Reduce Code Patching (SecVolution@Run-Time)

J. Jürjens, K. Schneider

During the first phase of the SecVolution project, formal and informal sources of information were used to identify security-related issues and automate the identification process through natural-language processing and ontologies. Mechanisms to mitigate known problems were investigated.

In the second phase, SecVolution@Run-Time continued using a socio-technical approach with diverse sources, but included phenomena that occur at run-time – which can partially be mitigated at run-time, too. For example, the degradation of service levels can help to reduce damage when a security breach has been identified. At the same time, ordinary customers can continue using the software, since an interruption would cause a substantial loss.

We developed a scheme of heuristics to identify suspicious effects during run-time monitoring, and investigated several instantiations of that scheme: FOCUS+ is a technique to extract specific expert knowledge for a given piece of code that may contain a security problem. It instantiates once more the principle of using both technical and human resources for mitigating a wicked security problem and prepares the ground for dealing with them at run-time, by supporting the decision whether it is necessary to revert back to design-time approaches (with the disruption that this will cause).
Using the framework for semi-automated co-evolution of security knowledge and system models that we developed, various approaches for performing security analysis can then be employed in this context, thus bridging the gap between security evolution approaches at design-time and run-time.