Comput Sci Res Dev (2015) 30:321–331
DOI 10.1007/s00450-014-0273-9

SPECIAL ISSUE PAPER

Design for future: managed software evolution

The DFG priority programme for long-living software systems

Ursula Goltz · Ralf H. Reussner · Michael Goedicke · Wilhelm Hasselbring · Lukas Märtin · Birgit Vogel-Heuser

Published online: 8 October 2014

Abstract Innovative software engineering methodologies, concepts and tools which focus on supporting the ongoing evolution of complex software, in particular regarding its continuous adaptation to changing functional and quality requirements as well as platforms over a long period are required. Supporting such a co-evolution of software systems along with their environment represents a very challenging undertaking, as it requires a combination or even integration of approaches and insights from different software engineering disciplines. To meet these challenges, the Priority Programme 1593 Design for Future—Managed Software Evolution has been established, funded by the German Research Foundation, to develop fundamental methodologies and a focused approach for long-living software systems, maintaining high quality and supporting evolution during the whole life cycle. The goal of the priority programme is integrated and focused research in software engineering to develop methods for the continuous evolution of software and software/hardware systems for making systems adaptable to changing requirements and environments. For evaluation, we focus on two specific application domains: information systems and production systems in automation engineering. In particular two joint case studies from these application domains promote close collaborations among the individual projects of the priority programme. We consider several research topics that are of common interest, for instance co-evolution of models and implementation code, of models and tests, and among various types of models. Another research topic of common interest are run-time models to automatically synchronise software systems with their abstract models through continuous system monitoring. Both concepts, co-evolution and run-time models contribute to our vision to which we refer to as knowledge carrying software. We consider this as a major need for a long life of such software systems.

Keywords Software life cycle · Design, maintenance and operation · Legacy systems · Co-evolution · Knowledge carrying software

We would like to thank Zoya Durdik, Gregor Engels, Christof Momm, Andreas Rausch, and Stefan Sauer for their contributions and all projects of the priority programme for providing details on their goals and collaborations.

U. Goltz · L. Märtin (✉)
TU Braunschweig, 38106 Braunschweig, Germany
e-mail: maertin@ips.cs.tu-bs.de

U. Goltza
e-mail: goltz@ips.cs.tu-bs.de

R. H. Reussner
Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany
e-mail: reussner@kit.edu

M. Goedicke
University of Duisburg-Essen, 45127 Essen, Germany
e-mail: michael.goedicke@paluno.uni-due.de

W. Hasselbring
Kiel University, 24098 Kiel, Germany
e-mail: hasselbring@email.uni-kiel.de

B. Vogel-Heuser
TU München, 85748 Garching bei München, Germany
e-mail: vogel-heuser@ais.mw.tum.de

1 Introduction: today’s challenges of the software industry

Software has become an enabling factor in many ground breaking developments across the board. In this role software is considered to provide a degree of flexibility which is breathtaking—especially for software developers. Amazon, for instance, releases a new version, release etc. of their services and products every eleven seconds on average [7]. If this phenomenon is an indicator of the general direction software development will look like in the future it shows good processes for this particular systems of this company. However, it is and will be a major challenge to sustain such a pace in general software development, even if change is less frequent.

This role of software makes it necessary to address the fast pace new emerging opportunities, changing customer needs, environmental changes just to name a few. The question is what are the challenges and how to address them. The effect of such a fast pace is usually that a software system shows quickly signs of ageing. Thus, software quality often decreases during the lifetime of long-living software systems in various aspects, e.g., conformance to user and system requirements, functionality, performance, reliability, and maintainability. This is ageing not by wear and tear but by accommodating the required changes quickly in order not be become obsolete by being too slow. As a result such changes to the software system are done in a hurry without considering the quality control and management, which was in place in the original development. Since decades, software engineering experts have been well aware of this phenomenon, called *software deterioration* [11,14]. Furthermore, the requirements software has to meet and the underlying hardware and infrastructure (e.g., execution platform) constantly change. If software does not continuously adapt, it will age relative to its environment. In the field of business information systems, we are very familiar with these problems, referred to by the term *legacy systems*. Software industry still has to cope with large-scale business applications based on opaque and monolithic FORTRAN or COBOL systems, which cannot be substituted due to their enormous complexity and—even worse—**lack of knowledge** on how they work. The same problem arises in the embedded software systems sector where software engineering has to deal with the problem of providing complex software for potentially long-living technical devices and systems.

In the wake of these problems an initiative called *DevOps* emerged. In DevOps development and operations are working together in order to avoid too much separation between the two departments. However, this is barely a wish and a high level management strategy. It remains to be seen what a systematic approach has to be. We address this in our research programme by providing appropriate detailed models and processes. Indeed, several areas of software engineering need to be considered and their respective contributions have to be evaluated. Furthermore, huge investments need to be made in retaining software quality for *long-living software systems*. The following three problem areas in developing long-living software systems are of particular interest:

– A lack of understanding of the mutual dependencies between functional and quality requirements, of the internal structure of software systems, and dependencies on other hardware and software system components prevents the efficient and effective evolution of software systems. This causes major shortcomings in the productivity of the users of a software system and finally leads to acceptance problems.
– There are various methods and techniques for specific aspects of software evolution available. However, a consistent and comprehensive methodology which covers development, operation, and quality assurance at the same time does not exist. This, of course, includes and addresses model based approaches in particular.
– The complexity and timing of the development of applications on the one hand and the development of platforms and technologies on the other hand, frequently interfere each other and prevent a focused and continuous development.

To address these challenges, in August 2012 the German Research Foundation (DFG) launched the Priority Programme 1593 (SPP1593) to encourage the German software engineering community to develop approaches for evolution in software and software/hardware systems.

Section 2 introduces the evolutionary life cycle of software systems that includes both design and construction phases as well as operation phases and involves different disciplines, before Sect. 3 presents our vision of managed software evolution. The joint case studies are essential to promote close collaborations among the individual projects, see Sect. 4. The priority programme projects, their collaboration and a clustering are presented in Sect. 5, before Sect. 6 concludes the paper.

2 Evolutionary software life cycle

Traditional release cycles, where software is updated as a whole, vanish. Nowadays, software relies on several independent or loosely coupled components using complex technology stacks comprising hardware, middleware and reusable software components and other (software) systems. Regarding software evolution, we have to find ways to deal with a co-evolution of these different parts taking place at different release cycles. The challenges in integrating the operation and development at different levels are depicted in Fig. 1 for the context of a technical system in automation engineering.

The *evolutionary life cycle* of the system includes both design and construction phases as well as operation phases and involves various disciplines. The life cycle starts with the original design and construction of infrastructure and soft-

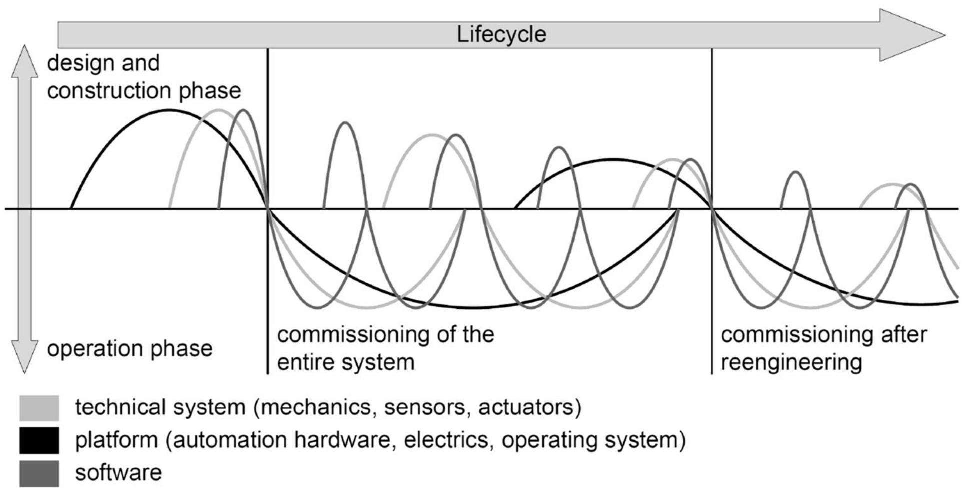
Lifecycle
design and construction phase
operation phase
commissioning of the entire system
commissioning after reengineering
technical system (mechanics, sensors, actuators)
platform (automation hardware, electrics, operating system)
software

Fig. 1 Integration of development and operation of hardware/software systems [12]

ware. Before operation starts, a commissioning of the entire system (i.e., a consolidation of all disciplines) is necessary. During operation mechanical construction phases take considerably more time than designing the platform and software, especially in area of production automation systems. Often technical systems or platforms are running for many years, resulting in a smaller change frequency than in software engineering. As changes, adaptations and updates may affect necessary parts of the system, the system may have to be shut down in order to commission the entire re-engineered system. In automation of production systems, it is usually an important requirement to keep these downtime phases extremely short. In Fig. 1 the various areas (software, platform and technical system) are depicted in different grey scales and the height of each curve indicates the amount of effort required in these areas. As one can see each of these areas have their own dynamics and the challenge is to manage the effort in such a way that the entire system will develop and operate in a meaningful manner.

In fact, we need a tight integration of construction/design, deployment and operation phases. As sketched in Fig. 1 this means we need for new compositional software life cycle management approaches, which reflect the complex technology stack of modern service-oriented systems, production automation systems etc. In addition, further integration of design-time and run-time information about the software is required. For instance, during system operation and evolution a great deal of design-time information about the operated software is missing. On the one hand, development artefacts are not kept up-to-date after the design phases and, on the other hand, relations between run-time management information and design-time artefacts are barely considered. As a consequence, development processes should take the operation and evolution of the software systems explicitly into account [1].

3 Managed software evolution

In the Priority Programme 1593 the continuous co-evolution of software and software/hardware systems is addressed by making systems adaptable to changing requirements and environments. Several fields in the discipline of software engineering contribute to this vision and may be focused and integrated to reach the goal of a new development methodology for long-living software systems, maintaining high quality and supporting evolution during the life cycle. However, these approaches need to be further developed and integrated with a special focus on long-living software systems. For this, we propose the new paradigm where (1) development, adaptation and evolution of software and their platforms as well as (2) operation, monitoring and maintenance are no longer separated but integrated. The goal is to define meta-models for preserving and accessing the knowledge provided and gained during the system development process. Furthermore, methods and process models, as well as suitable infrastructures, will be provided to comprehensively support the integration of software development and evolution.

Three guiding themes have been proposed as research structure for the priority programme and its projects:

Guiding Theme I: Knowledge Carrying Software. The knowledge contained in software or underlying its design needs to be integrated and made accessible in the resulting artefacts, both for functional and for quality requirements. Appropriate meta-models with specific support for continuous evolution of software and software/hardware systems need to be developed, in particular—but not necessarily exclusively—for advancing model-based and model-driven software engineering.

Guiding Theme II: Methods and Processes. Design and evolution of software need to be supported by methods ensuring

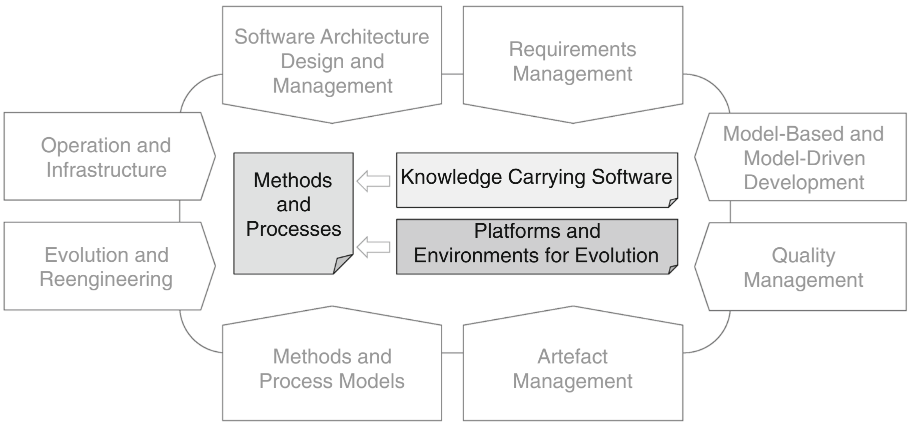
Software Architecture Design and Management
Requirements Management
Operation and Infrastructure
Methods and Processes
Knowledge Carrying Software
Model-Based and Model-Driven Development
Platforms and Environments for Evolution
Evolution and Reengineering
Quality Management
Methods and Process Models
Artefact Management

Fig. 2 Guiding themes of the priority programme related to current research in software engineering

that knowledge is preserved and integrated. A new model for the life cycle of software or software/hardware systems needs to be developed, allowing and considering differing evolution cycles on different levels of the software/platform/hardware stacks.

Guiding Theme III: Platforms and Environments for Evolution. Infrastructure for the evolution of software or software/hardware systems in terms of suitable middleware and robust run-time environments, for monitoring and changing during operation, needs to be provided. Design- and run-time information needs to be made accessible wherever needed during the operation of systems.

Figure 2 summarises the relevant fields of today's software engineering research and shows how the overall structure of the priority programme relates to these fields of software engineering.

The priority programme is designed for two periods of funding, each with a duration of three years. Figure 3 shows the orientation both funding periods. In the first period, we are focused on guiding themes I and II to lay the foundations by defining new concepts for software system evolution. In the second period we will have a stronger focus on evaluation and appropriate infrastructures w.r.t. guiding theme III.

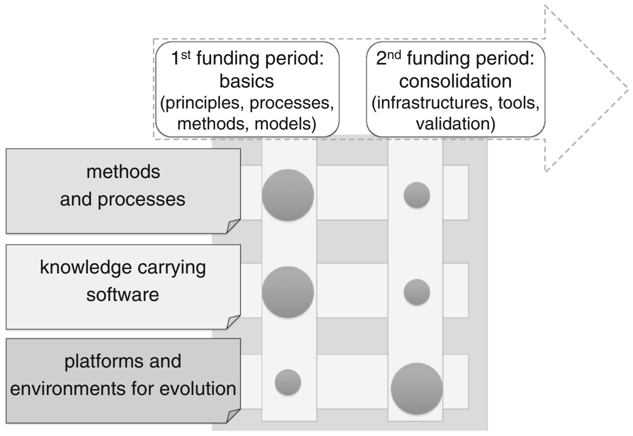
1st funding period: basics (principles, processes, methods, models)
2nd funding period: consolidation (infrastructures, tools, validation)
methods and processes
knowledge carrying software
platforms and environments for evolution

Fig. 3 Focuses in funding periods of the priority programme

4 Case studies

Several of the projects in the priority programme develop methods and approaches tailored to the domain of production systems in automation engineering. They evaluate their results using the *Pick & Place Unit* [8], a lab-size demonstrator at the Institute of Automation and Information Systems[1] at the Technische Universität München. Another group of projects concentrates on information systems and embedded systems. For those projects, we provide a joint case study based on *CoCoME* [15] in terms of a cash desk with an associated stock management for supermarkets, supervised by the Chair for Software Design and Quality[2] at the Karlsruher Institute for Technology. This case study has already been deployed as an international benchmark for component-based modelling approaches, but is now adapted to the goals of the priority programme.

Sections 4.1 and 4.2 introduce the two case studies, respectively. Later, Sect. 5 will present how the individual projects work and collaborate on these case studies.

4.1 Pick and place unit

The engineering of automation software for manufacturing systems differs from traditional software design for embedded systems. Run-time environments, implementable in the programming languages standardised in IEC 61131-3[3] (e.g., Sequential Function Charts), abstract from several techni-

[1] http://www.ais.mw.tum.de/en.

[2] http://sdq.ipd.kit.edu.

[3] Open international standard for programmable logic controllers, http://www.iec.ch.

cal details of the platform. However, a dependency between automation software and physical hardware, i.e., context and platform, still exists. The complexity of manufacturing systems with typically thousands of input and output signals of sensors and actuators used within one plant is nonetheless high. Furthermore, reuse is handicapped by several aspects: Nearly each plant is customised individually to technical requirements, local standards, local qualification of maintenance personnel and requested components.

In this context, model-driven development of automation software for manufacturing systems grows in importance. First tool prototypes are available using object-oriented mechanisms and integrating the UML into traditional IEC 61131-3 environments [17] being the programming and development standard for manufacturing systems since years. Furthermore, service-oriented approaches are well applied in manufacturing systems on the level of sensor and actuator integration, the so-called device description languages [2,3] as well as on the level of dynamic adaptation during run-time using agent-based approaches to replace sensors by a soft sensor [16] or adjust control behaviour [4] in case of failures. Nevertheless, a major drawback still exists: unstructured legacy code from existing plants and mechatronic libraries is frequently reused. Therefore an approach is required to analyse legacy code and develop migration concepts to model-driven development or service-orientation applicable for new plants. Evolution of automation software of manufacturing systems is not always applicable in a model-driven way as a planned change. At plant site, in case of downtime phases, software code is frequently adapted unplanned under high time pressure with limited access to all engineering data and limited knowledge. Both of these challenges are addressed by projects of the priority programme using the manufacturing automation demonstrator. Product line approaches and delta engineering seem to be promising ways to support evolution of manufacturing system software, but requires changes to the standardised and established programming languages as well as adaptations in engineering and run-time platforms.

To evaluate the applicability of their approaches regarding automation software for manufacturing systems, nine projects use the *Pick & Place Unit* (PPU) [8] as exemplary lab-size demonstrator for manufacturing systems (see Fig. 4). The evolution of the mechanical, electrical and software parts of the PPU over the last ten years is structured into 16 scenarios. Besides functionally driven evolution, i.e., processing of other types of work pieces (colour, material), also quality requirements [9], e.g., precision, throughput, and dependability, lead to these evolution steps. A detailed description of the PPU's evolution scenarios was elaborated for the priority programme [10]. The final configuration of the PPU consists of four modules: a stamp, a stack, a crane, and a sorter. After separating work pieces (cylindrical chumps) at the stack with

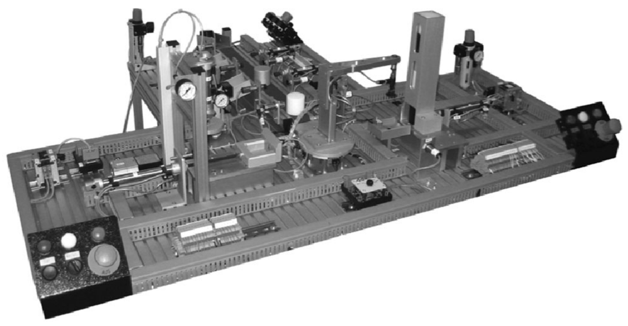

Fig. 4 PPU lab-size demonstrator [8]

a pneumatic cylinder, chumps can be transported by the crane either to the stamp for further processing or directly to the sorter. For stamping material with different pressure profiles, the stamp is also equipped with a pneumatic cylinder. A variety of sensors installed at the sorter enables the identification of the type of the work piece (colour, material) and ejecting them correctly by corresponding ramps using the pneumatic cylinders.

As an interface to the projects, a full documentation for each of the scenarios in SysML is available as well as corresponding IEC 61131-3 programs. For evaluation purposes, a MATLAB/Simulink model of all evolution steps is provided which offers the opportunity to run the program code on a PC-based logic controller (PLC) in loop with the simulation. Furthermore, the customised control software of the project can be evaluated in loop with the simulation as well as connected to the PPU via OPC [13], a standardised interface in automation engineering.

4.2 Common component modelling example

The case study *Common Component Modelling Example* (CoCoME) describes a fictitious business information management system, originally developed and internationally used as a benchmark system for several component-based modelling approaches [6]. CoCoME is an open source system with a comprehensive documentation including a detailed requirement specification, architectural models in several modelling languages and two runnable implementation versions: a component-based implementation.[4] and a SOA-based variant.[5]

The context of CoCoME is a trading system in a supermarket or a similar retail unit. System structure of CoCoMe is illustrated in Fig. 5.

It consists of cash desks, store and enterprise. A cash desk has several devices connected to it, such as a bar code scanner or a receipt printer. Multiple cash desks are connected to

[4] http://www.sourceforge.net/projects/cocome/develop.

[5] http://www.sourceforge.net/apps/trac/sla-at-soi.

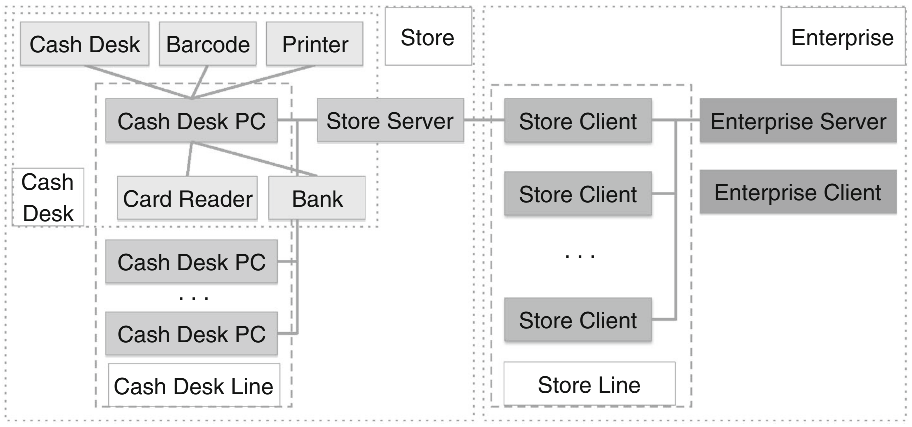

Fig. 5 CoCoME system overview

the CoCoME store server. These devices belong to the store environment. Moreover, store servers are connected to the enterprise server via store clients, comprising the enterprise CoCoME environment. Cash desk applications, store server applications, store clients, enterprise applications and enterprise clients can be installed on independent nodes and also can be replicated. The communication runs via RMI in the traditional implementation of CoCoME, and via WS calls in the SOA-based variant. The SOA-based implementation is also deployable in the cloud.

The usage scenario of the CoCoME system reflects customers buying goods in a supermarket. Items the customer wants to buy are scanned at the cash desk and can be paid either by cash or with a credit card. At the end of a transaction, the customer receives a receipt. If the supply of certain goods is coming to an end, the inquiry about resupply is sent via the store client to the enterprise server. There it is processed, and items are shipped to the supermarket.

The current variants of CoCoME are not intended to support evaluation of methods targeting evolution. Hence, we propose the following evolution scenarios:

– Exchange of the AMPL [5] optimisation library
– Error-resistance and performance adaptation
– Introduction of an event channel into the system
– Exchange of communication protocol
– Home delivery service, payback and discount service offers
– Implementation of a web shop support
– CoCoMe as a product line
– Platform or infrastructure change
– Migration to cloud infrastructures
– Model-driven development of CoCoME

The most relevant of these scenarios will be elaborated and implemented for evaluation in the priority programme. The CoCoME case study is relevant for seven projects of the priority programme.

5 Projects of the priority programme

The priority programme comprises thirteen scientific projects with 25 principal investigators from software and automation engineering. We present the individual project goals, project collaborations and project clustering in the following Sects. 5.1, 5.2 and 5.3.

5.1 Individual project goals

ADVERT[6] addresses the challenge of capturing and evolving design decisions to keep development artefacts synchronised (requirements, documentation, architecture specification, and code). A vetical tool prototype for synchronisation of architectural design and implementation artefacts is available. For evaluation, an empirical study on CoCoME and some real-world case studies from former industry and research projects are planned.

DAPS[7] develops new high-level models for performance specification and modelling of variability and evolution. The work is motivated by the increasing complexity of software in long-living systems, and the resulting number of variants over time. New techniques for efficient performance analysis both at design-time and at run-time are engineered, and evaluated on the PPU case study.

ENSURE[8] investigates the co-evolution of quality of service (QoS) models and architectural models. Particularly, the certification of QoS properties of evolving systems, including run-time certification, is introduced and evaluated via model-based safety and reliability evaluations of the evolving PPU (consistent models, model transformations, coevolution study). The model-based performance evaluation of CoCoME is envisioned.

[6] http://www.dfg-spp1593.de/advert.
[7] http://www.dfg-spp1593.de/daps.
[8] http://www.dfg-spp1593.de/ensure.

EvoLine[9] supports consistency checking between variability models and artefacts during evolution. Employing the Linux kernel and kconfig files as case studies, EvoLine identifies modifications in a product line that lead to problems in single products.

FYPA2C[10] develops methods and processes for realising knowledge carrying software for evolving production systems, especially in the context of undocumented changes. This includes the automatic generation and scenario-based evaluation of knowledge models and test cases for the PPU case study on the basis of occurring low-level signal traces within an active component architecture. A semi-automated requirements verification process based on anomaly detection of changes as well as test cases and common usage scenarios has already been developed. Besides the PPU case study, a local laboratory manufacturing plant evolution scenario is under construction and a simulated benchmark plant is planned.

IMoTEP[11] investigates the co-evolution of software product line (SPL) models and test artefacts, particularly the propagation of changes in SPL models into updates of model-based SPL test artifacts. Via a case study of its associated partner Eckelmann AG and the PPU case study, IMoTEP evaluates quality assurance for (re-)configuration at run-time based on dynamic SPL testing strategies as well as for unforeseen evolution based on regression testing.

IMPROVE[12] leverages advances in deductive program verification for regression verification (formally proving that software behaviour is sustained through its evolution). IMPROVE develops powerful regression verification methods and tools for regression verification for Java and intends to evaluate these on a collection of micro-benchmarks from literature and on the CoCoMe case study.

iObserve[13] investigates model-based observation, prediction, forecasting and analysis of performance and privacy in dynamic cloud contexts. In particular, new methods and techniques for model-driven instrumentation, creation and update of run-time models for verification of geo-location-constraints are addressed. The techniques are evaluated on the CoCoMe case study, particularly with the scenario of moving a database from one cloud provider to another while complying with data policy requirements.

MOCA[14] develops a methodology and tools for specifying and recognising changes in versions of models. The key idea is to integrate model-driven key technologies for model transformation and model versioning and to evaluate this with managing and planing the evolution of SysML models of the PPU case study.

MoDEMAS[15] investigates evolution challenges in automation engineering in practice. The MoDEMAS project employs simulation and model checking to support evolution of automation systems. The approach is evaluated via simulation of typical evolution scenarios on the PPU case study. The goal is to identify and combine those viewpoints, which are required to model an automation system holistically.

Pythia[16] utilises various sources of information obtained via product line analysis to improve the predictive power of state-of-the-art prediction models for product line engineering. A product line dashboard that aggregates, integrates, and visualises organisational, architectural, and implementation-related information is developed to support decisions such as whether to add a feature, remove a feature, or alter the organisation, architecture, or implementation (refactoring). Pythia is evaluated based on several open-source software systems, several case studies from partners at Fraunhofer IESE, Keba, and Siemens, as well as the source code of the PPU case study.

SecVolution[17] addresses the systematic co-evolution of knowledge and software for security. Hereby, SecVolution develops techniques, tools, and processes that support security requirements and design analysis techniques for evolving, long-living systems in order to ensure *lifelong* compliance to security requirements. The evaluation is planned both on the CoCoME case study with the evolution scenario of migrating the shop to the cloud and the PPU case study for investigating relevant security vulnerabilities.

URES[18] has the goal to support the documentation and usage of decision knowledge during software evolution. URES investigates the structures to represent decisions, the resulting effort for developers and how to exploit decision knowledge in order to support the system evolution. End user behaviour is reflected in evolution decisions to identify software improvements through mismatches between expected and observed user behaviour. These techniques are evaluated on the UNICASE tool and both the CoCoME and the PPU case studies.

9 http://www.dfg-spp1593.de/evoline.

10 http://www.dfg-spp1593.de/fypa2c.

11 http://www.dfg-spp1593.de/imotep.

12 http://www.dfg-spp1593.de/improve.

13 http://www.dfg-spp1593.de/iobserve.

14 http://www.dfg-spp1593.de/moca.

15 http://www.dfg-spp1593.de/modemas.

16 http://www.dfg-spp1593.de/pythia.

17 http://www.dfg-spp1593.de/secvolution.

18 http://www.dfg-spp1593.de/ures.

5.2 Project collaboration

In order to achieve the intended integration between partners of different fields of software engineering, and hence to reach a focused approach towards a coherent methodology, considerable efforts are necessary. Frequent information exchange and discussion of research results in early stages are absolutely essential. This is achieved on the one hand by organising frequent meetings in suitable forms, not only for information exchange within the priority programme but also with national and international experts. On the other hand, we build networks between related projects, both guided by the joint case studies and by establishing working groups for related research interests.

ADVERT and iObserve collaborate on the development of CoCoME evolution scenarios, the integration of a monitoring probe definition language into synchronised development artefacts to gather and represent run-time deployment information. ADVERT and SecVolution collaborate on capturing relevant security knowledge from architectures and synchronising that knowledge between model and code. DAPS collaborates with FYPA2C on measurements of the PPU. ENSURE collaborates with MOCA on the PPU system architecture and fault tree models, with FYPA2C on probabilistic QoS models, and with iObserve on learning performance attributes based on the CoCoME implementation for model update via dynamic analysis. FYPA2C collaborates with MoDEMAS on run-time verification of production systems on basis of PLC signal traces, with DAPS on the PPU regarding performance verification is planned and with ENSURE on probabilistic models. IMoTEP collaborates with Pythia on SPL test case generation based on symbolic model checking, with MoDEMAS on feature modelling in the automation engineering domain, with MOCA on consistency-preserving evolution of SPL test artefacts by model differencing, and with DAPS on variability modelling in SPL engineering. iObserve collaborates with ADVERT on run-time and data modelling and with ENSURE, SecVolution, URES, and MOCA on model-driven code generation for the CoCoME case study. MOCA collaborates with MoDeMAS on the comparison of SysML models of the PPU, with IMoTEP on the comparison of feature models, with SecVolution on the change recognition of changes in security maintenance models, and with ENSURE on the comparison of system architectures and fault tree models. Pythia collaborates with DAPS on variability-aware performance analysis, with IMoTEP on variability-aware test-case generation and execution and with MOCA on structural differencing and merging to compare tools. URES collaborates with SecVolution on integrating heuristic analyses for requirements with the decision editor in UNICASE as well as with iObserve, ADVERT, and SecVolution on the development of CoCoME evolution scenarios.

In particular the shared case studies promote close collaborations among the individual projects. We observe several research topics that are of common interest, for instance:

Co-evolution of models and implementation code, of models and tests, and among various types of models.
Run-time models to automatically synchronise software systems with their abstract models through continuous system monitoring.

Both, co-evolution and run-time models contribute to our vision of knowledge carrying software which is a requirement for a long life of these software systems.

5.3 Project clustering

One of the goals in the first funding period of the priority programme was to evenly cover our guiding themes I and II as well as our application domains. The assignment of projects is depicted in Fig. 6. We find that four projects concentrate solely on information systems, five projects solely on automation engineering. Four projects are working in both domains. The width of each block in the figure illustrates the intensity (one-, two- or three-thirds) of the assignment to a guiding theme. In accordance with the focus in the first funding period, the majority of the projects contribute to guiding theme I and II.

Besides these classifications, we identify seven important themes from software engineering which are covered by our projects, illustrated in Fig. 6.

In the projects concerned with Software Product Lines (*SPLs*), performance and testability of variant-rich software systems are examined. The projects in *Verification & Validation* develop sophisticated software approaches for quality assurance in practical application. Also formal methods are part of this field. *Meta Modelling* covers approaches with strong orientation to model-based/model-driven development and analysis of systems on high level of abstraction. *Change Analysis* is strongly related to the preservation of knowledge in maintenance phases. Close to that, *Monitoring* concepts support the integration of human and environmental aspects into the development process. Highly flexible system and software structures are investigated in the field of *Architectures*. Finally, the field *Non-functional Properties* focuses on system properties on the level of qualitative ratings beyond functional requirements.

6 Conclusions

In the Priority Programme 1593, we have the ambitious goal to provide methods and tools for developing "forever-young software". This leads to software whose expenses in main-

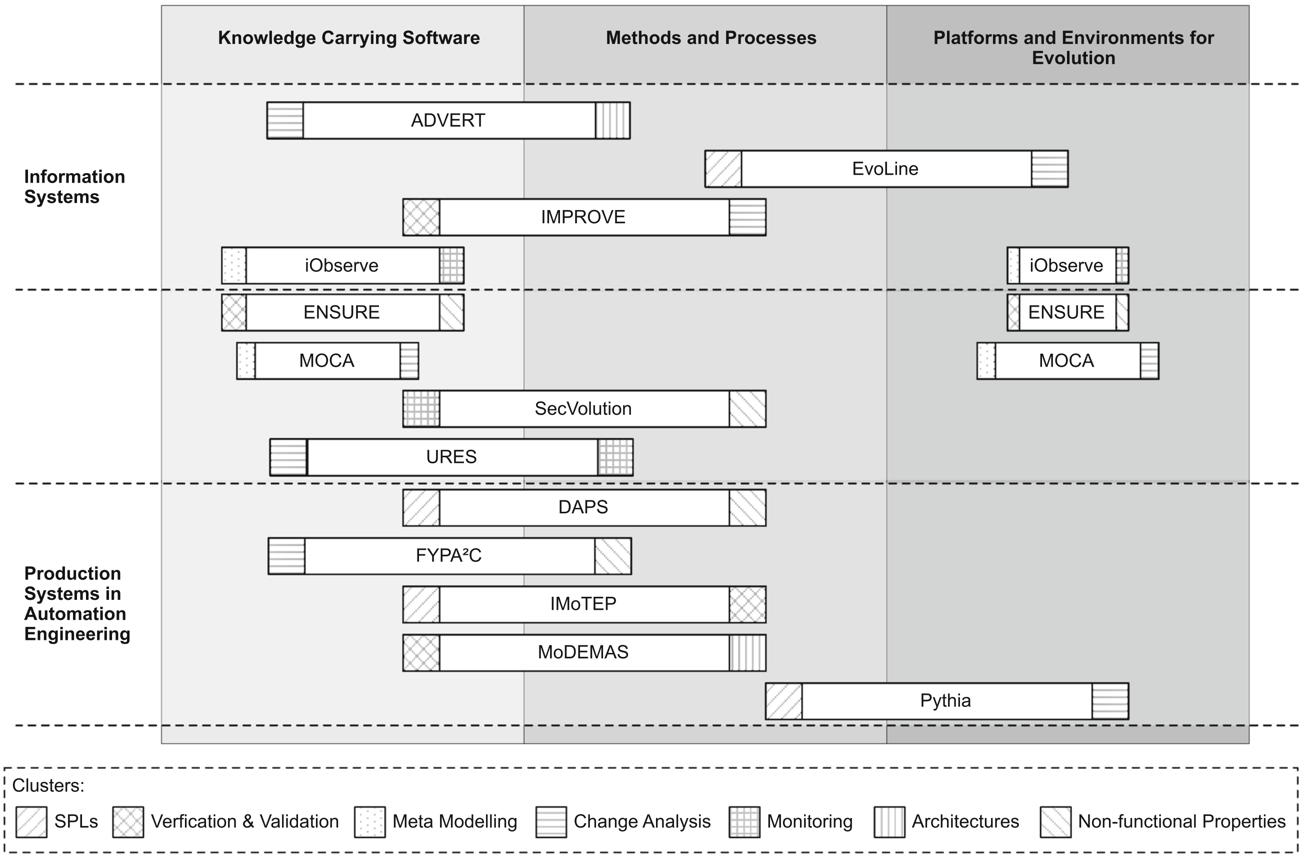

Fig. 6 Clusterisation along the research structure of the priority programme

tenance are considerably lower than the maintenance costs of current legacy systems and which will even improve in quality over its life cycle. Thus, improvements regarding all software development phases and software artefacts on the various levels of abstraction in an evolutionary software life cycle are required. Several research fields in software engineering need to be combined for the development of integrated approaches to reach our goals.

We proposed a research structure of three guiding themes to integrate all activities in development, operation, monitoring and maintenance. Two of these guiding themes reflect that gathering, preservation and usage of knowledge during development and operation, and the integration into methods and processes, are key issues. In our third guiding theme, adequate infrastructures for evolutionary platforms as well as process models are investigated. Furthermore, several fields from current research in software engineering contribute to our goals. We have shown that we have an excellent representation of these fields in the priority programme, as well as a good coverage of our guiding themes.

In order to stay close to application, we have identified two particular important application domains: production systems in automation engineering and information systems. For this purpose, two joint case studies for these domains are broadly accepted in the priority programme, and promising scenarios for evolution were already determined.

If this coordinated effort of software engineering research will help to save only a fraction of the huge expenses spent nowadays on the maintenance of long-living software systems, the situation might be considerably improved. However, we expect much more. We intend to provide the methods, processes and infrastructures to not only preserve, but even to improve the quality of long-living software systems over their whole life cycle.

Acknowledgments This work was partially supported by the DFG (German Research Foundation) under the DFG Priority Programme 1593: Design For Future—Managed Software Evolution (SPP1593).

References

1. Bencomo N, Blair G, France R, Muñoz F, Jeanneret C (2009) Proceedings of the 3rd international workshop on Models@run.time. In: Models in software engineering. Springer, Berlin, pp 90–96

2. Diedrich C (2006) Integrating technologies of field devices in distributed control and engineering systems. In: Integration technologies for industrial automated systems, chap. 11. CRC Press, Boca Raton
3. Evertz L, Epple U (2013) Laying a basis for service systems in process control. In: Proceedings of the 18th international conference on emerging technologies & factory automation
4. Feldmann S, Loskyll M, Rösch S, Schlick J, Zühlke D, Vogel-Heuser B (2013) Increasing agility in engineering and runtime of automated manufacturing systems. In: Proceedings of 14th international conference on industrial technology
5. Fourer R, Gay D, Kernighan B (2002) AMPL: a modeling language for mathematical programming. Duxbury, USA
6. GI-Dagstuhl Research Seminar: Modelling Contest: Common Component Modelling Example (CoCoME). http://www.cocome.org
7. Information Week Magazine: Amazon's Vogels Challenges IT: Rethink App Dev (Interview). http://www.informationweek.com/d/d-id/1107599
8. Institute for Automation and Information Systems: The pick and place unit-demonstrator for evolution in industrial plant automation (2014). http://www.ppu-demonstrator.org/
9. Ladiges J, Haubeck C, Wior I, Arroyo E, Fay A, Lamersdorf W (2013) Evolution of production facilities and its impact on non-functional requirements. In: Proceedings of the 11th international conference on industrial informatics
10. Legat C, Folmer J, Vogel-Heuser B (2013) Evolution in industrial plant automation: a case study. In: Proceedings of the 39th international conference of the IEEE Industrial Electronics Society
11. Lehman M (1980) Programs, life cycles, and laws of software evolution. Proc IEEE 68(9):1060–1076
12. Li F, Bayrak G, Kernschmidt K, Vogel-Heuser B (2012) Specification of the requirements to support information technology-cycles in the machine and plant manufacturing industry. In: Proceedings of the 14th IFAC symposium on information control problems in manufacturing, vol 14, pp 1077–1082
13. OPC Foundation: OPC Data Access Custom Interface Standard Version 3.00 (2003). https://opcfoundation.org/developer-tools/specifications-classic/data-access/
14. Parnas DL (1994) Software aging. In: Proceedings of the 16th international conference on software engineering. IEEE Computer Society Press, New York , pp 279–287
15. Rausch A, Reussner R, Mirandola R, Plasil F (eds) (2008) The common component modeling example: comparing software component models [result from the Dagstuhl research seminar for CoCoME, August 1–3, 2007]. Lecture Notes in Computer Science, vol 5153. Springer, Berlin
16. Schütz D, Wannagat A, Legat C, Vogel-Heuser B (2013) Development of PLC-based software for increasing the dependability of production automation systems. IEEE Trans Ind Inform 9(4): 2397–2406. doi:10.1109/TII.2012.2229285
17. Witsch D, Vogel-Heuser B (2011) PLC-Statecharts: an approach to integrate UML-Statecharts in open-loop control engineering—aspects on behavioral semantics and model-checking. In: Proceedings of the 18th IFAC World Congress

Ursula Goltz studied Computer Science at the Technical University of Aachen and graduated there with a diploma degree 1982. She received their Ph.D. degree from the TU Aachen in the year 1988. She worked as a scientific assistant at the TU Aachen from 1982–1985 and at the Institute on Methodological Foundations of GMD, St. Augustin from 1986-1992. After teaching activities at the Universities of Munich, Erlangen-Nürnberg, Mannheim and Bonn, she became professor for Programming at the University of Hildesheim in the year 1992. Since 1998 she is professor for Computer Science at the Technical University of Braunschweig. There she is chair of the Institute for Programming and Reactive Systems. Her main research interests are specification and system design, reactive systems, concurrency, process algebras and semantics.

Ralf H. Reussner holds the Chair for Software-Design and -Quality at the KIT (formerly University of Karlsruhe) since 2006. His research group works in the area of component based software design, software architecture and predictable software quality. In addition, he acts as a PC member or reviewer of several conferences and journals, including IEEE TSE and IEEE Computer. As Director and Scientific Executive of Software Engineering at the IT Research Centre in Karlsruhe (FZI) he consults various industrial partners in the areas of component based software, architectures and software quality. Since 2011 he is member of the executive board of the FZI and its speaker since 2013. He is principal investigator or chief coordinator in several grants from industrial and governmental funding agencies. He graduated from University of Karlsruhe with a PhD in 2001. After this, Ralf Reussner was a Senior Research Scientist and project-leader at the Distributed Systems Technology Centre (DSTC Pty Ltd), Melbourne, Australia. From March 2003 till January held the Juniorprofessorship for Software Engineering at the University of Oldenburg, Germany, and was awarded with a grant of the Emmy-Noether young investigators excellence programme of the National German Science Foundation.

Michael Goedicke studied Computer Science at the Technical University of Dortmund and graduated there with a diploma degree in 1980. He received his Ph.D. degree from the TU Dortmund in the year 1985 and his Habilitation in 1993 there as well. He worked as a scientific assistant at the TU Dortmund from 1980-1989 and at the University of Duisburg-Essen from 1990. He was a research fellow at Imperial College in 1989 and became Professor for Specification of Software Systems at the University of Duisburg-Essen (then University of Essen) in 1994. In 2010 he was co-founder of paluno The Ruhr Institute for Software Technology and serves as a Vice Director since then. His main research interests are specification and system design, semantics, software architecture and recently eAssessment for programming tasks.

Wilhelm Hasselbring is a full professor of software engineering at Kiel University, Germany, and chair of the Kiel/Lübeck KoSSE competence cluster on software systems engineering. His research interests include software engineering and distributed systems. He received his Ph.D. in computer science from the University of Dortmund in 1994 and his diploma degree in computer science from the Technical University of Braunschweig in 1989. Before moving to Kiel in 2008, he was full professor at the University of Oldenburg, assistant professor at the University of Tilburg (NL), postdoc at the University of Dortmund, and Ph.D. student at the University of Essen. He's a member of the ACM, the IEEE Computer Society, and the German Association for Computer Science.

Lukas Märtin studied Business Informatics at TU Braunschweig in Germany. Since he graduated with diploma degree in May 2010, he is doing his Ph.D. as research assistant at the Institute for Programming and Reactive Systems at the same university. His main researches focus on quality-oriented reconfiguration mechanisms for fault-tolerant software architectures of cyber-physical systems in space domain. He is the managing director of the DFG Priority Programme 1593 and responsible for supporting the collaborative work between 13 scientific sub-projects. In this capacity, he organized four workshops and one spring school over the last two years, each venue with about 50 participants.

Birgit Vogel-Heuser graduated in electrical engineering and received the Ph.D. in mechanical engineering from the RWTH Aachen in 1991. She worked for nearly ten years in industrial automation in the machine and plant manufacturing industry. After holding different chairs of automation she has been head of the Institute of Automation and Information Systems at the Technische Universität München since 2009. Her research work is focused on modeling and education in automation engineering for distributed and intelligent systems.