A Platform for Empirical Research on Information System Evolution*

Robert Heinrich[1], Stefan Gärtner[2], Tom-Michael Hesse[3], Thomas Ruhroth[4], Ralf Reussner[1], Kurt Schneider[2], Barbara Paech[3], and Jan Jürjens[4]

[1]Karlsruhe Institute of Technology, Germany, {heinrich, reussner}@kit.edu
[2]Leibniz Universität Hannover, Germany, {stefan.gaertner, kurt.schneider}@inf.uni-hannover.de
[3]University of Heidelberg, Germany, {hesse, paech}@informatik.uni-heidelberg.de
[4]TU Dortmund, Germany, {thomas.ruhroth, jan.jurjens}@cs.tu-dortmund.de

Abstract

Software-intensive systems are subject to continuous change due to modification of the systems themselves and their environment. Methods for supporting evolution are a competitive edge in software engineering as software is operated over decades. Empirical research is useful to validate the effectiveness of these methods. However, empirical studies on software evolution are rarely comprehensive and hardly replicable. Collaboration in empirical studies may prevent these shortcomings. We analyzed the support for such collaboration and examined existing studies in a literature review. Based on our findings, we designed CoCoMEP– a platform for supporting collaboration in empirical research on software evolution by shared knowledge. We report lessons learned from the application of the platform in a large research programme.

1 Introduction

In industrial practice, many information systems [1] are operated over decades. During operation they face various modifications, e.g. due to emerging requirements, bug fixes, and environmental changes, such as legal constraint or technology stack updates. In consequence, the systems change continually which is named software evolution [2]. Supporting software evolution is a competitive advantage in software engineering. A variety of methods aim at supporting different aspects of software evolution. However, it is hard to assess their effectiveness and to compare them due to divergent characteristics. Empirical research in terms of case studies and controlled experiments is useful to validate these methods. However, empirical studies on software evolution are rarely comprehensive. They often cover only one of the many aspects needed to study evolution: (i) long time-frames of observation are required to analyze changes, (ii) large amount of artifacts and (iii) various types of artifacts are affected by evolution, (iv) artifacts repeatedly change, (v) changes partly build upon each other, (vi) various stakeholders are involved, (vii) access to relevant project data, (viii) relevant project data must be documented over long time spans, (ix) relevant context knowledge must be documented beyond the code base and issue trackers.

To study evolution comprehensively, we believe it is important to collaborate by joint research in order to increase coverage of the aspects. Joint research supports sharing of knowledge and resources [3]. In particular, this allows replicating studies which in general is important to confirm and to strengthen results of empirical research [4] and thus enhance evidence. Our goal is to support joint research by collaboration and replication in empirical studies based on common evolution scenarios and artifacts. Currently, empirical studies on software evolution are seldom comparable as they vary in analyzed subjects and execution process. Furthermore, these studies are rarely reusable as important artifacts (e.g., requirements, design decisions, or context knowledge) are often not provided to the community. To the best of our knowledge, there is neither a community-accepted case study for software evolution nor a common benchmark available. Consequently, a common basis for study collaboration and replication is missing.

In this paper, we propose CoCoMEP[1] – a platform for collaborative empirical research on information system evolution. Under a "platform" we understand a comprehensive knowledge base for the evaluation process that can be exploited and extended by other researchers with different backgrounds and research interests. It provides assistance on diverse characteristics important for software evolution, e.g. the life-cycle of the system, artifacts in different revisions, and comprehensive evolution scenarios.

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

[1]The term is a combination of Common Component Modeling Example "CoCoME " [5] and "Platform"

DOI reference number: 10.18293/SEKE2015-066

CoCoMEP builds upon the established CoCoME case study [5]. CoCoMEP is already in use for collaboration between several projects within the DFG Priority Programme *Design For Future - Managed Software Evolution* (SPP1593) [6]. These projects collected knowledge on experiences and lessons learned on research collaboration in software evolution. CoCoMEP, however, is not limited to SPP1593 but open for reuse and extension by researchers outside the scope of the priority programme. For constructing CoCoMEP, we first analyzed the current support for research collaboration (Sec. 2). Second, we conducted a literature review to examine existing empirical studies (Sec. 3). Based on identified issues and requirements derived, we designed CoCoMEP (Sec. 4). We discuss lessons learned from applying CoCoMEP in SPP1593 (Sec. 5). The paper concludes in Sec. 6.

2 Related Work in Empirical Research

In this section, we analyze related work with regard to collaboration. In particular, we focus on replicability and comparability that are both indispensable to enable research collaboration. The aim is to learn from experiences in empirical research and derive requirements **(R1-6)** as basis for the design of CoCoMEP. On this account, we focus on how other research communities standardized their evaluations to compare different solutions. We are interested in properties that enable or constrain comparability. Furthermore, we examine papers discussing replication in empirical research to consider replicability in our platform.

Standardized Evaluation within certain Research Communities: In Espinha et al. [7], a standard and open-source case study for SOA is proposed. The authors discussed that for case studies in the SOA research community a wide variety of small and closed systems is used limiting comparability of obtained results. Therefore, standardized study subjects are needed to enable assessment of ideas and methods within this community. Another attempt to standardize evaluation has been made in Proksch et al. [8]. They described a framework focusing on evaluations of developer assistance tools. As discussed in the mentioned papers, the standardized case studies have to address the major challenges within a particular community. Otherwise, it will not find broad acceptance.

To compose suitable case studies, we can learn a lot from the repository mining community. In this community, development histories (repositories) are analyzed with respect to certain research questions. To compare results, some projects (e.g., Apache Tomcat, Mozilla Firefox, etc.) exist that are used by several research teams. For example, Lott et al. [9] and Lessmann et al. [10] proposed frameworks for comparative software defect prediction experiments. According to this, all artifacts of the study subject should be made available to motivate a certain community to conduct necessary empirical studies and to achieve better and more convincing result as well as research collaboration. Hence, we consider standardized evaluation as requirement **R1** for joint empirical research.

Replication of Empirical Studies: The aim of standardized evaluation is to enable comparison and replication of empirical and complementary studies. As described in Juristo et al. [4], replication in empirical studies is required to confirm or deny original results as well as to complement the original experiment. However, experiences have shown that replication is hard to achieve [4]. One reason is to establish identical conditions of the experimental context which might be impossible in some cases. In addition, the replicating researchers need to fully understand the experimental design, but most rationale is not provided (tacit knowledge). To enable replication effectively, Schull et al. [11] proposed to provide laboratory packages for experiments. The authors defined a laboratory package as an experimental infrastructure including experiment design, necessary material, and possible variation points. As stated in Mendonça et al. [12], the problem is to compose static laboratory packages that cover all aspects of the experiment. The replicating researchers need to fully understand the experiment and the corresponding material to avoid unpredictable variants in the experiment limiting the meaning of the achieved results. As a consequence, Mendonça et al. proposed that effective replications also require well-defined processes involving the original researchers. This implies the need for an effective collaboration structure among researchers, which we consider as requirement **R2**. For this purpose, they introduced a framework for improving the replication of experiments (FIRE) and emphasize the importance of knowledge sharing for internal and external replications (e.g., experimental details and rationale).

Requirements on a Research Platform: As stated in Demeyer et al. [13], case studies are popular for assessing new approaches relating to evolution, but most of them use toy examples that have a bias towards the approach. Moreover, as stated in Runeson et al. [14], different study subjects and missing documentation of the evaluation process decrease replication and comparability of case studies. To cope with these issues, a standardized study as well as evaluation process is needed. Regarding evolution, Demeyer et al. proposed a set of requirements. **R3**: the case must comprise artifacts that correspond to all life-cycle phases (life-cycle requirement). **R4**: the evolution process must contain iterations and increments (evolution requirement). **R5**: the application, problem, and solution domains of the case must be qualified (domain requirement). **R6**: tools necessary to replicate the case must be evaluated (tool requirement). Our research platform must address these six requirements properly.

3 A Literature Review on Empirical Studies

To understand how well existing studies on software evolution support the requirements identified above we conducted a literature review. This section provides the search strategy, process documentation, and findings of the literature review. Basically, the review followed the guidelines by Kitchenham and Charters [15]. However, it was not conducted as a strict review as every paper was only reviewed by one of the authors. As empirical methods we considered case studies and experiments. We neither aimed at giving a comprehensive overview of approaches for supporting software evolution nor at presenting the approaches found. The research questions (RQ) for our review were:

Which aspects of evolution are addressed explicitly in case studies and experiments? (RQ1) According to the requirements of Demeyer et al. [13], we consider the following aspects of evolution: *Life-cycle* which covers the artifacts, activities and their relationships that correspond to all phases in the system's life-cycle. *Evolution process* covering iterations in the life-cycle which we surveyed by the time horizon of the study (i.e. design-time, run-time, or post-mortem). *Domain* which covers the artifacts to provide a concrete study setting. We did not include the requirement *tool* as this is hard to examine by literature review.

With **RQ2** we again refer to the focus of Sec. 2 by asking **how is comparability and replication supported in case studies and experiments?**

3.1 Paper Search and Selection Process

To answer the research questions publications were required to be related to *evolution*, *information systems* or *software engineering*, and *empirical studies*. In consequence, three sets of keywords were created. The evolution set covers the keywords *evolution*, *maintenance*, *changeability*, and *modifiability*. The domain set contains the keywords *information system* and *software*. The methods set comprises *case study* and *experiment*. We did not derive search terms from RQ2 as comparability and replicability is typically not stated within the single study, but has to be assessed manually. We searched journals (e.g., ESEJ, TOSEM, TSE, KAIS, and ICSM) and conference proceedings (e.g., CSMR, ESEM, ICSE, and FSE) related to empirical research and software evolution with an impact factor greater or equal one and an acceptance rate lower than 30%, respectively.

We performed two selection iterations on the initial amount of 272 search hits. Each iteration was performed by one author guided by defined inclusion and exclusion criteria as proposed by Kitchenham and Charters [15]. The first iteration evaluated whether the papers conformed to the formal requirements on case studies and experiments as described by Runeson et al. [14]. In the second iteration, the contribution of the papers to one or both research questions was evaluated. After the first iteration 105 papers were selected for further analysis. Within the second iteration 53 papers were identified that contribute to the research questions. The identified papers are listed online (`www.dfg-spp1593.de/cocome/platform`) due to page restrictions in this paper.

3.2 Findings

As a general answer to RQ1, no study has been found considering the entire evolution life-cycle. In addition, neither artifacts nor relations between the different development activities are comprehensively covered by existing studies. Distributions for the findings are depicted in Fig. 1. *Focus on design-time*: Our review shows that design-time and post-mortem studies (52 out of 53) outweigh run-time studies (1 out of 53). *Focus on a specific activity*: Most studies are only focused on a specific activity within the life-cycle. In particular, requirements engineering and maintenance phase were least covered. *Focus on a specific artifact*: For supported activities, the approaches usually consider a typical type of artifact, like code or UML models. Only a few studies (2 out of 53) focus on changes of additional documentation. We could not find a study focusing on changing decisions during evolution. Moreover, only a few studies (10 out of 53) cover co-evolution of development artifacts. The majority of these (6 out of 10) covers co-evolution within the same type of artifact, like co-evolution of components or test cases. *Relationships between activities mostly not considered*: Only a few studies (8 out of 53) examine the relationships between activities within different life-cycle phases (requirements engineering and implementation, design and implementation, implementation and maintenance).

The following findings answer RQ2. *Missing comparability and replicability of studies*: Most empirical studies and their results are not *comparable* in terms of domain, size, or complexity. In particular, this is true for controlled experiments, where the complexity of tasks is limited which may lead to less realistic settings. Thus, the obtained results have only limited evidence for software evolution in practice. This is related to the problem of *replicating* empirical studies [4]. Regarding software evolution, replication is difficult to achieve due to a large amount of changes required in the study subjects within a long period of time. Consequently, a complete change history would be required for the study subjects. Moreover, no study in our review made a clear distinction which types of evolution were addressed by given changes. As introduced by Lientz and Swanson [16], three types can be distinguished – *corrective*, *perfective* and *adaptive* evolution. If a case study does not specify the ad-

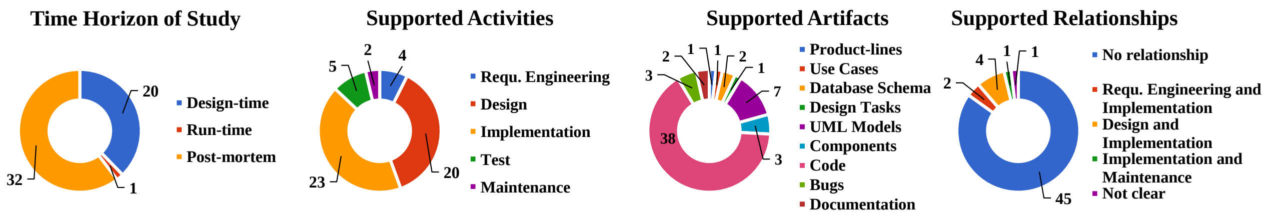

Figure 1. Distribution of Supported Development Time Horizon, Activities, Artifacts and Relationships

dressed evolution type, it is difficult for other researchers to assess whether the study is appropriate for their approach.

Only a few publications provide enough details about their empirical study to enable replication. Most of these studies are performing a post-mortem analysis on code repositories. However, there exist only a few open-source projects for repository mining studies, on which the community for post-mortem analysis agreed. Overall, no common guidelines have been found for studies on software evolution in order to support joint research.

4 The CoCoME Platform

The findings of our literature review clarified the need for improvement in case study research on information system evolution. According to the requirements identified in Sec. 2, we developed the research platform CoCoMEP depicted in Fig. 2. On this account, the established CoCoME system [5] serves as the study subject (Sec. 4.1). We developed examples of change scenarios in information system evolution (Sec. 4.2), constructed sample activities in system development and operation, and arranged them in life-cycle form (Sec. 4.3).

4.1 Evolution Subject

An evolution subject is the amount of artifacts in different revisions (e.g., requirements or monitoring data) that represent an information system. We used CoCoME [5] as evolution subject. CoCoME has been set up in a Dagstuhl research seminar as a common case study on which several methods in the context of component-based software engineering have been applied. Since more and more people do research on software evolution, CoCoME has been applied in new areas as a demonstrator for software evolution methods. CoCoME represents a trading system as it can be observed in a supermarket chain handling sales. This includes processing sales at a single store of the chain, e.g. scanning products or paying, as well as enterprise-wide administrative tasks, e.g. inventory management or reporting. A detailed description of CoCoME is given in [5]. Since CoCoME has been applied and evolved successfully in various research projects, e.g. SLA@SOI (`http://sla-at-soi.eu`) and Q-Impress (`www.q-impress.eu`), several variants exist that span different platforms and technologies, such as plain Java code or service-oriented frameworks. Furthermore, various development artifacts are available, such as requirements specification or design documentation, which changed over time. CoCoME is well suited to serve as evolution subject because the supermarket context is commonly comprehensible and the complexity of the system is appropriate. As CoCoME is a distributed system, several quality properties are affected by evolution.

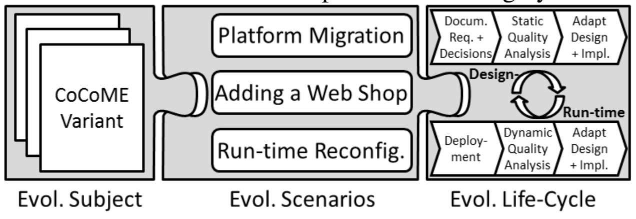

Figure 2. Overview of the CoCoME Platform

4.2 Evolution Scenarios

An evolution scenario describes changes to a certain evolution subject. Based on CoCoME, we implemented distinct evolution scenarios (S1-S3) covering the categories adaptive and perfective evolution (cf. Sec. 3). Corrective evolution is not considered as this merely refers to fixing design or implementation issues. A perfective evolution with regard to a changing environment is represented in S1 by emerging user requirements. An adaptive evolution is reflected in S2 by platform alterations due to evolving technology. Furthermore, in order to accommodate the self-adaptiveness of modern software architectures, reconfiguration during system operation is addressed in S3. Implementation details are visualized online (`www.dfg-spp1593.de/cocome/platform`) due to page restrictions.

S1: Web Shop Extension: A web shop is added where the customers can order online and pick-up the goods at the store. This design-time modification includes adding new use cases and modifying existing design models. S1 represents a requirements-driven evolution that transforms a closed system (only employees can access) to an open system (customers can accessed via internet). Hence, various quality properties are affected, e.g. privacy, security, performance, and reliability.

S2: Platform Migration: The enterprise server and its connected database are now running in the Cloud to reduce operating costs of resources. The introduction of the Cloud enables flexible adaptation and reconfiguration of the system, however, causes new challenges regarding aforementioned quality properties.

S3: Database Migration: During a big advertise campaign, the performance of the system may suffer due to limited capacities of the Cloud provider currently hosting the database. Migrating the database from one Cloud provider to another may solve the scalability issues. S3 represents a reconfiguration at run-time. Migrating the database may cause privacy issues, as described in further detail in [17].

4.3 Evolution Life-Cycle

An evolution life-cycle integrates activities and their relationships required to implement one or more evolution scenarios. We developed a set of sample activities typical in information system evolution and arranged them in life-cycle form (cf. Fig. 2) to cope with aforementioned evolution scenarios.

An iteration in the life-cycle starts with a change request, e.g. for S1 or S2. Decisions are made and documented. A static quality analysis is conducted to identify quality issues at design-time. The design is adapted and implemented. After deployment, a dynamic quality analysis is conducted for the running system which may result in automated adaptation at run-time (S3) or a new iteration for manual evolution.

The life-cycle addresses the findings from literature review as it (i) spans design-time and run-time, (ii) covers various activities located in different phases of software development and operation, (iii) contains a variety of heterogeneous artifacts associated to the life-cycle activities, e.g. requirements, decisions, UML models, monitoring data, simulation data, and (iv) covers relationships between the activities. Tab. 1 gives an excerpt of the review findings and how they are supported by CoCoMEP.

Diverse variants of the three parts of CoCoMEP are possible. However, CoCoMEP is appropriate to conduct empirical studies on software evolution as it covers the requirements (see Sec. 2). **R1**: It provides standardized study subject, evolution scenarios, and life-cycle activities. **R2**: This standardization in conjunction with the community offers a structure for collaboration and study replication (see Sec 5). **R3**: CoCoMEP comprises activities and artifacts that correspond to all phases in the system's life-cycle (life-cycle req.). **R4**: It covers iterations and increments in the development process (evolution req.). **R5**: It provides a concrete setting to qualify the application domain (i.e. supermarket), problem domain (i.e. web-based system) and solution domain (e.g., architecture, code, etc.) of the case (domain req.). **R6**: It supports evaluating the tools necessary to replicate the case, such as implementation/design languages, operating system, or development environments (tool req.).

Finding	**Dimension in Fig. 1**	**CoCoME Platform**
Time horizon	design- and run-time	both [18]
Activities	requirements, design, maintenance	use cases [19]/static [20]/ dynamic analysis [18]
Artifacts	documentation, UML models, code	design decisions [19], simulation/instrumentation data [18, 21], java code
Relation	req. and impl.	all phases related

Table 1. Supported Review Findings (excerpt)

5 Lessons Learned

In this section, we discuss experiences (wrt. outcomes of Sec. 2 and 3) from applying CoCoMEP in the DFG Priority Programme 1593 which comprises 13 research projects with a focus on long-living systems [6]. The application is exemplified in [18, 19, 20, 21]. CoCoMEP proved to be a suitable knowledge base and supported us in: *(i) Gathering project-spanning understanding on activities and artifacts wrt. evolution.* Mapping the diverse activities and artifacts specific to the single projects within the priority programme into the given life-cycle structure enabled a common understanding of them. Furthermore, common understanding has been supported by a joint communication and documentation infrastructure, i.e. mailing lists, media wiki, SVN repository. The wiki contains all the information about life-cycle activities and related artifacts to be shared and refined among the projects. We use the SVN repository to share source code as well as configuration and documentation artifacts. Based on the life-cycle and infrastructure it was easy to identify and solve uncertainties and misunderstandings among the projects and to create a project-spanning understanding. This is one foundation for research collaboration (i.e. comparability and replication). *(ii) Identifying common artifacts.* Mapping activities and artifacts into the life-cycle allows for identifying artifacts used by diverse projects and relations between artifacts. This is another foundation for research collaborations. *(iii) Reuse of activities and artifacts.* Mapping activities and artifacts into the life-cycle allows for reusing them among the projects and for others. In the priority programme context, the output of activities associated to one project is often reused as an input for activities associated to another project. Using the artifacts in subsequent activities by another project contributes to the evaluation of the artifacts and thus the applied approaches. Furthermore, activities are reused as they are performed by two or more projects. This also contributes to the evaluation of the approaches applied by one projects by comparison to another project. *(iv) Clarifying interfaces between projects.* Project-spanning understanding and knowledge about dependencies between activities and artifacts supports clarifying the interfaces between the single projects. This leads to distribution of responsibilities and thus results in more efficient collaborations. For example, if a required artifact has already been created by one project, it can often be reused by another project without additional effort. *(v) Using feedback loop*. Including design-time and run-time in the life-cycle allows for analyzing the effects of design decisions at run-time within the same study. This is in contrast to existing studies, which are mostly limited to design and implementation. *(vi) Establishing a technical basis*. CoCoMEP contributed to the development of a common technical basis between the single projects. It supported us in developing tools that interact with each other based on clearly defined interfaces and in configuring common execution environments. Joint tool development and configuration reduces effort for the single projects. Additionally, the integrated tooling eases collaboration while evaluation.

Applying CoCoMEP in the priority programme context, however, showed some potentials for improvement. *Change history of some artifacts is rather short*. Since the priority programme started in 2012, artifacts still face few evolutionary changes compared to ordinary repository mining studies for instance. This is caused by the fact that CoCoME is a research prototype and we do not have the amount of resources (human and financial) involved in real-life development. Nevertheless, as shown by studies in the priority programme, CoCoME provides a sufficient knowledge basis so far for conducting various analysis, e.g. on use cases, decisions, or monitoring and simulation data. We are confident to produce a larger change history in the future as the priority programme continues for further three years and simultaneously CoCoME is applied in a growing number of studies beyond the programme.

6 Conclusion

Based on requirements for collaboration support from related work and a literature review on empirical studies on software evolution, we developed CoCoMEP. The platform consists of three interconnected parts – an established study subject, related evolution scenarios, and a life-cycle covering activities to address the scenarios. Thus, it supports collaboration in and replication of empirical studies by enabling common understanding and reuse of activities and artifacts, interfaces between projects and technical infrastructure, as perceived while applying CoCoMEP in a large research priority programme. In short, CoCoMEP is expected to provide the following benefits to researchers: (i) less effort in scenario definition, study setup and execution, as well as (ii) increased evaluation confidence and (iii) community acceptance by interaction with others. In the future, the subject CoCoME will be further modified to create new and evolve existing artifacts by new evolution scenarios such as the introduction of mobile clients. These scenarios may include parallel evolution and co-evolution of artifacts which are difficult to achieve in most empirical settings.

References

[1] J. O'Brien and G. Marakas, *Introduction to Information Systems*, 15th ed. McGraw-Hill, 2010.

[2] M. M. Lehman and L. A. Belady, Eds., *Program Evolution: Processes of Software Change*. Academic Press, 1985.

[3] D. I. Sjoberg et al., "The future of empirical methods in software engineering research," in *Future of Software Engineering*. IEEE, 2007, pp. 358–378.

[4] N. Juristo and O. Gómez, "Replication of software engineering experiments," *Empirical software engineering and verification*, pp. 60–88, 2012.

[5] S. Herold et al., "CoCoME – the common component modeling example," in *The Common Component Modeling Example*. Springer, 2008, pp. 16–53.

[6] U. Goltz et al., "Design for future: managed software evolution," *CSRD*, pp. 1–11, 2014.

[7] T. Espinha et al., "Maintenance research in SOA - towards a standard case study," in *CSMR*. IEEE, 2012, pp. 391–396.

[8] S. Proksch et al., "Towards standardized evaluation of developer-assistance tools," in *RSSE'14*. ACM, pp. 14–18.

[9] C. Lott and H. Rombach, "Repeatable software engineering experiments for comparing defect-detection techniques," *Empirical Software Engineering*, vol. 1, no. 3, 1997.

[10] S. Lessmann and B. Baesens, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE TSE*, vol. 34, no. 4, pp. 485–496, 2008.

[11] F. Shull et al., "Replicating software engineering experiments: addressing the tacit knowledge problem," *Intl. Symposium on Empirical Software Engineering*, pp. 7–16, 2002.

[12] M. G. Mendonça et al., "A Framework for Software Engineering Experimental Replications," *ICECCS*, pp. 203–212, 2008.

[13] S. Demeyer et al., "Towards a Software Evolution Benchmark," in *IWPSE*. ACM, 2001, pp. 174–177.

[14] P. Runeson et al., *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley, 2012.

[15] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University, Tech. Rep., 2007.

[16] B. P. Lientz and B. E. Swanson, *Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*. Addison-Wesley, 1980.

[17] R. Heinrich et al., "Integrating run-time observations and design component models for cloud system analysis," in *MRT*. CEUR Vol-1270, 2014, pp. 41–46.

[18] W. Hasselbring et al., "iObserve: integrated observation and modeling techniques to support adaptation and evolution," CAU Kiel, Tech. Rep. 1309, 2013.

[19] S. Gaertner et al., "Capturing and Documentation of Decisions in Security Requirements Engineering through Heuristics," *SWT-Trends*, vol. 34, no. 1, pp. 21–22, 2013.

[20] R. Heinrich et al., "Architecture-based analysis of changes in information system evolution," *WSRE, SWT-Trends*, vol. 34, no. 3, 2015.

[21] R. Heinrich et al., "Run-time architecture models for dynamic adaptation and evolution of cloud applications," CAU Kiel, Tech. Rep. 1503, 2015.