A Decade of Reverse Engineering at Fraunhofer IESE – The Changing Role of Reverse Engineering in Applied Research[1]

Jens Knodel and Dirk Muthig

Fraunhofer Institut für Experimentelles Software Engineering (IESE)
Fraunhofer-Platz 1, D-67663 Kaiserslautern, Germany
{jens.knodel, dirk.muthig}@iese.fraunhofer.de

Abstract: The Fraunhofer Institute for Experimental Software Engineering (IESE) recognized reverse engineering as an essential competence and thus has been active in the field since 1996, which is right from its beginning. The role of reverse engineering, however, within the institute changed over time. Since Fraunhofer IESE is an applied research institute its competencies are tailored to the market to be and stay attractive to industry. This paper analyzes the evolution of the competence reverse engineering in retrospective and derives a picture of the role and importance of reverse engineering to practice in the last ten years where reverse engineering moved from a set of techniques applied in pure reengineering project to an enabling technology for other software engineering disciplines. Furthermore, we present our vision for the future role of reverse engineering at Fraunhofer IESE, as a supportive technology for the development of software, which gives constant and continuous feedback already at construction time.

1 Introduction

The Fraunhofer Institute for Experimental Software Engineering (IESE) provides services to industry for improving their software and system engineering practices. The concrete services are based on state-of-the-art technology and methods, as well as aim for sustainable and measurable effects in industrial practice. Reverse engineering (REE) – i.e., identifying components and dependencies within software systems and representing them at higher levels of abstraction [CC90] – has been an essential competence for important services applied in many projects since IESE has been founded in 1996. REE is always required when crucial information on existing software systems must be recovered from diverse artifacts including source code, models, or documentation. REE has been constantly applied in many projects covering all kinds of domains, which is documented by numerous publications at major conferences and workshops, such as the International Conference on Software Maintenance (ICSM), the Working Conference on Reverse Engineering (WCRE), the European Conference on Software Maintenance and Reengineering (CSMR), or the Workshop on Software-

[1] This work was performed as part of the project ArQuE (Architecture-Centric Quality Engineering), which is partially funded by the German Ministry of Education and Research (BMBF) under grant number 01 IS F14.

Reengineering (WSR). Its nature, however, has changed over time while going through four distinct stages:

1. **Classical reverse engineering,** the examination and transformation of (parts of) existing system into a new form mainly motivated by maintenance scenarios. Prominent examples are the Y2K problem or the Euro introduction.
2. **Request-driven reverse engineering,** analyses of selected parts of existing systems triggered by concrete requests initiated in the context of development or evolution scenarios where limited aspects of field-tested systems are extracted, evaluated, or migrated. We distinguish basic analyses (low effort, quick results) and more advanced, specialized analyses (high effort, longer analysis time), which are all performed on demand only. IESE applied this form of REE mostly in projects realizing the next product generation.
3. **Integrated reverse engineering,** performing REE as an integral part of the architecting creation and evolution process. It is not longer initiated explicitly but is performed as an inherent and natural part of architectural activities developing and maintaining high-quality (product line) architectures for various purposes.
4. **Constructive reverse engineering,** exploiting REE continuously over the lifetime of software starting with the first day of development. Thereby modifications are instantly analyzed and immediate feedback to developers is given. Continuous and prompt feedback aims at ensuring consistency and compliance to thus prevent software systems to degenerate.

This paper presents the four stages in detail, as well as rationales for the transition from one stage to another. Figure 1 puts the four stages along a timeline that starts with the IESE's foundation in 1996.

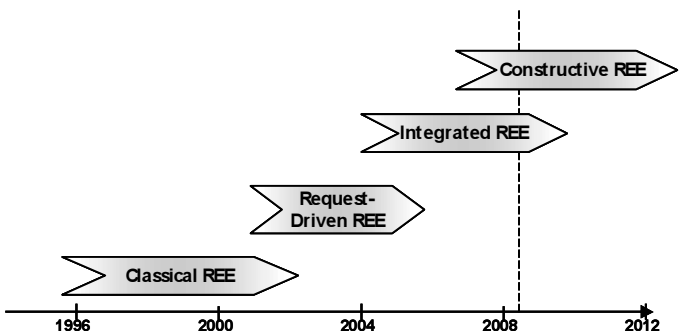

Figure 1: Reverse Engineering at Fraunhofer IESE

Sections 2 and 3 look back on the first two stages of the past, the transition from classical reverse engineering to its request-driven application. While today REE is more and more applied at earlier points in time during software development, Sections 4 and 5 thus present the two recent stages. Section 6 concludes the paper.

2. Classical Reverse Engineering and Reengineering

In the first stage – classical reverse engineering –reverse engineering had research-centric orientation at Fraunhofer IESE. New reverse engineering technologies were developed to support industrial projects covering mainly pure reengineering projects. Our reverse engineering work back then addressed the integration and use of aspects of the application domain to improve reverse engineering results (e.g., see [De96], [De97]), component identification, abstract data type detection, and clustering (e.g., [GK97], [GKS97]) and fact extraction (e.g., [KC03, KP03]). Furthermore, in joint research, contributions (e.g., [Cz00]) were made to early versions of the Bauhaus tool [RVP06].

Product line engineering – emerging 1990s – became one of Fraunhofer IESE main research fields. This had impact on our reverse engineering work, which is, for instance, captured in [Ba99b] and [De98]. One of the key publications in this early stage introduces the approach RE-PLACE (Reengineering Enabled Product Line Architecture Creation and Evolution, which first reengineered parts of a legacy system implemented (from Fortran to C++) and then introduced product line engineering concepts [Ba99b]. Figure 2 depicts the RE-PLACE approach. The main focus, however, was on the reengineering part and often transformation scripts, parser and analyses were developed.

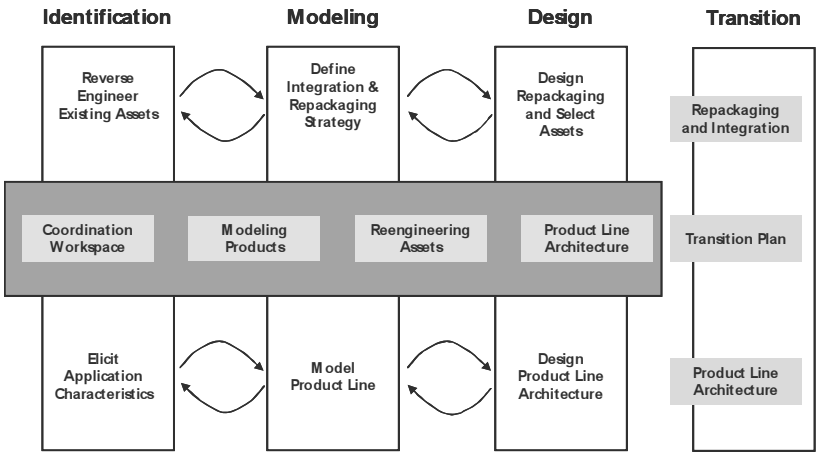

Figure 2: Classical Reverse Engineering and Reengineering (RE-PLACE)

However, though successful in industrial projects and dissemination, the research-centric development of reverse engineering technologies did not match the strategy of Fraunhofer IESE to be a mediator between research and practice. Moreover, the development of new reverse engineering technologies like clustering or parsers was a risk in time-critical projects. Instead of solving the industrial partners' problems, we were busy with the development of our own analyzers, which not always lead to the expected results in terms of quality, time and effort. Therefore, we re-focused our reverse engineering activities and entered the stage of request-driven reverse engineering.

3 Request-driven Reverse Engineering

Request-driven reverse engineering was the first step from research to applied research in reverse engineering at Fraunhofer IESE. We started to focus rather on the selection, adaptation and applicability of reverse engineering technology than to develop new technology. By the term request-driven we understand analyses of existing systems triggered by explicit requests that aim at reusing solutions implemented in field-tested systems supported by a predefined analyses catalogue. The selection of appropriate analyses is strongly goal-driven by the intended usage of the expected results. Explicit decision making whether or not to execute an analysis is based on the project context. The main decision criteria are the usefulness of the expected results, the effort to apply a certain analyses technique, and the appropriateness of the technique. Analyses are only executed on-demand. Therefore a reasonable usage of available resources (system experts, reverse engineering experts) is achieved. Usually, a combination of several analyses has to be selected in order to solve the reverse engineering problem (i.e., to provide a useful response to the request). Request-driven reverse engineering has two major phases (see Figure 3, and [KG04]): extraction and analysis, both are based on a common infrastructure that provides access to analyses tools and visualizes the results:

- The fact extraction phase that exploits artifacts (e.g., fact extraction from source code is usually done by parsing or pattern matching) and populates the fact base or repository. Raw data contained in artifacts such as source code, code comments, configuration files, user documentation, and architectural descriptions is aggregated in a fact base. Facts about existing systems (e.g., method A calls B, B is a method of class C) represent low-level information (i.e., classified data with relations among entities). Since relevant information is often hidden in overcrowded low-level facts, the analysis phase abstracts and aggregates the data. In short, transformation of data contained in artifacts into information presented in relevant views.
- The analysis phase identifies the facts relevant to a given request and then abstracts these facts to the level of abstraction appropriate for their intended users. These abstractions are packaged into views – the responses – that their intended users can understand based on their knowledge of the system. The analysis phase explicitly distinguishes between basic and detailed analysis.
    - **Basic analysis:** Basic analyses are predefined, parameterized, repeatable analyses that can be applied with reasonable usage of available resources. Basic analyses transform a set of relevant facts into information visualized in views. These analyses can be regarded as a standard catalogue which can be executed directly with only slight adaptation. Examples for such basic analyses are context analysis of code entities, visualization of class and inheritance hierarchies, call graphs, compliance checking.
    - **Detailed analysis:** Detailed analyses typically require additional effort for their application, for extending the analysis infrastructure, for fact extraction, and often require several iterations and calibration until they produce adequate results. They aim at getting a deeper understanding of system aspects, and therefore need more effort and expert involvement.

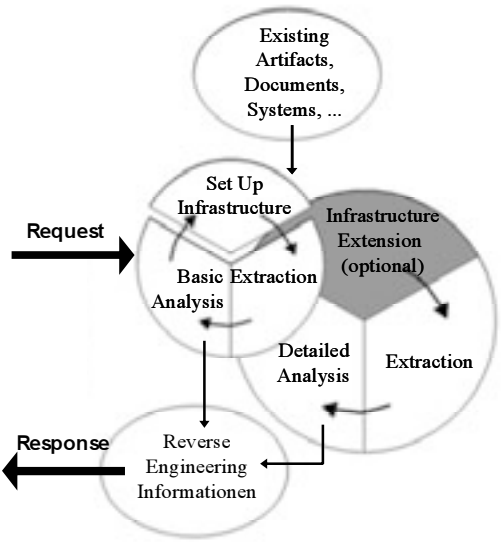

Figure 3: Request-driven Reverse Engineering (Fraunhofer ADORE™)

This request-driven reverse engineering was marketed under Fraunhofer ADORE™ (Architecture- and Domain-Oriented Re-Engineering) trademark and had a strong focus on the migration of existing single systems into a software product line. For basic analyses, we strongly emphasized the product line engineering focus. We focused on the selection and combination of reverse engineering technique (e.g., see [Ba04], [KFG07], [KG04]), the extraction of reusable components (e.g., [GK05], [KM05]), and monitoring the quality of existing systems (e.g., [GVG04], [Ve05]). For detailed analyses, we often partnered with other researchers (e.g., [Kn05], [Pi03]). Over time, the request-driven reverse engineering synthesized a set of certain standard analyses that became part of the Fraunhofer PuLSE™–DSSA, our approach comprising all activities of defining, documenting, maintaining, improving, and certifying proper implementation of a (product line) architecture. Eventually, the analyses were realized as part of the Fraunhofer SAVE (Software Architecture Visualization and Evaluation) tool, which is a product line for analyzing and optimizing the architecture of implemented systems.

4. Integrated Reverse Engineering

The current stage – integrated reverse engineering – considers reverse engineering as an underlying activity of software engineering disciplines like architecting, quality assurance, or product line engineering. In this stage, we use reverse engineering as an enabling technology. This is also reflected our publications, where we had a shift towards experience reports about applications in industry. Integrated reverse engineering became a substantial part of PuLSE™-DSSA. PuLSE™-DSSA is the part of the Fraunhofer PuLSE™ approach [Ba99a] that focuses on the definition and development of product lines architectures. The main underlying concepts of the PuLSE-DSSA

method are (see Figure 4) are scenario-based, incremental development with successively prioritized requirements and view-based documentation to support the communication of different stakeholders. Reverse engineering activities are directly integrated into the architecting process regularly executed. The main process loop of PuLSE-DSSA consists of four major steps (see Figure 4):

- **Planning:** The planning step defines the content and delineates the scope of the current iteration: the selection of a limited set of scenarios, the identification of relevant stakeholders, as well as the decision whether or not to include an architecture assessment at the end of the iteration.
- **Realization:** The realization step selects solutions and makes design decisions in order to fulfill the requirements. Prototypes may be created to evaluate the soundness of the solutions, and reverse engineering is either used to analyze the prototypes or the existing legacy code, COTS components, or peripheral systems.
- **Documentation:** This step documents architectures by using an organizational-specific set of architectural views. It relies on standard views as and customizes or complements them by additional aspects as requested by key stakeholders.
- **Assessment:** The goal of the optional assessment step is to analyze and evaluate the resulting architecture with respect to functional and quality requirements and the achievement of business goals.

The integrated reverse engineering synthesized certain standard analysis like metrics computation, architecture compliance (e.g., see [Kn06a], [KP07]), ownership analysis (e.g., [Ga06]), and the visualization of software architectures (e.g., see [KMN06], [Kn06b]), assessment of reuse potentials in existing systems (e.g., see [Ko06a], [Ko06c], [YGM06]) or reverse engineering as an quality engineering instrument (e.g., see [Ko06b],) mostly in a product line context.

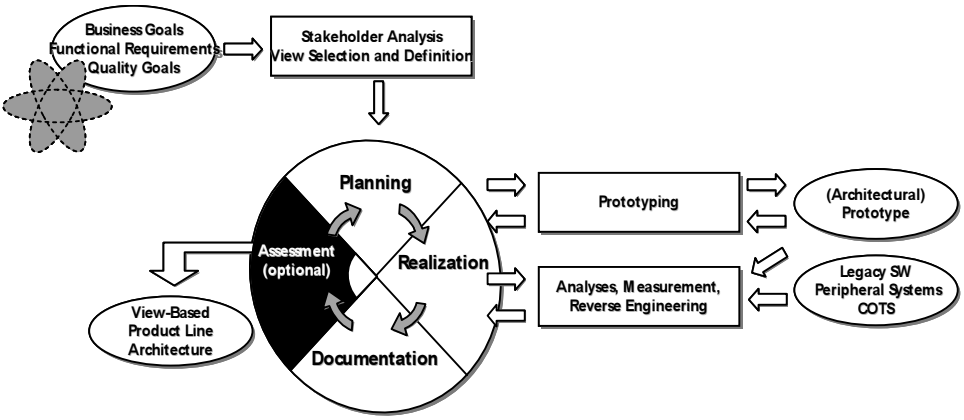

Figure 4: Architecture Creation Process (Fraunhofer PuLSE™–DSSA)

5. Constructive Reverse Engineering

Our practical experiences in industrial projects show that the reverse engineering part is usually applied late in the development process (e.g., to conduct architecture compliance checking, the implementation has to be realized in order to have the counterpart that can be checked against the specified architecture). The late application of reverse engineering detects issues, but often there is neither time nor effort left to fix or refactor them in an adequate manner. The overhead for addressing the issues identified can not be addressed before the envisioned release dates or market delivery, since the application was only late in the process. Our vision for constructive reverse engineering aims at tackling this problem. Hence, constructive reverse engineering starts as early as the software development begins. System artifacts produced or modified are continuously reverse engineered and allow giving immediate feedback, as soon as there is a deviation from the planned or intended specification. The goal constructive reverse engineering is to prevent system decay and ensure traceability in the software development lifecycle. At the moment, we are in the transition phase towards constructive reverse engineering, from our current viewpoint the predominant stage for the upcoming years. We envision an increasing the frequency in which reverse engineering technologies are applied so that these analyses run constantly in the background. When issues or deviations are encountered the developers receive prompt and immediate feedback in real time. This live feedback allows them to fix issues as soon as they are introduced.

To achieve our vision, we follow the general approach for continuous development and quality improvement by Deming [De86]. This approach consists of four consecutive stages: Plan, Do, Check, Act. In the "Plan" stage, the root cause of the problem is determined and then a change or a test aimed at improvement is planned. This change or test is then carried out during the "Do" stage. In the "Check" stage, it is then checked whether the desired result was achieved, what or if anything went wrong, and what was learned. Finally, in the "Act" stage, the change is adopted if the desired result was achieved. If the result was not as desired, however, the cycle is repeated using knowledge obtained from the previous cycle. The challenges for constructive reverse engineering are thereby to adapt mature analyses techniques to this new way of applying them. The computation has to be fast, since we want to provide feedback about reverse engineering results in real-time. The analyses have to be robust against incomplete (sometimes even not compiling) source code, being currently under development. Our vision for constructive reverse engineering is to make it an integral part of forward engineering development environments. Adopting the Plan-Do-Act-Check paradigm the constructive reverse engineering results in the following four phases:

- **Plan:** This phase decides and, if necessary, refines the rules, the specifications or thresholds values for analyses to which the software system under development should comply. The step is repeated in each increment of the development, but once the rules matured, it will only consist of short reviews and minor adaptations.
- **Do:** The do-phase is then the construction of the solutions, in other words, the developers implement the system or they modify other system artifacts.

- **Check:** The check phase is conducted continuously, constantly and concurrently to the do-phase and analyzes the modification made in the do-phase.
- **Act:** The continuously checking enables immediate action in the act phase, which is only entered in case deviations from the plan were detected.

Following this instantiation of the Plan-Do-Check-Act paradigm, reverse engineering has a direct impact on the construction of the software systems. Our vision is to establish an evolution control center that provides continuous, customizable, tool-supported and integrated measurement, assessment and controlling based on reverse engineering. One step further, decision support and recommendations how to countersteer certain development trends or the identification of recurring problems could complement such an evolution control center. We currently investigate in research projects to which degree our vision can become reality. First results integrating architecture compliance checking into the development of software system are promising (e.g., [Mi04], [RKK07]). Recently, we investigated the impact of continuous analyses in a controlled experiment with students of the Technical University of Kaiserslautern, where we monitored the implementation of components for a period of five weeks. The student developers were split into two groups (one with continuous architecture compliance checking support and the other without). We could show that the supported group had constantly 60% less architectural violations in the implementation of their components [RKK07].

6 Summary

In this paper we present a retrospective review of reverse engineering at Fraunhofer IESE. Since the institute's foundation, our view on and the way how we applied reverse engineering evolved in four stages: classical, request-driven, integrated, and constructive (the latter is still in an early stage). We described the motivation for the transitions form one stage to the other, and give an overview on representative work in this field. The vision of constructive reverse engineering sketches the outlook on our future work. Currently, we aim at achieving this vision and the first positive results make us confident that this is the right direction to go for us, as an applied research institute.

References

[Ba04] Bayer, J., Girard, J.-F., Knodel, J., Kolb, R., Muthig, D.: Architekturentwicklung, basierend auf existierenden Systemen. Software-Produktlinien. Methoden, Einführung und Praxis. dpunkt.Verlag, Heidelberg (2004) 165-176

[Ba99a] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., DeBaud, J.-M.: PuLSE: A Methodology to Develop Software Product Lines. Proceedings of the Fifth Symposium on Software Reusability. SSR'99. ACM Press, Los Angeles (1999) 122-131

[Ba99b] Bayer, J., Girard, J.-F., Würthner, M., DeBaud, J.-M., Apel, M.: Transitioning Legacy Assets to a Product Line Architecture. Software Engineering ESEC/FSE'99. Springer-Verlag, Berlin (1999) 446-463

[CC90] Chikofsky, E., Cross, J.H.: Reverse Engineering and Design Recovery: a Taxonomy. IEEE Software 7 (January 1990) 13-17

[Cz00] Czeranski, J., Eisenbarth, T., Kienle, H.M., Koschke, R., Plödereder, E., Simon, D., Zhang, Y., Girard, J.-F., Würthner, M.: Data Exchange in Bauhaus. 7th Working Conference on Reverse Engineering. WCRE'2000 - Proceedings. IEEE Computer Society, Los Alamitos (2000) 293-295

[De86] Deming, W.E.: Out of the crisis. Center for Advanced Engineering Study, Massachusetts Institute of Technology, Cambridge, MA. (1986)

[De96] DeBaud, J.-M.: Lessons from a Domain-Based Reengineering Effort. 3rd Working Conference on Reverse Engineering. WCRE'96 - Proceedings. IEEE Computer Society, Los Alamitos (1996) 217-226

[De97] DeBaud, J.-M.: DARE: Domain-Augmented ReEngineering. 4th Working Conference on Reverse Engineering. WCRE'97 - Proceedings. IEEE Computer Society, Los Alamitos (1997) 164-173

[De98] DeBaud, J.-M., Girard, J.-F.: The Relation Between the Product Line Development Entry Points and Reengineering. Development and Evolution of Software Architecture for Product Families. Springer-Verlag, Berlin (1998) 132-139

[Ga06] Ganesan, D., Muthig, D., Knodel, J., Yoshimura, K.: Discovering Organizational Aspects from the Source Code History Log during the Product Line Planning Phase - A Case Study. 13th Working Conference on Reverse Engineering. WCRE'2006 - Proceedings. IEEE Computer Society, Los Alamitos (2006) 211-220

[GK05] Ganesan, D., Knodel, J.: Identifying Domain-Specific Reusable Components from Existing OO Systems to Support Product line Migration. First International Workshop on Reengineering towards Product Lines. R2PL 2005 - Proceedings (2005) 16-20

[GK97] Girard, J.-F., Koschke, R.: Finding Components in a Hierarchy of Modules: a Step towards Architectural Understanding. International Conference on Software Maintenance 1997. ICSM'97 - Proceedings. IEEE Computer Society, Los Alamitos (1997) 58-65

[GKS97] Girard, J.-F., Koschke, R., Schied, G.: A Metric-based Approach to Detect Abstract Data Types and State Encapsulations. 12th IEEE International Automated Software Engineering Conference ASE'97 - Proceedings. IEEE Computer Society, Los Alamitos (1997) 82-89

[GVG04] Girard, J.-F., Verlage, M., Ganesan, D.: Monitoring the Evolution of an OO System with Metrics: an Experience from the Stock Market Software Domain. 20th IEEE International Conference on Software Maintenance. ICSM'2004 - Proceedings. IEEE Computer Society, Los Alamitos (2004) 360-367

[KC03] Knodel, J., Calderon-Meza, G.: A Meta-Model for Fact Extraction from Delphi Source Code. Proceedings of the First International Workshop on Meta-Models and Schemas for Reverse Engineering (2003)

[KFG07] Knodel, J., Forster, T., Girard, J.-F.: Comparing Design Alternatives from Field-Tested Systems to Support Product Line Architecture Design. 9th European Conference on Software Maintenance and Reengineering. CSMR 2005 - Proceedings. IEEE Computer Society, Los Alamitos (2005) 344-353

[KG04] Knodel, J., Girard, J.-F.: Request-driven Reverse Engineering for Product Lines. Workshop Reengineering Prozesse (RePro 2004). Fallstudien, Methoden, Vorgehen, Werkzeuge - Proceedings (2004)

[KM05] Knodel, J., Muthig, D.: Analyzing the Product Line Adequacy of Existing Components. First International Workshop on Reengineering towards Product Lines. R2PL 2005 - Proceedings (2005) 21-25

[KMN06] Knodel, J., Muthig, D., Naab, M.: Understanding Software Architectures by Visualization - An Experiment with Graphical Elements. 13th Working Conference on Reverse Engineering. WCRE'2006 - Proceedings. IEEE Computer Society, Los Alamitos (2006) 39-48

[Kn05] Knodel, J., John, I., Ganesan, D., Pinzger, M., Usero, F., Arciniegas, J.L., Riva, C.: Asset Recovery and Their Incorporation into Product Lines. 12th Working Conference on Reverse Engineering. WCRE'2005 - Proceedings. IEEE Computer Society, Los Alamitos (2005) 120-129

[Kn06a] Knodel, J., Lindvall, M., Muthig, D., Naab, M.: Static Evaluation of Software Architectures. 10th European Conference on Software Maintenance and Reengineering. CSMR 2006 - Proceedings. IEEE Computer Society, Los Alamitos (2006) 277-286

[Kn06b] Knodel, J., Muthig, D., Naab, M., Zeckzer, D.: Towards Empirically Validated Software Architecture Visualization. ACM Symposium on Software Visualization, SOFTVIS 06 - Proceedings. ACM Press, New York (2006) 187-188

[Ko06a] Kolb, R., Ganesan, D., Muthig, D., Kagino, M., Teranishi, H.: Goal-Oriented Performance Analysis of Reusable Software Components. Reuse of Off-the Shelf Components. 9th International Conference, ICSR 2006 - Proceedings. Springer-Verlag, Berlin (2006) 368-381

[Ko06b] Kolb, R., John, I., Knodel, J., Muthig, D., Haury, U., Meier, G.: Experiences with Product Line Development of Embedded Systems at Testo AG. 10th International Software Product Line Conference, SPLC 2006 - Proceedings. IEEE Computer Society, Los Alamitos (2006) 172-181

[Ko06c] Kolb, R., Muthig, D., Patzke, T., Yamauchi, K.: Refactoring a legacy component for reuse in a software product line: a case study. Journal of Software Maintenance and Evolution Research and Practice 18 (2006) 109-132

[KP03] Knodel, J., Pinzger, M.: Improving Fact Extraction of Framework-Based Software Systems. 10th Working Conference on Reverse Engineering. WCRE'2003 - Proceedings. IEEE Computer Society, Los Alamitos (2003) 186-195

[KP07] Knodel, J., Popescu, D.: A Comparison of Static Architecture Compliance Checking Approaches. Sixth Working IEEE/IFIP Conference on Software Architecture, WICSA 2007. IEEE Computer Society, Los Alamitos (2007)

[Mi04] Miodonski, P., Forster, T., Knodel, J., Lindvall, M., Muthig, D.: Evaluation of Software Architectures with Eclipse (2004)

[Pi03] Pinzger, M., Gall, H., Girard, J.-F., Knodel, J., Riva, C., Pasman, W., Broerse, C., Wijnstra, J.G.: Architecture Recovery for Product Families. Fifth International Workshop on Product Family Engineering. PFE-5 - Proceedings (2003)

[RKK07] Rost, D., Knodel, J., Knauber, P.: Real-Time Tracking of Evolving Software Architectures. (2007)

[RVP06] Raza, A., Vogel, G., Plödereder, E.: Bauhaus - A Tool Suite for Program Analysis and Reverse Engineering. In: Reliable Software Technologies, Ada-Europe 2006 (2006) 71-82

[Ve05] Verlage, M., Girard, J.-F., Ganesan, D., Kiesgen, T.: Überwachung der Qualität der Architektur einer Software-Produktlinie am Beispiel eines web-basierten Wertpapierinformationssystems. Software Engineering 2005 - Proceedings. GI - Gesellschaft für Informatik, Bonn (2005) 243-254

[YGM06] Yoshimura, K., Ganesan, D., Muthig, D.: Assessing Merge Potential of Existing Engine Control Systems into a Product Line. Proceedings of the Third International Workshop on Software Engineering for Automotive Systems. SEAS 2006. ACM Press, New York (2006) 61-67