Towards Understanding the Effects of Requirements Engineering Techniques

Jürgen Münch

Fraunhofer Institute for Experimental Software Engineering,
Sauerwiesen 6, 67661 Kaiserslautern, Germany, E-Mail: muench@iese.fraunhofer.de

Abstract. Developing software of a predetermined quality in a predictable way requires a deep understanding of the effects of software engineering techniques (such as requirements analysis techniques) in different project and organizational contexts. As a consequence, the development or improvement of software engineering techniques should include the determination of their effects in varying contexts. This allows for comparing new or modified techniques with established state-of-the-art techniques and reduces risks in transferring innovative software engineering techniques originating from basic research into industry. This article presents a stepwise and empirically-based approach for transferring software engineering techniques from basic research to industrial usage. The approach is being applied in the context of the project A4 ("Experimenting with Techniques for Generic Software Product Lines") of the temporary research institute SFB 501 ("Development of Large Systems with Generic Methods"). The description of the approach is illustrated by an application example from the requirements engineering field.

1 Introduction

The efficient application of techniques and tools by industrial developers requires precise knowledge about the constraints and effects of their application: It is necessary to understand under which circumstances techniques and tools are applicable and how appropriate they are. This is especially true for the introduction of new technologies. Therefore, successful transfer requires that technologies have to be systematically proven and evaluated before their industrial application. Not until empirical studies and ensuing qualitative and quantitative analysis have discovered strengths and weaknesses of the respective technologies, is a secured assertion possible for which task and under which constraints these technologies are suited. In the context of the SFB 501 [10], empirical studies of different kinds are performed in order to understand the effects of different techniques and tools. In detail, the transfer proceeds in the following steps:

1. In the context of basic research, new development technologies (i.e., techniques and tools) are being developed or existing technologies are modified with the intention of improving them.

2. Then, the conceptually designed techniques and tools are being evaluated in academic contexts (e.g., laboratory studies, student experiments in a university environment) [9]. Local evaluations are distinguished from global evaluations: Local evaluation aims at understanding cause-effect relationships of individual techniques and tools. Global evaluations aim at understanding the interdependencies with other development techniques (such as the combination of a new requirements analysis technique with a state-of-the-art design technique). Additionally, it can be examined whether individual techniques or tools lead to positive results in the context of a full lifecycle process (i.e., reduction of the overall development effort, quality improvements, reduction of development time).
3. With the gained experience, a stepwise evaluation and improvement under industrial constraints in selected pilot projects is performed. The goal is to understand the effects of techniques and tools in industrial contexts. Practice-oriented problems should be considered and solutions should be created that can not be realized in pure university environments. Furthermore, this offers the possibility to evaluate basic research results together with experts from a specific application domain (e.g., automotive).

The SFB project A4 ("Experimenting with Techniques for Generic Software Product Lines") aims at evaluation and industrial transfer of software development techniques that enhance reusability and genericity. The industrial transfer approach is based on the PuLSE^{TM} (Product Line Software Engineering) framework for product line engineering [3] (see Figure 1). Product line engineering approaches define how to take advantage of commonalities and variabilities of products and create reusable assets that increase the efficiency of developing several products by reusing the core functionality [6]. The framework allows for establishing product lines in industrial contexts in order to overcome difficulties with unmanaged diversity among product variants, unstructured reuse of analysis and design information, maintenance of multiple code bases, conceptual integrity among product family members, etc. Benefits are expected with respect to improved ways for reuse of software artifacts and documentation, faster time-to-market, explicit knowledge representation, and improved quality. The core steps of the PuLSE framework are:

1. Baselining the organization and customizing the framework.
2. Scoping the application area based on a sound economic analysis.
3. Modeling that area in terms of concepts and their relationships.
4. Transitioning the domain model into a fully reusable design (a reference architecture).
5. Specifying an application engineering process that makes use of the reference architecture and maintains over time.

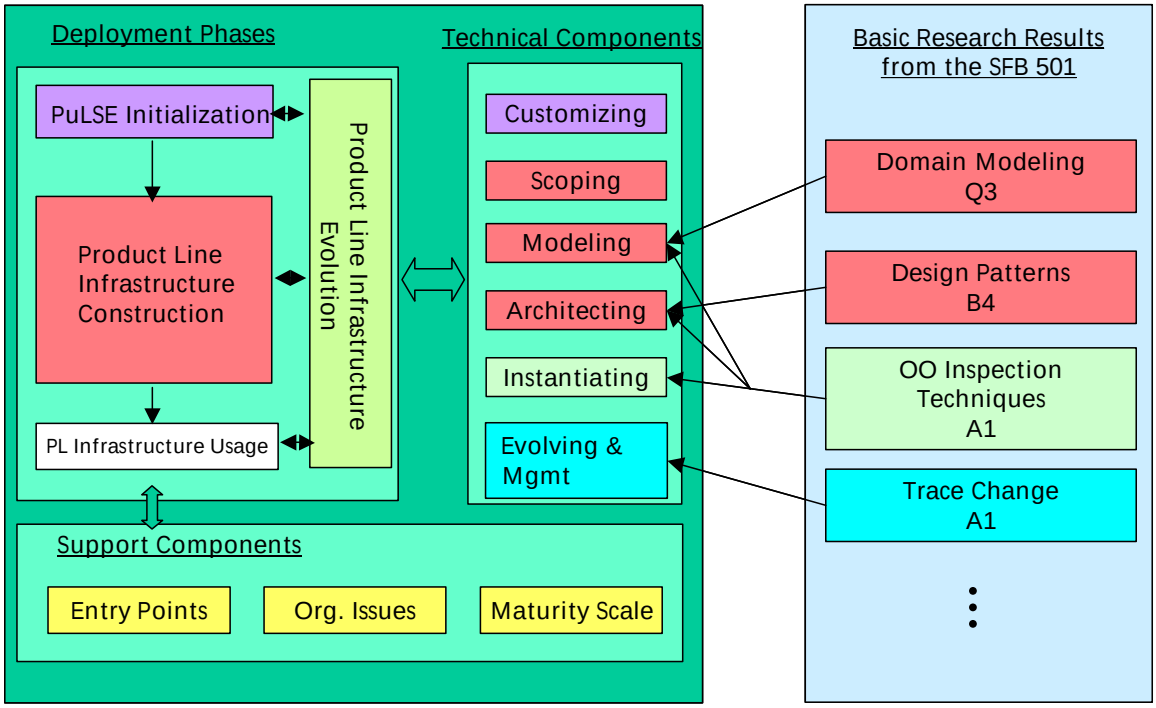

Figure 1. Transfer Approach

This article focuses on the evaluation and industrial transfer of the requirements engineering technique TraceChange [8] for implementing change-oriented requirements traceability. The technique was developed as part of the basic research efforts of the SFB 501 (subproject A1) and evaluated in controlled laboratory experiments. Afterwards, the technique was adapted for industrial use (i.e., supporting requirements recycling in the automotive domain). First experience with the transfer and the industrial use are sketched (indicated in the following as 'application example'). The TraceChange technique is used throughout the article in order to illustrate the application of the transfer approach.
The remainder of this article is structured as follows: Section 2 provides an overview of the benefits and weaknesses of technology evaluation in controlled contexts. Section 3 discusses requirements for adapting techniques originating from basic research for industrial use. In Section 4, qualitative experience regarding technology evaluation in industrial contexts is described. Section 5 sketches ways for packaging technologies for future reuse. Finally, Section 6 gives an outlook on further transfer projects and future work.

2 Technology Evaluation in Controlled Contexts

Experimentation in Software Engineering supports the advancement of the field through an iterative learning process [1]. Experimenting with changes to the current situation enables learning and consequently improving software engineering techniques. Generally, this requires determining the current situation, changing parameters (i.e., values of influence factors), and analyzing the results. Because the context varies with every project, it is hard to identify and control all influencing factors. As a consequence, the evaluation of a new technique is difficult due to the different project contexts. Additionally, a software development technique usually has different effects in different contexts. Results from one controlled experiment can not

be generalized for other contexts. [1] defines experimental scopes by the number of teams involved in an experiment and the number of projects performed during the experiment. 'Teams' are (possibly single-person) groups that work separately, 'projects' are separate problems on which teams work. A m:n setting requires a high amount of control that can only be reached in laboratory settings. Such controlled experiments are mostly focused very narrowly, but they promise a high significance of the results. In the context of the SFB 501 controlled experiments are mainly performed in university environments. A 1:1 relationship between teams and projects characterizes case studies.

Application example. The impact analysis approach TraceChange [8] was developed in the context of the basic research project A1 of the SFB 501. It supports the establishment and management of relationships between documentation entities. The goal of the approach is to perform efficient and precise impact analyses that improve planning and implementing changes. The term precise refers to correctness and completeness of the identified change impacts. Expected benefits of the approach are improved product maintenance and evolution through consistent changes as well as better support for estimating the costs of implementing changes. The underlying principle of the approach is the definition of fine-grained relationships on the level of documentation types with so-called conceptual models and traceability guidelines. Additionally, processes for establishing traces and analyzing impacts of changes are part of the approach. The approach was empirically investigated in controlled environments in the building automation domain. Different controlled experiments were performed in order to (a) identify the difficulties in predicting complete sets of change impacts and motivate the problem to be solved, and (b) verify the expected benefits of the approach. The experimental results showed a significantly beneficial influence of the approach on the correctness and completeness of a predicted set of change impacts in comparison to traditional approaches.

3 Adaptation of Techniques for Industrial Use

The industrial evaluation of basic research results (techniques, tools) requires an adaptation to the specifics of the application domain and the constraints of the industrial partner. This can be done, for instance, by modifying processes and products or porting tools to customer platforms. If necessary, additional preparation or packaging of techniques and tools is required in order to allow for effective and efficient use. This can include the creation of guidelines and user-oriented handbooks. An important requirement for adaptation is the identification of customer needs and the context characteristics of the industrial partners as well as a good understanding of the domain.

Application example. A starting point for the industrial evaluation is the identification of appropriate collaborations with industry partners who could benefit from the technique. A collaboration with a big automotive company was selected for adapting TraceChange for industrial use. The collaboration is part of the research project QUASAR, which aims at integrating techniques for requirements specification and quality assurance within a single, coherent process. In particular, it focuses on software in automotive systems. The situation at the industrial partner can be characterized as follows [7]: Specifications for variants of products (such as electronic control units) are usually documented in separate requirements documents.

Commonalities and differences between product variants are not explicitly used for creating requirements documents. Due to many similarities of the contents between different requirement documents, developers copy parts of existing requirements documents into new requirements documents. This practice has several weaknesses: Besides the difficulty of identifying appropriate requirements for reuse, one major problem are unclear dependencies of the reused requirements. Such dependencies should be appropriately considered. Otherwise, this can lead to severe defects in the new requirements document. It is often not clear whether enough or too many parts of a requirements document are being copied that might have a relationship with the requirements intended to be reused. As described in [7], omissions, inconsistencies and superfluous features are introduced into the new document.

The intention of the automotive company was to improve the current practice through *systematic requirements recycling*, where recycling means the activity of taking requirements from existing requirements documents and inserting them into a new document that describes a similar product. A pragmatic way for adapting the existing process was necessary. Only marginal deviations from the established development process were allowed, the size of the requirements documents should not increase significantly, and the effort for creating requirements documents should not increase significantly. Additionally, there was no time to perform a detailed analysis of the variabilities of the document and create a domain model. Therefore, pure product line techniques (such as domain modeling) were not applicable in this context. An approach based on TraceChange was applied that combines the use of conceptual models to determine relationships necessary for correct recycling with traceability guidelines to solve the problem of omission and inconsistencies through copying an entity with small changes. This can be seen as a first step towards product lines where the conceptual models comprise commonalities of different requirements documents of a domain.

From the viewpoint of transferring TraceChange into industry, several adaptations had to be performed: The traceability goal shifted from improving planning and implementing changes to requirements recycling, the domain changed from building automation to automotive, and the users of the technique are professional developers instead of students. Additionally, the recycling process had to be integrated into a company-specific process. As a consequence, the conceptual models describing the system (CSM) and the documentation (CDM) were adapted for the subdomain electronic control units (ECU). The CSM describes contents of different software artifacts and their relationships for the subdomain. The SDM describes types of documentation entities and their relationships for the subdomain. The effort for creating the CSM and the CDM and the traceability guidelines must be spent only once. The CSM can also be used for similar subdomains. The CDM can easily be adapted for more informal requirements representations.

Both conceptual models were used to create a template that defines a domain-specific document structure. This template reduces the relationships of the documentation entities. An example for such a relationship is a so-called representation relationship between different views of an entity. The reduction of representation relationships helps to omit redundant information and supports the identification of logical entities for reuse. Additionally, both conceptual models were used to derive domain-specific guidelines on how to recycle requirements. The guidelines describe how to document relationships explicitly and how to use these explicit relationships for the correct copying of requirements.

The process for requirements recycling was tailored to the domain and the traceability goal (i.e., requirements recycling). Major steps of the process are:

1. Setting up the document structure according to the template;
2. Searching for recycling candidates in existing requirements documentations;
3. Copying and adapting the recycling candidates with related documentation. Additionally, explicit relationships between entities of the new document have to be described for future recycling of the new requirements document.

The adaptation of TraceChange for industrial use exposed several new insights and improvement opportunities, which might enrich the technique and extend its benefits. For instance, the development of the conceptual models resulted in several opportunities for restructuring requirements documents of the domain. As an example, the restructuring of the documentation allowed for positioning dependant entities close to each other and thus reducing copying effort. The explicit description of relationships supports correct recycling. Furthermore, the reduction of relationships leads to more effective recycling and less defects in the new document. Setting of implicit links (such as name tracing) was additionally introduced in order to reduce the necessity for explicit link setting.

4 Technology Evaluation in Industrial Contexts

Performing case studies is one means for the industrial evaluation of software development techniques in the context of the SFB 501. Case studies can be conducted in realistic contexts (such as industrial environments). A disadvantage is that case studies typically do not provide reliable information for comparisons. A combination of controlled experiments and case studies promises trustworthy results for the effects of software development techniques. Therefore, the determination of the effects of software development techniques in subproject A4 is based on a combination of these two types of experiments. Case studies can be done quantitatively with an accompanying measurement program or qualitatively by observing and collecting qualitative experience (e.g., lessons learned).

Application example. After adapting TraceChance to the industrial context and the specifics of the domain, the evaluation was performed in a scenario-based fashion. The domain experts from the automotive company identified a set of typical changes between two generations of electronic control units, which influence the recycling process. The following classes were identified: Changes in the system environment, architectural changes, changes in functionality, changes of parameters, and changes in conditions. These change scenarios are described in detail in [7]. Based on the change scenarios and the domain knowledge provided by the domain experts, an example requirements document was used to identify recycling candidates. This experience resulted in the identification of typical recycling candidates.
Experience concerning the search of reuse candidates shows that in this context enhanced search algorithms are not necessary because the developers implicitly know very well how to identify recycling candidates. Keyword search is sufficient for this domain. Further experience shows that the traceability guidelines and the documentation structure cannot be separated.
A major risk of the approach is bad process conformance. Successful use of the approach requires disciplined use of the document template and the traceability

guidelines. Bad process conformance may significantly reduce or destroy the benefits of the approach. For instance, a wrong application of the traceability guidelines (such as naming conventions) hinders the identification of relationships between documentation items and consequently leads to incorrect requirements recycling. Organizational measures are needed for assuring high process conformance. An example for such a measure could be the provision of so-called Electronic Process Guides (EPG) that guide the development steps and provide appropriate templates for creating documents. Another measure to improve process conformance could be the performance of careful requirements reviews at the end of selected process steps.

Another risk is that semantic dependencies caused by common context assumptions (e.g., the assumption of a fixed range of a global variable) or design decisions are not considered during recycling of requirements. This can also lead to omissions and inconsistencies and in consequence to wrong system behavior in critical situations. The used conceptual models and traceability guidelines do not cover these semantic dependencies so far. It should be examined whether an extension of the approach in such a way is worthwhile compared to the application of a product line approach.

As an overall experience, the adaptation of TraceChange for requirements recycling in industrial contexts helps the developers to reduce effort because of improved structuring of information. It also helps to avoid omissions and inconsistencies, because it supports copying a documentation entity together with its related entities [7].

5 Packaging of Technologies

The industrial evaluation of basic research results should ideally be a precondition for packaging techniques and tools for reuse. As software development is a human-based development activity and experience in this field is typically context-dependent, the packaging of basic research results (e.g., in an experience base) should include a description of their application scope and the effects in this scope.

In the context of the SFB 501, knowledge is kept in a prototype experience base (SFB-EB). For the SFB-EB an experience model was created [5], which is an adaptation of basic principles of the Experience Factory (EF) approach [2]. The Experience Factory Approach aims at building software competencies and transferring them to projects. Experience elements are regarded as all kinds of software engineering experience, especially models, instances, and qualitative experience [5]:

Models are a means for explicitly describing real-world experiences and building abstractions therefrom. In the context of the SFB-EB, the following models are mainly used: Development activities (such as requirements engineering, design, testing) are represented by process models and complementing guidelines. Software artifacts (such as requirements documents, code) are represented by product models. Formal process modeling notations or template descriptions are used for formalizing process and product models. Quality aspects (such as effort, defect detection effectiveness) are described by characterizing quality models. Relationships between quality aspects (such as the relation between test effort and design complexity) are described by predicting quality models. Functions, diagrams, pie charts, etc. are used as a means for formally describing quality models (e. g., an effort model can consist of a functional relationship between single process elements and empirically derived effort data). The instrumentation of product and process models for the purpose of

applying quality models is represented by GQM models (Goal/Question/Metric), especially Abstraction Sheets and GQM plans [4].

Instances are software artifacts that are consumed, produced, or modified during the development process. An instance refers to exactly one thing in a specific context (in contrast to models that refer to a group of similar things). Some sample instances stored in the SFB-EB are: products, measurement data, technology packages, and process traces. Products (such as requirement documents) are described informally or in the notation of a description technique in the framework of a specific development approach/method/technique (such as UML, NRL, Statecharts). Measurement data (such as effort data) is captured using forms. Technologies and tools are represented in the SFB-EB via so-called technology packages, which support role-specific information necessary for applying the technique/tool (e. g., the NRL technology package supports the requirements engineer, the designer, and the verifier with appropriate information for performing their specific development tasks). Additionally, technology packages help select the appropriate techniques when setting up a new experiment. Process traces are also regarded as instances. Process traces refer to enactments of the project plan and are documented as sequences of project states or lists of plan deviations.

Qualitative experiences are experiences with models and instances that are documented informally during project enactment or after project completion. Documenting experiences qualitatively is especially appropriate for those experiences that are valuable for future projects and not in the focus of GQM goals, i. e., they are not gained from measurement activities and quantitative analyses. Documenting lessons learned is an example for describing qualitative experiences in a structured way as input for subsequent analyses and processing. In the SFB-EB, the following aspects are described: the situation (in which situation did a problem emerge, i. e., what are the project characteristics and the project state?), the symptom (what problem appears?), the diagnosis (what are possible causes for the problem?), the reaction (what are possible solutions for the problem?), the reason (why was a specific solution chosen?), and the result (to which extent could the problem be solved and what were the consequences?). The documentation of the reaction, the reason, and the result is optional depending on whether the problem solution is important for the success of the running experiment, respectively for subsequent replications or similar experiments.

One of the main ideas of the EF approach is that an organization should learn from its own business, not from external, ideal models. This implies that the “domains“ in which a particular experience element may be reusable have to be determined. To follow this idea for each experience element in the SFB-EB, the scope of its validity is described. The scope consists of a context vector and the significance.

The context vector characterizes the environment in which the experience element is valid. It represents characteristics that may determine whether or not an experience element can be reused or shared within or across experiments. A context vector consists of attribute-value-pairs, e. g., <(project effort, 50.000 h), (application domain, building automation), (programming language, JAVA), (programmer experience, high), …>. It should be mentioned that methods for the selection and evaluation of reusable experience elements are highly dependent on the underlying classification scheme.

The *significance* describes how the experience element has been validated and to which extent. Our model for describing the significance of an experience element describes the number of controlled and industrial evaluations in which the element

was involved as well as experimental results concerning the element and their statistical significance (in the case of controlled experiments).
Another opportunity for packaging experience for a broader community is offered by the Virtual Software Engineering Competence Center ViSEK. ViSEK addresses the need for quick and easy access to the latest methods and tools for software engineering. The aim of ViSEK is to establish broad German research competence in software engineering. It tailors existing software technologies for the use in practice. Its main target groups are small to mid-sized companies.

Application example. The controlled experiments with TraceChange are packaged and integrated into the SFB-EDB. The industrial evaluation is described in [7]. It is planned to package the process and corresponding products (such as checklists, guidelines, templates) of TraceChange.

6 Future Work

Future work concentrates on using experience from controlled and industrial evaluations for improving techniques and identifying relevant future research questions. Another issue not performed yet is comparing results from controlled empirical studies in academic contexts with results from industry evaluations. One goal is to use meta analysis techniques for gaining experience that could not be derived from single studies. The intention is to extend the external validity of the results. Finally, by combining controlled evaluations and practice-oriented industry studies, a deeper understanding and confidence in the discovered effects of new techniques is expected.
Several other techniques developed in the SFB 501 are being transferred into industrial practice. One example is an approach for supporting efficient and systematic software evolution through domain analysis. This approach uses legacy documentation in order to create a domain model. Design spaces are used as a notation for the creation of the model. General industrial experience in the domain of planning and management of rosters is described in [6]. Currently, the approach is being adapted for the automotive domain in the context of the EMPRESS project.

Application example. The industrial evaluation of TraceChange demonstrated in a convincing way that TraceChange pays off for the automotive domain. The adaptation of the technique for industrial use and the scenario-oriented evaluation indicated mainly the following directions for future work [8]:

- More fine-grained recycling would be supported through the provision of default values and default sentences in the documentation template.
- Better understanding of the overall purpose of recycling candidates could be achieved through capturing the context and the rational (i.e., the underlying design decision).
- Tools could support the efficiency and effectivity of the approach. Currently, an implementation of the approach based on the COTS tool Telelogic DOORS for requirements management is being performed.
- The approach mainly focuses on functional requirements. Tracing non-functional requirements (such as performance, maintenance) is a major research challenge that could contribute enormously to more effective and correct product evolution, change management and software reuse.

Acknowledgements. This work was partly supported by the DFG in the context of the Sonderforschungsbereich 501 “Development of Large Systems with Generic Methods”. I would like to thank Dr. Antje von Knethen who developed and evaluated the TraceChange approach and Isabel John for their very good cooperation in the project “Experimenting with Techniques for Generic Software Product Lines”. Furthermore, I would like to thank Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) for reviewing the first version of the article.

[1] V. R. Basili, R. Selby and D. Hutchens, Experimentation in Software Engineering, IEEE Transactions on Software Engineering, 1986.

[2] V. R. Basili, G. Caldiera, and D. Rombach. Experience Factory. In John J. Marciniak, editor, Encyclopedia of Software Engineering, volume 1, pages 469–476. John Wiley & Sons, 1994.

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

[4] L. C. Briand, C. M. Differding, and H. D. Rombach. Practical guidelines for measurement-based process improvement. Special issue of International Journal of Software Engineering & Knowledge Engineering, 1997.

[5] R. L. Feldmann, J. Münch, and S. Vorwieger. Towards Goal-Oriented Organizational Learning: Representing and Maintaining Knowledge in an Experience Base. In Proc. of the 10th International Conference o Software Engineering and Knowledge Engineering (SEKE’98), San Francisco, USA, June 18-20, 1998.

[6] I. John, D. Muthig, P. Sody, and E. Tolzmann. Efficient and Systematic Software Evolution through Domain Analysis. In International Workshop on Requirements Engineering for Product Lines (REPL'02), Essen, Germany, September 2002.

[7] A. von Knethen, B. Paech, F. Kiedaisch, F. Houdek, Systematic Requirements Recycling through Abstraction and Traceability, IEEE Joint International Requirements Engineering Conference, 2002.

[8] A. von Knethen, Change-Oriented Requirements Traceability. Support for Evolution of Embedded Systems. PhD Theses in Experimental Software Engineering, Vol. 9, Fraunhofer IRB, 2001.

[9] J. Münch, D. Rombach. Eine Prozessplattform zur erfahrungsbasierten Softwareentwicklung. Jürgen Münch, Dieter Rombach. In Manfred Nagel, Bernhard Westfechtel (Hrsg.): “Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen”,Wiley-VCH, 2003.

[10] D. Rombach. Entwicklung großer Systeme mit generischen Methoden, GI-Jahrestagung, Aachen, September 1997.