Software Architecture at Siemens: The challenges, our approaches, and some open issues

Lothar Borrmann and Frances Newbery Paulisch
Siemens AG, Corporate Technology, Software and Engineering,
D-81730 Munich, Germany.
Lothar.Borrman@mchp.siemens.de, Francis.Paulisch@mchp.siemens.de

Key words: Software architecture, software process, frameworks, architecture assessment, product families, evolution of software architectures

Abstract: The importance of software architecture in the design of large software systems is unquestioned in both the academic and industrial software engineering communities. At Siemens, software is an important, often dominant, factor in the success of our products and this trend towards software is increasing as software becomes even more prevalent in our product spectrum. Our experience indicates clearly that attention to three aspects - to people, to process, and, in particular, to architecture - are important for successful product developments. This paper lists some of the challenges that we face in the area of software architecture, what approaches we have taken as well as a set of issues that require further attention in future.

1. INTRODUCTION

The importance of software architecture in the design of large software systems is unquestioned in both the academic and industrial software engineering communities. In the past few years, there has been a growing body of good literature (e.g., Bass, Clements and Kazman, 1997, Garlan and Shaw, 1996, Jacobson, Griss and Jonsson, 1997, Kruchten, 1995, Perry, 1997) on the topic of software architecture as well as conferences and workshops focusing on this topic. At Siemens, we welcome the efforts, such as this Working IFIP Conference on Software Architecture, to provide a

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35563-4_35

P. Donohoe (ed.), *Software Architecture*

forum for practicing software architects to exchange information with the academic and research community in this area. Practicing software architects are, generally, very busy people because their skills are so important and are in such high demand. We hope that this event will help make it possible for practicing architects to learn as quickly and effectively as possible what techniques are currently available and what techniques will soon be sufficiently mature to be applied.

Siemens is a large, globally-operating, electrical engineering and electronics company with a very diverse range of products. Siemens consists of about a dozen groups that cover the core business areas:

- energy (e.g., power plants)
- industry (e.g., industrial plants)
- communication (e.g., switching systems and mobile phones)
- information (e.g., software products)
- transportation (e.g., control systems for trains and cars)
- health care (e.g., medical imaging equipment)
- components (e.g., ASICs)
- lighting

Global development (e.g., geographically distributed and culturally diverse teams all working on one project) and global sales (e.g., country-specific customization of products) play a significant role in the product development of software, systems, and industrial plants at Siemens. Some of our products are highly customized individual solutions; others are more oriented towards the mass market. Many of our products have strong requirements in the areas of safety, reliability, robustness, and performance and this is further complicated by the fact that this often has to be achieved in real time. Furthermore, the maintainability and serviceability (sometimes over decades!), as well as the evolution of our products is important for our businesses.

Due to the nature of our products, many of the Siemens groups have had a strong orientation towards hardware, electrical engineering, or mechanical engineering. But software is increasingly becoming an important, often dominant, factor in the success of their products and this trend towards software seems to be increasing more rapidly all the time. To give you a feeling for the importance of software at Siemens, consider that:

- More than 50% of our enterprise-wide sales stem from software-based products or systems.
- 27,000 software engineers are employed worldwide (about 10% of our employees).
- Some of our projects are very large, global projects, e.g., one with 2000 developers in 13 countries.

We see software as the key to being able to meet the challenges of flexibility, time-to-market, and reducing costs while maintaining quality.

Because of the importance of software for our company an enterprise-wide "software initiative" was founded in 1995 as part of the "top" initiative (Gonauser, 1997). A dozen groups (e.g., automation, automotive, train transportation, health care, communication, etc.) as well as regional units like Siemens Austria in Vienna and Siemens Switzerland in Zurich actively participate in the enterprise-wide software initiative. One of our main goals of this initiative is to promote an intensive and extensive exchange of information in regards to people, process, and architecture within the various Siemens groups, so that we can learn from each other and improve and help maintain our software expertise. The software initiative focuses on particular topics that are most important to their businesses, e.g., cycle time, cost, quality, process innovation, architecture evolution, measurement-based management, component-oriented development, etc.

The authors have a good overview of the challenges facing our business groups in the areas of software, in particular in the areas of software architecture, software processes, and the human factors involved in both. This paper lists some of the challenges that we face (especially in the area of software and systems architecture), what progress we have made thus far, and what areas we intend to focus on in the future.

2. PEOPLE, PROCESS, ARCHITECTURE

Our experience indicates clearly that attention to all three aspects (people, process, and architecture) are important for successful product developments. Of course, there are many areas where process, people, and software are intertwined, e.g., in our increased focus on component-oriented development, we need to concentrate, not only architectures, but also the processes, and people that support this approach. It is a well-known fact that the capabilities and motivation of the people involved in software development can and does vary widely. Having clearly defined processes and architectures is an important factor both directly and indirectly (via the thereby satisfied customers) in employee satisfaction and motivation.

One of our central research and development departments has a long history (since 1993) of performing process and architecture assessments and associated improvement projects. In the past five years, this group has conducted about 100 process assessments and 10 architecture assessments (Mehner et al., 1998). The main objectives of the assessments are:

- to analyze and evaluate processes and architectures. Architectures are as important as processes for optimizing the triad "costs - quality - schedule" and simultaneously being flexible enough for introducing new and innovative products in the market in time.

- to identify in detail the potential to start and perform improvement projects.
- to specify in detail measurements to improve the evaluated process and architecture and realize this potential.

We place significant emphasis on measurement and evaluation, because our belief is that you can only effectively improve what you can measure. To get optimal insight into the process capabilities of an organization, various interrelated measurement and evaluation techniques have to be applied. In order to drive changes in software and engineering processes, it is necessary to set precisely defined goals and measure progress towards these goals. These goals may be at the project, process, or business level. At Siemens, we have developed and use the so-called top^{Six} controlling instruments (Lebsanft and Rheindt, 1998) for the software-related business (this includes metrics and controlling instruments at the process, project, and management level). These allow us to control the six success factors:

1. customer satisfaction
2. quality
3. cycle time
4. productivity
5. process maturity
6. technology maturity

These final two, process maturity and technology maturity, are where architecture aspects are relevant. As in the capability maturity model (CMM) of the Software Engineering Institute, the process maturity measurement includes a focus on design/architecture issues (e.g., to what extent are architecture reviews performed, how architectures are embedded in the product line management, etc.). The technology maturity indicates the importance of technological trends and how well an organization can adapt to them for business benefit. One aspect of this is related to architecture issues (e.g., the use of COM/DCOM, CORBA, Java, etc.).

Tailored to the needs of the various business groups, the top^{Six} aim to give an objective appraisal of the current state and to allow the early detection of changes. Furthermore, interpreted together, they provide a good understanding of the capability of the organization to develop software. Several of the business groups already have extensive experience with these and other measures and can report on their effectiveness, especially in relation to controlling and improving their development processes.

2.1 Architecture assessments

Due to the increased complexity and size of today's systems, an increased focus on the software architecture of a system is needed. The architecture gives the overall structure of the system and identifies the main components and their interactions. The long-term success of a system depends in large part on the quality of the software architecture. At Siemens, we have developed a method called "system architecture analysis" (SAA) (Gloger et al. 1997) that allows us to evaluate the major technical concepts of an architecture as well as to serve as the basis for proposed improvements for the evolution of the architecture. It includes the determination of the evaluation and weighting of the criteria for the architecture, an analysis of the design decisions, an analysis of the interdependencies between the design decisions, and the evaluation of this information that makes the pros and cons of the various design alternatives clearer (see *Figure 1*).

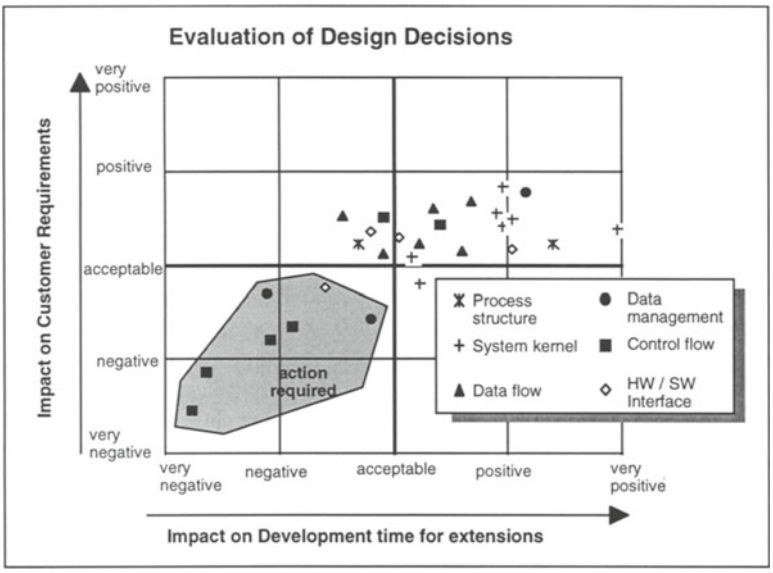

Figure 1: Making design alternatives clearer and indicating where action is required

In comparison to the "Software Architecture Analysis Method" (SAAM) (Kazman et al., 1994), our approach is narrower and more focused on the immediate needs of our business groups.

3. EXPERIENCE AND LESSONS LEARNED

The following section describes some of our experiences and lessons learned in the area of software architecture at Siemens.

3.1 Use innovative processes

We have found that being willing and able to use more innovative processes that, for example, take the purchasing of commercial off-the-shelf software and standard components into account and/or that allow regular and incremental development (e.g., weekly builds) to be effective. Results on the order of 50% reduction in cycle time and 35% reduction in development costs over a period of about five years are not uncommon. An additional advantage is the ability to meet customer requirements and/or react to our competition more quickly. Some organizations are moving from a product-oriented to a process-oriented development approach and the initial results here are very positive, both for the development costs, accuracy at meeting the deadline, as well as for the motivation of the staff.

3.2 Migrate from software "construction" to "composition"

The migration from construction to composition has a strong effect on both the process and the architecture. It is becoming more common to at least consider the integration of existing components (either our own or third-party commercial off-the-shelf (COTS) products). This is considered for various reasons, for example, to focus on our core areas of technical expertise, to reduce costs, to enhance productivity, to adhere to standards, etc. It is, however, very important to be aware both of the potential problems due to architectural mismatch (Garlan, Allen and Ockerbloom, 1995) and the potential process-related challenges associated with the integration of components in a product, for example:

- In order to be flexible and meet the needs of a large number of potential users, the components may be slower and larger than components developed to more closely match the actual needs
- Changes are often impossible (e.g., "black box" components) or difficult (because one has to understand the architecture to change it)
- There are complicated legal and contractual issues, such as liability, associated with this approach.

Typically, we have found that incremental processes work better for composition because they allow the integration of the "foreign" components earlier in the development and because they allow for the early analysis of key performance issues. Not only a good process, but also a clear software

architecture, is necessary for this approach to work well.

3.3 Architecture review sessions are effective

At least one of our business groups regularly holds architecture review sessions in which a real architecture is presented and discussed in detail. This has been shown to have advantages both for those directly involved in the architecture, because they receive valuable suggestions for improvement, and for the other participants who profit from a better understanding of the architecture.

3.4 Frameworks are useful for both process and architecture

We have found that frameworks are a key for achieving an optimal balance between stability and flexibility for both development processes as well as for the software architecture. For example (Völker and Wackerbarth, 1997) cycle time was reduced in the digital switching system area by 50% in the past five years due, to a large extent, on the structure of their development processes. It provides a globally agreed-upon "process framework" giving the stable structure and within the "process components." There is a significant amount of flexibility allowed for use in the specific business groups.

Similarly, we have found a framework approach for software architecture to be effective for software development. In our experience, although there is a significant investment that has to be made in such a framework up front, in the long term the return on investment can be substantial. In three of our business groups we have been able to show that approaches based on components, design patterns (Gamma et al. 1995, Buschmann et al. 1996, Beck et al. 1996), and frameworks have led to significant reductions in cost and faster and more accurate time-to-market (i.e., that the investment in the architecture has started to pay off). In one business group the total software development costs that had been increasing by 30–35% annually have now actually decreased by 10% annually. This reduction is due to their architecture-based approach and their investment in components, design patterns and frameworks, despite similar time constraints and requirements.

As cited in (Buschmann et al., 1998), the decision as to whether a framework approach for software architecture is worthwhile depends on a number of factors, for example, the stability of the subject area and the technology. An in-depth analysis should be done beforehand to decide whether the investment is likely to pay off in the long run. Note also that the

benefit is not just in the reduced development time of the n^{th} framework; the stable framework is likely to be better understood and more tested (i.e., usually of higher quality) than a new development.

The stability and flexibility given by the framework approach is also particularly useful in our global developments since it allows a clear definition of the interactions involving the geographically-distributed people, the processes, and the architecture.

3.5 Investment in domain analysis/product family/product line can be worthwhile

Having a good software architecture is the key to building systems that are scaleable and configurable and thus can be used effectively for a product family or product line. We have had, for example, the case where, for historical reasons, two independent product families had been developed and it became clear that it would make business sense to merge them. We applied our so-called "Harmonization of Software Architectures and Platforms (HAP)" process (Gloger et al., 1998) to harmonize these heterogeneous product spectrums (see *Figure 2*).

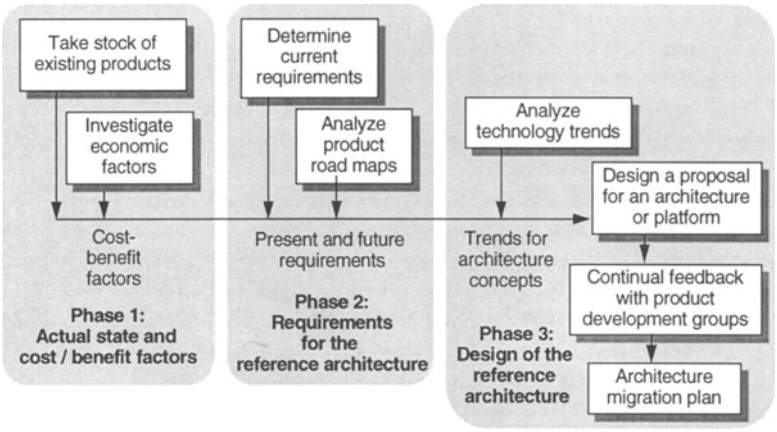

Figure2: The harmonization of software architectures and platforms (HAP) process

The basic principles of this process are to:

- Determine the current state (i.e., what products have harmonization potential) and the cost-benefit factors. This is done in a set of workshops that includes both the development teams and the marketing and sales departments.

- Determine the requirements of the reference architecture. This is a list of requirements together with a set of priorities indicating their significance.
- Define the reference architecture. The aforementioned SAA method (Gloger et al., 1997) is used here to help structure the alternatives and make the decision-making process more transparent. As part of this phase, we also develop a plan for the migration to the new reference architecture.

Note that, although the harmonization approach is of benefit in the product development phase, the benefits further down the line, especially in the logistics (e.g., the installation, commissioning, maintenance, and service phases of the product) are even more dramatic. Configuring, delivering, and installing a dozen different versions of a product is time-consuming and error-prone; architecture harmonization helps reduce the number of alternatives.

From the business perspective it is also important to note that such a harmonization approach can only work if the development processes are also changed accordingly. If multiple units within an organization focus only on their own cost/benefit scenarios, it is very difficult to get them to support such a merged approach. Certainly, attention to the process implications of the product line approach is essential (see also Perry, 1996).

3.6 N-tier architectures are popular, especially for distributed systems

4-tier (user interface, web-top server, application server, database/network) or n-tier architectures are used increasingly especially because the browser-based interface offers increased platform-independence. This approach is particularly popular for large distributed systems and IT systems; for example, the "ComUnity" approach of Siemens Nixdorf Information Systems is based on such a structure. This approach can also be very suitable for mobile devices.

3.7 Maintain an online repository of "best practices"

Several groups have established centers of particular technical competence within their groups and have found this approach to be effective. The software initiative has begun extending this notion by encouraging so-called "best practice networking" in which key staff members serve as "champions" of a particular topic area in our online information repository.

4. PROBLEMS FACED BY SOFTWARE ARCHITECTS IN INDUSTRY

Although section 3 lists some of the areas where our architecture approaches are effective, there are a great deal of unsolved or insufficiently solved problems in the global software architecture community. In this section we list some software architecture problems we encountered in professional software projects within our company. We hope that the discussions at the WICSA1 working conference will help find ways to address these issues.

4.1 Increasing system complexity

With the move from monolithic systems to client-server architectures and n-tier systems, system complexity was reduced by cutting the system into pieces. At the same time, complexity was increased by introducing distribution and heterogeneity.

a) To cope with distribution, communication layers and middleware platforms were added which are not always understood by the average programmer.
b) The interworking of different operating systems, GUIs, database systems, middleware platforms, etc. imposes a number of technical difficulties and requires a combined expertise which is rarely found in a single software architect.
c) "Standard" communication schemes and interfaces are developing rapidly, causing incompatibility issues and necessitating continuous updating of the system components.
d) Programming environments, testing/monitoring tools, and even most conventional programming languages are designed for monolithic systems, but support for distributed environments is still rudimentary.
e) Designing complex systems requires design methods which are sound, but lead to tangible results in a reasonable time. The use of design patterns is one example, but still, architecture design is seen as more of an "art" and the architectural design process is not well-defined.

4.2 Architecture of high-lifetime and rapidly evolving systems

In our company, there exist systems that have a system lifetime of 30 years, like railway control systems. Other systems have a considerable product life time, with continuous development according to technical

progress, like public telephone switching systems. Systems with a high lifetime and/or those that evolve rapidly pose the following problems:

a) Architectural drift is a well-known issue. A properly defined architecture is being continuously degraded by modifications and added functionality. As designers and architects move within the organization, the knowledge about the original architecture fades away, and developers are not capable of preserving the proper architecture when making changes. To improve preservation of architectural knowledge, better ways to document architectures - in the large - are desirable.
b) Once the above has happened, the system with its degraded architecture is considered a "legacy system". It is a common approach to avoid changes to legacy systems, and rather complement them with new components when functionality is to be added. Here we need standard approaches for ensuring interoperability of old systems with new systems. One example is the use of wrappers for legacy systems.
c) A well-designed architecture must support change - it must be stable to allow flexibility of the systems which are built after this architecture. This includes system scalability, in order to be able to provide a family of systems according to user needs, but also the potential to fulfill new requirements, interoperate with other systems, or adapt to technological changes. To achieve this, assessment schemes for architectural quality are required.

4.3 Issues caused by organizational structure

In large companies like Siemens, a clear organizational structure is required for business needs. There is a strong trend to enforce the separation of business units, product lines, etc. While this is a commercial necessity, it can impede the enforcement of an architectural strategy:

a) When developing systems in a vertical structure, each system architecture is typically designed independently. Different approaches may be used and different platforms may be chosen. Interoperability and scalability are in question, and support for various entirely different systems may be required, with high development and maintenance cost incurred. In such a situation, a harmonization of these architectures (as described in section 3) may help, but this is a non-trivial task.
b) To avoid the problem in an early stage, it is recommended to enforce an architecture strategy across organizational boundaries. A way to achieve this is to install an architecture group that works closely with the system designers. In a few cases, we have seen the cooperation of system designers with architects fail. Architects did not have the power or the acceptance to enforce a valid strategy, and were made redundant in the

end. Best practice studies are required to find the correct approach for the empowerment of the architects and the way to cooperate with designers.

c) When the focus of a design and development team is on short-term commercial success, a "quick and dirty" approach may be favored over a properly designed system architecture. While this may lead to a short-term success, it may be costly in the long run. We must find ways to properly count the value of an architecture as an investment. As an example, consider the framework approach: As a rule of thumb, a framework needs to be used three times before the cost of its design and implementation is offset by the reduction in system development cost. Framework development is thus an investment.

4.4 Architectures including COTS components or platforms

It is a common trend in system development projects not to develop all software from scratch, but rather include commercial off-the-shelf (COTS) software components in the system. When we talk about software components, we do use the term in the narrow sense of component-based software; we mean that certain parts of the software system are obtained from commercial software suppliers as standard products. Example of such components include operating systems, database systems, and middleware platforms. In a modern software development environment the portion of COTS software in systems being developed is steadily increasing.

The goal of using COTS software is clear: Development cost and time-to-market is reduced, system functionality is improved, well-proven components are expected to have far fewer programming errors than newly developed code, and the component supplier is expected to take care of component maintenance. Nevertheless, these goals are not always met and we have seen some COTS-based development projects fail; why?

a) Traditional software engineering methods and development processes do not take COTS components into account. A top-down design approach will usually not lead to an architecture which fits the components available, rather a mixture of top-down and bottom-up is required. In the development process, a strong coupling of the requirements engineering stage and the design stage, incorporating rapid prototyping steps, are required, as the properties of the COTS components may strongly influence the properties of the resulting system. Working with COTS components is still more a matter of professional software-engieering experience than something that is well-understood and taught in courses.

b) Software components are more complex than nuts and bolts. Their behavior is—in the best case—only partially documented. The behavior of such a component in a given environment, where it interoperates with custom software and—worse—with other COTS software components, is often unpredictable. Suppliers of COTS components will in most cases give no guarantee that certain non-functional requirements will be met. Moreover, COTS components are not tailored for the specific purpose they are needed for. As a result, they provide unnecessary functionality, at the cost of increased resource consumption and degraded performance. Quality assurance techniques for software components are required.
c) Software development effort is now substituted with different tasks: Selection of components and suppliers, quality assurance and, often, negotiation of licenses. These "new" tasks require different skills and the techniques for implementing them is not widely established yet. When component evaluation and contract negotiation are required, a considerable amount of time may be spent—perhaps even more than the time saved in the reduced development time.
d) When a system is built with a COTS software platform as the base, there is a high risk that the system will be tailored to this platform, which leads to the so-called "vendor lock-in" problem. When this occurs, the system is strongly dependent on the base platform and its supplier. This is a considerable commercial and technological risk. When the platform is not available any more, for whatever reason, a major redesign of the system can be expected. Platform updates may require costly modifications to the overall system. Measures to reduce the risk, like the provision of an isolation layer, are not always appropriate, or may not be chosen, for example, due to the additional overhead and complexity. It is also well known that certain software vendors use this issue to improve their position in the market.

The above shows that the use of COTS components has its price, and that there are quite a few open questions. For some of them, research is under way, but solutions are not widely established yet.

5. OUTLOOK

In addition to the open issues mentioned in section 4, our company is particularly interested in the following issues:

a) What are the interrelationships between product families and process families (e.g., as discussed in Sutton and Osterweil, 1996)?
b) How can we ensure the quality of the individual components?

a) What can we foresee about the quality for software architectures built out of components? In other words, if we use a set of components with particular non-functional properties (e.g., very robust and another very safe) are there any conclusions we can make about the non-functional properties of the whole as opposed to the parts (e.g., as discussed in Clements et al., 1995)?
b) What techniques have proven most effective for the evolution of software architectures? We want to continue to build architectures that are easy to upgrade and that maybe even can be so "clever" as to adapt to particular configurations without human intervention.
c) Software architects are very busy and rightly so, since their knowledge is of great value to the company. How can we make sure that they have enough time not only to work on project-specific issues, but to stay up-to-date with current and future directions.

Within Siemens, both the software initiative and in the projects we have been involved in, we have seen that the exchange of information in these areas can be very beneficial. We hope that by further increasing the interaction with the international software engineering community and sharing some of our experiences that we can provide even more valuable information to our practicing software architects.

ACKNOWLEDGEMENTS

We gratefully acknowledge the discussions and valuable hints provided by Michael Stal and Axel Völker of Siemens that helped improve this paper.

REFERENCES

Buschmann,F; Meunier, R; Rohnert, H.; Sommerlad, P; Stal, M. (1996), *Pattern-oriented Software Architecture - A System of Patterns*, John Wiley.

Beck,K; Coplien, J; Crocker, R; Meszaros, G; Paulisch, F; Vlissides, J. (1996), *Industrial Experience with Design Patterns*, Proceedings of ICSE-18, Berlin, Germany.

Bass, L.; Clements, P.; Kazman, R. (1997), *Software Architecture in Practice*, Addison-Wesley.

Buschmann, F; Geisler, A.; Heimke, T.; Schuderer, C. (1998), *Framework-Based Software Architectures for Process Automation Systems,* 9th IFAC Symposium on Automation in Mining, Mineral, and Metal Processing, Cologne, Germany.

Clements, P; Bass, L.; Kazman, R.; Abowd, G. (1995), *Predicting software quality by architecture-level evaluation*, Proceedings of the 5th Intl. Conference on Software Quality, Austin.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (1995), *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Garlan, D.; Allen, R.; Ockerbloom, J. (1995), *Architectural Mismatch: Why Reuse Is So Hard*, IEEE Software.

Garlan, D.; Shaw, M. (1996), *Software Architecture - Perspectives on an Emerging Discipline*, Prentice-Hall.

Gloger, M.; Jockusch, S.; Weber, N. (1997), *Assessment and Optimization of System Architectures - Experience from Industrial Applications at Siemens*, Proceedings of the European Software Engineering Process Group Conference, Amsterdam, Netherlands.

Gloger, M.; Jockusch, S.; Weber, N. (1998), *Harmonisierung von Software-Architekturen und Plattformen (HAP): Erfahrungen aus dem industriellen Kontext bei Siemens* (in German), Proceedings of the Softwaretechnik '98 Conference, Paderborn, Germany.

Gonauser, M. (1997), *Mit einer Software-Initiative zur Weltspitze* (in German), Computerwoche.

Jacobson, I.; Griss, M.; Jonsson, P. (1997), *Software Reuse: Architecture, Process, and Organization for Business Success*, Addison-Wesley.

Kazman, R.; Bass, L.; Abowd, G.; Webb, G. (1994), *SAAM: A Method for Analyzing the Properties of Software Architectures*, Proceedings of the ICSE-16, Sorrento, Italy.

Kruchten, P. (1995), The 4+1 View Model of Architecture, IEEE Software.

Lebsanft, K.; M. Rheindt (1998), *Improvement of the development to increase customer satisfaction*, Proceedings of the Federation of Software Metrics Associations in Europe (FESMA), Antwerpen, Belgium.

Mehner, T.; Messer, T.; Paul, P.; Paulisch, F.; Schless, P.; Völker, A. (1998), *Siemens Process Assessment and Improvement Approaches - Experiences and Benefits,* Proceedings of the COMPSAC '98 Conference, Vienna, Austria.

Perry, D. (1996), *Product Line Implications for Process*, 10th Intl. Software Process Workshop, Ventron, France.

Perry, D. (1997), *State of the Art in Software Architecture*, Proceedings of the ICSE-19, Boston.

Sutton, S.; Osterweil, L. (1996), *Product Families and Process Families*, 10th Intl. Software Process Workshop, Ventron, France.

Völker, A.; Wackerbarth, G. (1997), *Competence in Software and Engineering: Siemens Software Initiatives*, Proceedings of the European Software Engineering Process Group Conference, Amsterdam, Netherlands.