2005

Going Beyond the Blueprint: Unravelling the Compex Reality of Software Architectures

Kari Smolander
South Carelia Polytechnic, kari.smolander@scp.fi

Matti Rossi
Helsinki School of Economics and Business Administration, mrossi@hkkk.fi

Sandeep Purao
Pennsylvania State University - Abington, spurao@ist.psu.edu

GOING BEYOND THE BLUEPRINT: UNRAVELING THE COMPLEX REALITY OF SOFTWARE ARCHITECTURES

Kari Smolander, South Carelia Polytechnic, Koulukatu 5B, FIN-55120 Imatra, Finland, kari.smolander@scp.fi

Matti Rossi, Helsinki School of Economics, P.O. Box 1210, FIN-00101 Helsinki, Finland, mrossi@hkkk.fi

Sandeep Purao, School of Information Sciences and Technology, The Pennsylvania State University, University Park, State College, PA 16802, spurao@ist.psu.edu

Abstract

The term Software Architecture captures a complex amalgam of representations and uses, real and figurative, that is rendered and utilized by different stakeholders throughout the software development process. Current approaches to documenting Software Architecture, in contrast, rely on the notion of a blueprint that may not be sufficient to capture this multi-faceted concept. We argue that it might not even be feasible in practice to have such a unified understanding of this concept for a given setting. We demonstrate, with the help of in-depth case studies, that four key metaphors govern the creation and use of software architecture by different communities: "blueprint", "literature", "language", and "decision". The results challenge the current, somewhat narrow, understanding of the concept of software architecture that focuses on description languages, suggesting new directions for more effective representation and use of software architecture in practice.

Keywords: software architecture, systems development, grounded theory, metaphors

1 INTRODUCTION

Modern information systems are increasingly built on top of complex underlying communication, computer and software infrastructures that must communicate across other systems, devices, and group boundaries. This level of interconnectedness poses significant challenges to the communities involved in their development and continued use. The state-of-the-practice for documenting these architectures is dominated by software architects and engineers, who rely heavily on technical representations (Bosch et al., 2002). Much of the research in this domain, therefore, focuses on building novel or elegant representation schemes (Bass et al., 1998, Garlan and Kompanek, 2000, Hevner and Mills, 1993, Medvidovic and Taylor, 2000). While useful in some respects, this research tends to negate the important roles that other communities play in the development and use of software architecture during the software development process (Bosch, 2000, Smolander and Päivärinta, 2002).

In practice, the idea of 'software architecture' is used, interpreted and communicated by a diverse set of stakeholders with varying skills and experience, including for instance, data administration departments, production organizations, external customers, hardware vendors, salespersons, and management. An understanding of the 'software architecture,' thus, allows them to communicate about the system, during construction or after deployment, during its continued use. The complexity of software architecture, its fuzzy nature, and varying manners of use, therefore, demand approaches that must go beyond current efforts at formalizing and devising special-purpose description languages for its representation (Allen and Garlan, 1997, Medvidovic and Taylor, 2000).

The objective of this research is to discover meanings that different communities of interest ascribe to the concept of Software Architecture during its creation and use. Drawing on an in-depth study of three organizations, we analyze the perceptions of different stakeholders (architects, software designers, managers, and customers) to uncover metaphors that govern the understanding of this complex yet fuzzy concept in practice. Our results suggest that 'software architecture' can be better understood with the help of multiple metaphors instead of the dominant one of 'Blueprint' in use today. The results are important because they provide a first account of how developers perceive the concept of software architecture during its creation and use.

2 METAPHORS

An important idea underlying our research process is that of 'Metaphors.' In the domain of information system design, Madsen introduced the idea (1994), suggesting that a metaphorical instead of classification approach may be better suited to software design. Metaphors reify linguistic and cognitive crutches that individuals and communities use to make sense of the complexities around them. In modern literature, the key proponents of Metaphors are Lakoff and Johnson (1980), who describe them as the underlying layer of the structure of our conceptual systems that we use to understand abstract or complex concepts we encounter. The idea of 'software architecture' clearly qualifies as such a complex concept. In fact, the term 'software architecture' itself represents a metaphor, where we attempt to understand the structure of virtual systems using a term – architecture – borrowed from physical structures. A metaphor, therefore, does not suggest different views or aspects of the underlying complex phenomenon. Instead, each suggests an image or an allegory that the participants use to make sense of a complex concept. The analysis we report, thus, attempts to unravel the complex reality of software architecture by uncovering different metaphorical forms that are employed by the communities engaged in the creation and use of software architecture in practice.

2.1 Research Method

An exploratory, qualitative, and theory-forming strategy following a *grounded theory* approach (Glaser and Strauss, 1967, Strauss and Corbin, 1990) was considered appropriate for this research. For phenomena that lack scientifically established theories and concepts, such an approach is recommended because it strongly grounds the generated theory to empirical data, making it a valid choice for software engineering research (Seaman, 1999).

To understand how different communities engage in the creation of software architecture in practice, we carried out a study of three software-producing organizations over a period of one year tracking the different communities involved in the software architecture development process with a particular focus on understanding how these communities generated, represented, used and shared knowledge regarding software architectures. Our research methodology was immersive, and used interviews as the primary mode of data gathering.

The study was conducted in three Finnish software-producing organizations (Table 1), selected because of the differences in their products, organizations, and business strategies. All the organizations were considered fairly advanced in their use of state-of-the-art techniques and methods for software development practices. For instance, they make extensive use of UML during design, and Java and components during implementation. The organizations represent different types of firms from the software industry – a service developer of an operator, a software developer for telecom devices, and a developer of tailored information systems.

Company	Business	Employees*	Interviewees
Telecom service developer	Development of software-based telecom services and platforms for in-house customer	200	2 architects, 2 designers, 3 managers
Handheld software producer	Software and software tool development for mobile terminals and hand-held devices	200	1 architect, 1 designer, 4 managers
IT solution provider	Development of tailored information systems for projects dictated by customers.	400 in a division, 600 in the other	3 architects, 1 designer, 2 managers

* = Number engaged in software development

Table 1 Target organizations and Interviewees

Grounded theory development is a research method developed originally in social sciences, which uses qualitative content analysis for the construction of a theory grounded in the data about the phenomenon under study. Grounded theory approaches have proved their usefulness when dealing with new and unexplored areas related to processes and change in organizations. The role and meaning of architecture in systems development projects exemplifies such an area that lacks scientifically established theories and concepts. Because theory creation following this approach is strongly grounded to the data (instead of researcher's inspiration), the resulting theory is more credible and the research tends to produce useful and practically valid results (Orlikowski, 1993). Information systems research offers many examples of the application of grounded theory (for instance, Calloway and Ariav, 1991, Orlikowski, 1993), and the field of software engineering also recognizes the need for qualitative approaches in the areas related to human behavior (Seaman, 1999).

The research process proceeded in two broad phases: pre-study, and data collection and analysis (Figure 1). The status of architectural practices in the three organizations was resolved by using both questionnaires and prepared presentations by the chief architects working in these organizations. The purpose of this phase was to achieve background information and to serve as the basis for interpretations.

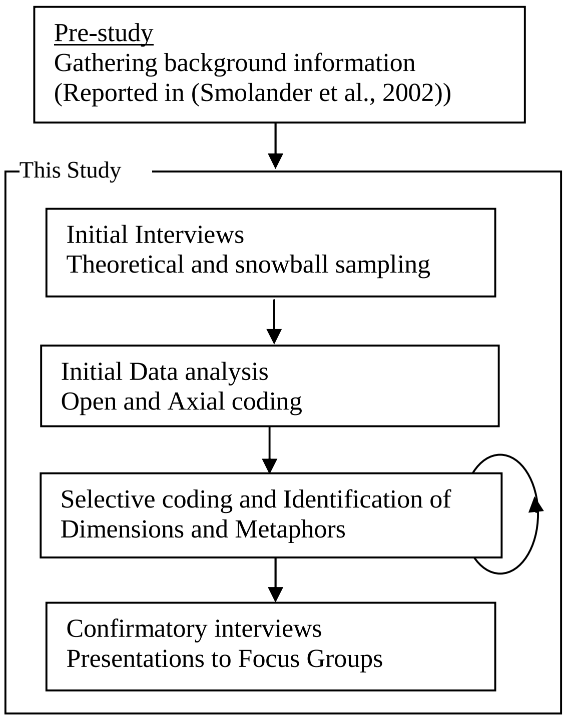
Pre-study
Gathering background information
(Reported in (Smolander et al., 2002))
This Study
Initial Interviews
Theoretical and snowball sampling
Initial Data analysis
Open and Axial coding
Selective coding and Identification of
Dimensions and Metaphors
Confirmatory interviews
Presentations to Focus Groups

Figure 1. The Research Process

The study reported in this paper gathered the data by conducting interviews in each organization using a theoretical sampling strategy. Based on the results of the first set of interviews in the first organization, which included two designers, two architects, one project manager, and one department manager; it was decided that the role of the internal customer should be added to potential interviewees. Glaser and Strauss (1967) call this dynamic process of data collection where the sample is extended and focused according to emerging needs as theoretical sampling. A total of 19 interviews were conducted, interspersed with transcription and open coding, which allowed this extension. The open coding proceeded in parallel, treating each interview as confirmation or further development of results from earlier interviews. These additional interviews included three more architects, three designers, and six managers, of which two were team leaders, two technology managers, one site manager, and one process development manager.

Open coding (Strauss and Corbin, 1990) of the interviews was done using ATLAS.ti (Scientific Software, 2001). Figure 2 shows an example. The open coding process started with the high-level "seed categories" (Miles and Huberman, 1984) such as stakeholders, problems, and rationale for architecture description (as in Figure 2). Additional categories were created and existing ones were merged as new evidence and interpretations emerged. The open coding of all the 19 interviews in the three organizations produced altogether 179 different categories, which were organized, following a process of axial coding, into eight category families named “communication”, “general features”, “problems”, “rationales”, “solutions”, “stakeholders”, “tools”, and “viewpoints,” each of which included 4 to 36 specific categories. The categories and their families were then used to identify the metaphors via an inductive analysis (Smolander, 2002). Each represented a different combination of values along dimensions describing the categories. Identification of the metaphors then led to the discovery of additional dimensions (by returning to the data) that characterized each metaphor. Table 2 shows these dimensions, indicating with italics, the dimensions added after identifying the metaphors.

Dimension	**Explanation**
Time orientation	descriptions of past architectural solutions versus current design situation versus prescriptions about future implementations
Formality	descriptions for enabling understanding versus those meant for generating executables
Detail	descriptions of technical details or descriptions that purposefully constrain the level of detail
Activity	nature of typical activities associated with the descriptions such as recording versus negotiating versus sense-making
Objective	the objective of architectural design and description
Customer orientation	frequency and strength of interaction between the development organization and customers utilizing the software architecture
Business orientation	extent of reasoning the development group must make about the business area of the system
Stakeholder diversity	the number of roles with multiple backgrounds, organizations, and professions occurring in software development projects

Table 2. Dimensions Discovered

Finally, the selective coding phase led to the emergence of metaphors, which captured the varied meanings of architecture in practice. The basic idea of the four metaphors, as described below, emerged as an idea during the early analysis (cf. Glaser, 1978), but to validate and refine this idea required many iterations. An iterative strategy was adopted during that analysis, interpreting the coded text, through a scheme of dimensions that were found in the instantiations of the core category “meaning of architecture in practice.”

3 FOUR METAPHORICAL FORMS FOR UNDERSTANDING SOFTWARE ARCHITECTURE

Four distinct metaphors emerged from this inductive analysis. These were: *Blueprint* – architecture as the structure of the system to be implemented, *Decision* – architecture as the decision and basis for decisions about the system to be implemented, *Language* – architecture as language for achieving common understanding, and *Literature* – architecture as documentation and frames of references for readers.

3.1 Architecture as Blueprint

As expected, the first metaphor that emerged from the analysis was the traditional, ‘Blueprint.’ Within this metaphor, architecture is considered as a high-level description of the system, directly guiding more detailed implementation aimed at the production of individual components. The complete specification of architecture then resides in the working implementation of the system. In the current software architecture research, this metaphor can be associated with the architecture description languages. Clearly, it is oriented towards the future. A typical activity associated with this metaphor is implementation of software artifacts and their interconnections within the system, which necessitate both high formality and high level of detail in the descriptions. Of the interviewees in our study, software designers and some of the technically oriented architects emphasized this metaphor. In particular, it was strongest among individuals involved in implementing or programming the system according to specifications.

“Our development is organized so that we first describe the architecture and from that comes the DLL descriptions and then possibly different persons make the individual DLL’s. In a way it [the architecture description] is the basis for the next phase, which is the DLL design.” - Jack, Software Engineer

3.2 Architecture as Decision

The second metaphor, ‘Decision,’ emphasizes the role of architecture as specific decisions about the system structure. The software architecture decision can have considerable impact on resources needed for building the system, including work force needs, special skills and resources that must be spent on requirements e.g. third party licenses. The decisions also provide concrete resolutions to trade-offs and resolve conflicting requirements such as usability, maintainability, and performance. Users consider too much formality and detail harmful for this metaphor to be useful. For some technical trade-off situations, however, high formality and detail may be needed. Among the participants we interviewed, stakeholders such as managers and resource planners (like project managers) emphasized this metaphor, which they also saw as the basis for division of work between working units.

“The customer is now wondering about the ambiguity of the situation. One architectural choice looked technically quite good but its price could rise so high that they must think about business premises. If it costs 10 millions then how much it must have usage so that it pays the price back in a reasonable time.” - Arthur, Architect

3.3 Architecture as Language

The third metaphor, ‘Language,’ sees architecture as the enabler of a common understanding about the system structure. According to this metaphor, software architecture serves as the vehicle for communication between different stakeholders about high-level structures and solutions. The metaphor, therefore, emphasizes understanding between different stakeholders. For this metaphor as well, too much formality or detail is considered harmful because it aims at achieving understanding among a diverse set of stakeholders, who are likely to possess varying backgrounds and experiences. In our study, those with high customer or business orientation, such as managers emphasized the need for this metaphor. Other situations when this metaphor

“From my point of view the purpose of architecture description is that you know where you are going.” - Harry, Project Manager
“The purpose of architecture description is that you must be able to tell to the customer and to the team what is your idea.” - John, Architect

assumed high importance were, when the customer participation was intense, marketing was closely involved in the process, or when external data administration departments had strong interests. Clearly, increased diversity among stakeholders meant a greater emphasis on this metaphor.

3.4 Architecture as Literature

The final metaphor that emerged from our analysis was 'Literature.' This metaphor is closely related to the idea of documentation that aids in transferring information among individuals who assume similar roles over time. Following this metaphor, architecture is seen as the documentation of the solution or the collection of solutions made in the past. Clearly, this metaphor is oriented towards past decisions, and aims at transferring explicit knowledge and understanding about technical artifacts to future generations such as maintainers. Typical activities associated with this metaphor include writing to create the literature and reading / analyzing to understand the literature. This also dictates the level of detail, which tends to be high though with a varying degree of formality. Of the participants in our study, software designers tended to emphasize this metaphor; however, many other stakeholder groups also suggested that they used this metaphor.

"I think the purpose of architecture descriptions is to keep the knowledge of what kind of a system we have. Many times the environments are heterogeneous and their parts are here and there. It is nice to have such a document or a part of a document that you can take a view and see what is the scope here. In addition, those who will possibly make further development or maintenance can learn the system easily with it."
- Michael, Software Engineer

3.5 Summary of metaphors

The brief description above lays out metaphors used by the research participants to make sense of the complex concept of software architecture in practice. Table 3 summarizes how the metaphors and dimensions identified in Table 2 relate each other. The table shows in a condensed form the differences between the metaphors. Clearly, these metaphors can overlap and may be used simultaneously by multiple stakeholders. A stakeholder may even use multiple metaphors, and the emphasis may vary across stakeholders and over time. For example, the role of a manager in most software producing organizations would necessitate use of the metaphors of blueprint and decision. Next we discuss the implications of these findings for practice and research.

Dimension	**Metaphor**			
	Blueprint	**Decision**	**Language**	**Literature**
Time orientation	Future	Future	Present/Future	Past
Formality	High	Usually low	Low	Varies
Detail	High	Usually low	Low	Usually high
Activity	Implementation of software artefacts	Evaluating alternatives, making choices	Communicating about structures and technologies	Reading and analysing
Objective	High-level description of the system guiding implementation	Making decisions concerning resources and strategies	Understanding system structures and technologies	Documenting the system, under-standing over time
Customer orientation	Low	High	Possibly high	Usually low
Business orientation	Low	High	Possibly high	Usually low
Stakeholder diversity	Low	High	High	Usually low

Table 3. The four metaphors and their dimension values

4 DISCUSSION

The most salient outcome of our research are the metaphors, which provide a plausible explanation for why the software architecture community has been unable to provide a commonly agreed upon and useful definition of software architecture (Baragry and Reed, 2001). Instead, most research on software architecture has relied on using practical experiences and insight presented as tentative ad-hoc lists or seen from the perspective of a software engineer (Hofmeister et al., 1999). The definitions have, therefore, been either too general to be useful in practice or too exact but limiting, emphasizing only some aspects of architecture e.g. within a certain methodical framework (Garlan et al., 2002). Studies dealing explicitly with different stakeholders and their informational needs have treated these stakeholders as passive consumers of the concept of software architecture instead of active participants dictating its creation and use. This is in sharp contract with our study, and explicit requirements in IEEE standardization efforts (IEEE, 2000). The results we have described argue that it important to allow decision makers, users and managers the requisite tools that they can use to discuss and negotiate the software architecture.

Another perspective provided by our analysis is that the use of software architecture varies according to stakeholder, situation, and phase of the project. At the beginning of a project, the 'language' and 'decision' metaphors prevail as different internal and external stakeholders interact to achieve a common vision of the goals. As the project proceeds towards implementation, technically oriented stakeholders, such as architects, designers, and programmers get more involved in the process and the emphasis shifts to the 'blueprint' metaphor. As the system is deployed, the 'literature' metaphor gains emphasis, because new participants must operate or maintain the system, which requires understanding its structure and principles through documentation. In spite of this broad progression, though, all metaphors appear to endure through the development process as the system and its structure must be constantly communicated and understood, new decisions must be made, new changes and extensions must be designed, and documented to enable learning. Because of the involvement of multiple stakeholders in its creation and use, the overarching perspective that allows us to further understand our discovery of metaphors is one of "negotiation" and "interaction" across communities of practice. Figure 2 captures this perspective.

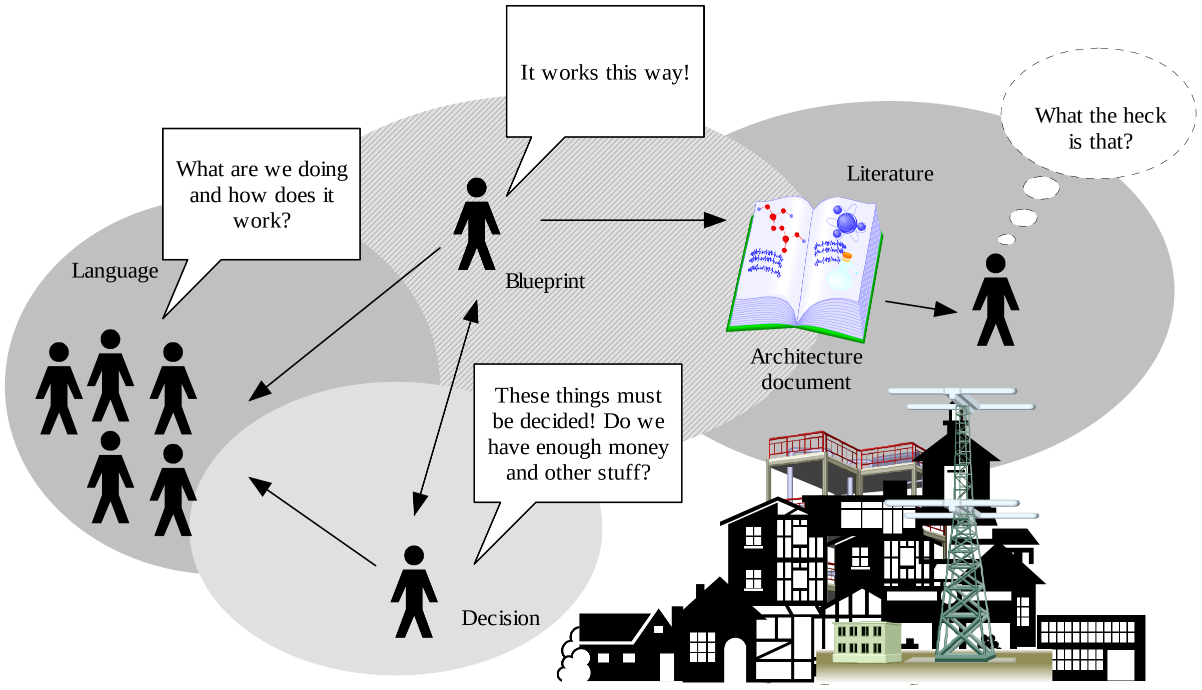
It works this way!
What are we doing and how does it work?
What the heck is that?
Literature
Language
Blueprint
Architecture document
These things must be decided! Do we have enough money and other stuff?
Decision

Figure 2. *Making Sense of Software Architecture*

As the progression suggests, the concept of software architecture usually forms a continuum from vague and noble ideals into stringent decisions about technical platforms and data interchange formats. As the concept becomes more concrete, stakeholders become aware of the consequences, conflicts, and problems they will face. This leads to two distinct requirements for architectural descriptions. First, they should allow refinement to a concrete level (both politically and technically) soon after the project initiation, and second, the descriptions (and their implications) should be approachable and intelligible by the stakeholders engaged in the process. The approaches used to describe architecture, such as the 4+1 architectural view (Kruchten, 1995), instead, represent a forced compromise, allow minimal variations during the development process, and focus primarily on technological requirements. The immediate and obvious implication of the four metaphors, therefore, is that there are many perspectives of architecture to be described and many experts with different skills and vocabularies that use the architecture, suggesting a need for informal and expressive approaches. Further, these approaches should allow maintenance of multiple and even conflicting views of the architecture simultaneously.

A clear research opportunity that we identify is to create representational forms that can satisfy many of these needs. Much existing research aims at creating precise architecture definition languages, whereas practice needs architecture representations, which allow the co-existence of several, maybe mutually incompatible, views of architecture. The participants in the creation and use of software architecture must engage in a dialogue with others, which requires visual representations. The metaphors we have discovered provide clear directions about properties needed from architecture development methods and representations, which can allow all groups to participate in the development process. Unlike representation forms for software development e.g. data flow diagrams and UML, where communication with users is considered important and has been the topic of research, this need appears to have been dismissed altogether for architectural description languages. The analysis we have reported essentially opens this direction for research. In a related vein, our analysis suggests that the dominant view about software architecture – which corresponds to the 'Blueprint' metaphor – may be too narrow, and may not be sufficient to resolve architectural problems in practice.

References

Allen, R. and D. Garlan (1997) "A Formal Basis for Architectural Connection," ACM Transactions on Software Engineering and Methodology (6) 3, pp. 213-49.

Baragry, J. and K. Reed. (2001) "Why We Need A Different View of Software Architecture." Proceedings of the Working IFIP/IEEE Conference on Software Architecture (WICSA 2001), Amsterdam, The Netherlands, 2001, pp. 125-134.

Bass, L., P. Clements, and R. Kazman (1998) Software Architecture in Practice: Addison-Wesley.

Bosch, J. (2000) Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach: Addison-Wesley.

Bosch, J., M. Gentleman, C. Hofmeister, and J. Kuusela (eds.) (2002) Software Architecture: System Design, Development and Maintenance - IFIP 17th World Computer Congress - TC2 Stream / 3rd Working IEEE/IFIP Conference on Software Architecture (WICSA3), Aug 25-30 2002, Montréal, Quebec, Canada: Kluwer Academic Publishers.

Calloway, L. J. and G. Ariav (1991) "Developing and Using a Qualitative Method to Study Relationships among Designers and Tools," in H. E. Nissen, H. K. Klein, and R. Hirschheim (Eds.) Information Systems Research: Contemporary Approaches and Emergent Traditions: North-Holland, pp. 175-193.

Garlan, D., S.-W. Cheng, and A. J. Kompanek (2002) "Reconciling the Needs of Architectural Description with Object-Modeling Notations," Science of Computer Programming (44) 1, pp. 23-49.

Garlan, D. and A. J. Kompanek. (2000) "Reconciling The Needs of Architectural Description with Object-Modeling Notations." Proceedings of the Third International Conference on the Unified Modeling Language - UML 2000, York, UK, 2000, pp. 498-512.

Glaser, B. (1978) Theoretical Sensitivity: Advances in the Methodology of Grounded Theory. Mill Valley: Sociology Press.

Glaser, B. and A. L. Strauss (1967) The Discovery of Grounded Theory: Strategies for Qualitative Research. Chigago: Aldine.

Hevner, A. R. and H. D. Mills (1993) "Box-structured methods for systems development with objects," IBM Systems Journal (32) 2, pp. 232-251.

Hofmeister, C., R. Nord, and D. Soni (1999) Applied Software Architecture. Reading, MA: Addison-Wesley.

IEEE. (2000) IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE IEEE Std 1471-2000.

Kruchten, P. B. (1995) "The 4+1 View Model of Architecture," IEEE Software (12) 6, pp. 42-50.

Lakoff, G. and M. Johnson (1980) Metaphors We Live by. Chigago: The University of Chigago Press.

Madsen, K. H. (1994) "A guide to metaphorical design," Communications of the ACM (37) 12, pp. 57-62.

Medvidovic, N. and R. N. Taylor (2000) "A Classification and Comparison Framework for Software Architecture Description Languages," IEEE Transactions on Software Engineering (26) 1, pp. 70-93.

Miles, M. B. and A. M. Huberman (1984) Qualitative Data Analysis: A Sourcebook of New Methods. Beverly Hills: Sage.

Orlikowski, W. J. (1993) "CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development," MIS Quarterly (17) 3.

Scientific Software (2001) "ATLAS.ti - The Knowledge Workbench," http://www.atlasti.de/ (28 Mar, 2002).

Seaman, C. B. (1999) "Qualitative Methods in Empirical Studies of Software Engineering," IEEE Transactions on Software Engineering (25) 4, pp. 557-572.

Smolander, K. (2002) “Four Metaphors of Architecture in Software Organizations: Finding out The Meaning of Architecture in Practice.” International Symposium on Empirical Software Engineering (ISESE 2002), Nara, Japan, October 03 - 04, 2002, 2002, pp. 211-221.

Smolander, K., K. Hoikka, J. Isokallio, M. Kataikko, and T. Mäkelä. (2002) “What is Included in Software Architecture? A Case Study in Three Software Organizations.” Proceedings of 9th annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS), 8-11 April 2002, Lund, Sweden, 2002, pp. 131-138.

Smolander, K. and T. Päivärinta. (2002) “Describing and Communicating Software Architecture in Practice: Observations on Stakeholders and Rationale.” Proceedings of CAiSE'02 - The Fourteenth International Conference on Advanced Information Systems Engineering, Toronto, Canada, May 27 - 31, 2002,, 2002, pp. 117-133. Lecture Notes in Computer Science.

Strauss, A. L. and J. Corbin (1990) Basics of Qualitative Research: Grounded Theory Procedures and Applications. Newbury Park, CA: Sage Publications.