European Journal of Information Systems (2008) 17, 575–588

www.palgrave-journals.com/ejis/

Software architectures: Blueprint, Literature, Language or Decision?

Kari Smolander[1], Matti Rossi[2] and Sandeep Purao[3]

[1]*Department of Information Technology, Lappeenranta University of Technology, Lappeenranta, Finland;* [2]*Department of Business Technology, Helsinki School of Economics, Finland;* [3]*College of Information Sciences and Technology, Penn State University University Park, PA, U.S.A.*

Correspondence: Matti Rossi, Department of Business Technology, Helsinki School of Economics, PO Box 1210, Helsinki FI-00101, Finland.
Tel: +358-9-43138996,
Fax: +358-9-43138700
E-mail: Matti.Rossi@hse.fi

Received: 6 June 2008
Revised: 16 September 2008
2nd Revision: 29 September 2008
Accepted: 30 September 2008

Abstract
This study questions the traditional view of software architecture as a specification that needs only be understood by software architects and engineers. Based on an intensive study of three software-producing organizations, we identify multiple metaphors ('Blueprint,' 'Literature,' 'Language' and 'Decision') that stakeholders use to understand the term *software architecture*, which in turn, allows them to effectively participate in its creation and use. Our results point to new research directions that may better encompass a broader view of software architecture.
European Journal of Information Systems (2008) **17,** 575–588.
doi:10.1057/ejis.2008.48

Keywords: software architecture; stakeholders; metaphors

Introduction

An explicit understanding of the underlying architecture is a prerequisite for the design, evolution and maintenance of modern information systems that must complement today's complex business processes spread across internal divisions and external partners. With technologies such as enterprise application integration (Linthicum, 2000), service-oriented computing (Huhns & Singh, 2005), components for enterprise systems (Fan *et al.*, 2000) and service-oriented architectures (Erl, 2004), which require significant cross-organizational collaboration and imply separating interfaces from implementations, an understanding of architecture is assuming an even more critical role. An important corollary to this requirement is that this understanding should not be restricted only to software developers, but must also be accessible to other stakeholders such as users, customers and managers (Smolander & Päivärinta, 2002a). This is especially important when we are moving to the era of utility, or service computing, which relies on external service architectures (e.g., Google Application Programmable Interfaces (APIs), trust and payment APIs) for service composition (Huhns & Singh, 2005). In this new world all developers, managers and other stakeholders must be able to understand and use architectures (Rozanski & Woods, 2007).

Although recent articles talk about the maturation (Kruchten *et al.*, 2006; Taylor & van der Hoek, 2007) and even a 'golden age' (Shaw & Clements, 2006) of software architecture, approaches to representing, designing and communicating software architectures continue to be narrowly focused, with an emphasis on software architects and engineers as the key stakeholders, and continue to employ highly technical representations (Bosch, 2000). Research in this domain (e.g., Dashofy *et al.*, 2005) has therefore continued to build representation schemes for documenting software architectures (Garlan, 2000; IEEE, 2000; Clements *et al.*, 2002) as technical blueprints in spite of calls for broadening the architecture

perspective to non-technical uses (Smolander & Päivärinta, 2002b; Medvidovic *et al*., 2007; Rozanski & Woods, 2007). This focus tends to downplay and discourage the necessary and often important roles (Carlile, 2002) that different stakeholders play in the creation and use of software architecture (Grinter, 1999; Smolander & Päivärinta, 2002b; Rozanski & Woods, 2007).

Ongoing research elsewhere (Medvidovic *et al*., 2007) outlines multiple perspectives or concerns (Rozanski & Woods, 2007) that are necessary to facilitate participation from different stakeholders in the creation and use of software architecture. Empirical studies to back the arguments made in these works are, however, not available. As a result, Medvidovic *et al*. (2007) multi-perspective view on software architecture languages reflects an *internally focused* argument that is derived from the slow adoption of research contribution to industrial practice. An example of such research contributions to the field is architecture description languages (ADLs) (Medvidovic & Taylor, 2000) that were widely researched, but rarely adopted in practice.

The study we describe in this paper provides an important complement. We describe the results of an intensive study of three software-producing organizations over a period of 1 year, tracking the process of software architecture development with a particular focus on understanding how different stakeholders generated, represented, used and shared knowledge regarding software architectures. The broad research question that drove this investigation was the following: *How is software architecture developed and used in an organization?*

Our results provide an *empirically derived* argument that complements the one provided by Medvidovic *et al*. (2007). To arrive at our research outcomes, we followed an immersive research methodology in the tradition of Curtis *et al*. (1988), characterized as grounded theory development (Strauss & Corbin, 1990). The primary mode of data gathering was interviews with organizational actors that belonged to different stakeholder groups.

Content analysis of the data gathered (19 interviews, 12 h, 310 pages of transcribed pages) was performed as a part of grounded theory building (Strauss & Corbin, 1990). In this paper, we present and discuss outcomes of this analysis as different 'metaphors' that exemplify how different stakeholder groups participate in the creation and use of software architecture. A key contribution of our research, therefore, is providing an empirical grounding for a multi-faceted understanding of the concept of software architecture. Together with the recent work by Medvidovic *et al*. (2007) and Rozanski & Woods (2007), findings from this study call for intensified efforts to develop architectural languages and methods that allow more varied representations that may be accessible to different stakeholder groups and more effective communication across these groups.

In what follows, we first summarize prior research on software architectures with a view to showing their current dominant focus on structural/technical specifications, and highlighting the need for expanding this understanding to encompass additional dimensions (mirroring the arguments made by Medvidovic *et al*. (2007)). The next section outlines the research method and describes the setting for the current study. In the subsequent section, we describe the results as *metaphors* that exemplify the stakeholders' perspectives of the concept of software architecture and extend the analysis to describe how this understanding can tell us how each stakeholder group may participate in its creation and use. The final section ties the results back to current research on software architecture representations, juxtaposing results from our empirical work against recent evolution of ADLs to derive implications for future research and practice.

Prior research

This section reviews prior research related to ADLs contrasting it against findings from research related to processes in the creation and use of software architecture.

Software architecture

The commonly accepted understanding of the term 'software architecture' among researchers is that of a structure that describes the system in terms of its components, their basic operational principles and their interconnections (e.g., IEEE, 2000; Bass *et al*., 2003; Kruchten *et al*., 2006). In many cases the term 'architecture' implicitly includes physical elements such as hardware constellations or network layouts and traces of physical entities such as files and executables. In this sense, the term 'software architecture' strongly resembles 'system architecture' that includes hardware, software and system environment (Rechtin, 1992; O'Neil *et al*., 2000; Maier & Rechtin, 2002). Table 1 shows selected definitions of the terms 'software architecture' and 'system architecture.'

The definitions proposed for software architecture (many available at Software Engineering Institute, 2006) reflect this structural perspective, exemplified as components and their arrangement. Several of these definitions have been criticized as oversimplifications (e.g., Baragry & Reed, 2001). For example, the definition by IEEE (2000) emphasizes the structural aspect by defining architecture as 'the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.' The emphasis on arrangement of components into a configuration is evident in many more definitions (Garlan & Shaw, 1993; Shaw & Garlan, 1996; Hofmeister *et al*., 1999; Bosch, 2000; Dashofy *et al*., 2005). Typically, the definitions focus on a single software system, unlike the terms 'information systems architecture' and 'enterprise architecture,' which refer to the architecture and infrastructure that encompasses information technology, employees, procedures and objectives among other things (IBM Corporation, 1975;

Table 1 Definitions of software architecture and system architecture

Term	*Definition*	*Source*
Software architecture	The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.	IEEE (2000)
	The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements and the relationships between them.	Bass *et al.* (2003)
	Software architecture involves the structure and organization by which modern system components and sub-systems interact to form systems, and the properties of systems that can best be designed and analyzed at the system level.	Kruchten *et al.* (2006)
	Structural issues include overall organization and global control structure; protocols for communication, synchronization and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives.	Garlan & Shaw (1993)
	Architecture descriptions of software systems are generally composed of at least three key entities: components, connectors and configurations.	Dashofy *et al.* (2005)
	The software architecture should define and describe the elements of the system at a relatively coarse granularity. It should describe how the elements fulfil the system requirements, including which elements are responsible for which functionality, how they interact with each other, how they interact with the outside world, and their dependencies on the execution platform.	Hofmeister *et al.* (1999)
System architecture	The underlying structure of a system, such as a communication network, a neural network, a spacecraft, a computer, major software or an organization.	Rechtin (1992)
	A system's fundamental, abstract structure that determines its behavior defined in terms of components, connections and constraints, along with the system's interactions with its environment.	O'Neil *et al.* (2000)

Zachman, 1987; Karimi, 1988; Sowa & Zachman, 1992). The term 'system infrastructure' inherits this emphasis and identifies different layers of an organizational infrastructure including components such as network, databases, servers and people (e.g., Monteiro & Hanseth, 1995; Weill & Broadbent, 1998; Maier & Rechtin, 2002).

Tracing the genealogy of the general term 'architecture' (Merriam-Webster, 2002), we see that it embraces several meanings including: (1) the art or science of building; specifically, the art or practice of designing and building structures, particularly habitable ones; (2) formation or construction as or as if as the result of conscious act; (3) a unifying or coherent form or structure; (4) architectural product or work; (5) a method or style of building; and (6) the manner in which the components of a computer or computer system are organized and integrated. This ambiguity of meanings and lack of consensus is visible also in the software architecture field as also expressed by Kruchten *et al*. (2006). One response to this ambiguity by the IFIP WG2.10 on software architecture is to define software architecture through its usage and sub-areas (Kruchten *et al*., 2006): (a) architectural design, or how the architecture is produced; (b) architecture analysis, or how we interpret and analyze the architecture and final product; (c) realization, or how to produce a working system from architecture descriptions; (d) representation, or how to produce design artifacts that are durable over time; and (e) economics, particularly the economics of architectural decisions. Although useful for focusing efforts for different sub-groups of professionals, these sub-areas do not provide clear definitions, nor do they claim to provide a non-overlapping set of categories. To understand concerns related to architecture representation better, we review this stream next.

'Representing' software architecture

Researchers from a number of disciplines have focused on clarifying the nomenclature for components that may be part of software architecture and enumerating ways of specifying interconnections among these components. Research efforts in this direction have resulted in several architectural models and ADLs such as MetaH (Binns *et al*., 1996), UniCon (Shaw *et al*., 1995), Rapide (Luckham *et al*., 1995), ACME (Garlan *et al*., 1997) and AML (Wile, 1999). To handle the large number of components and allow different partitioning of the models, architectural viewpoints (Zachman, 1987; Kruchten, 1995; IEEE, 2000) have been introduced. The proposals for ADLs have ranged from informal boxes and arrows diagrams (Hevner & Mills, 1993) and adaptations of software modeling formalisms (Medvidovic & Taylor, 2000) to identification of common architectural patterns and styles (Monroe *et al*., 1997) and specific ADLs aimed at different domains (Medvidovic *et al*., 2002; Dashofy *et al*., 2005). There are also a number of industrial approaches (Soni *et al*., 1995; Hofmeister *et al*., 1999;

Kruchten, 1999; Bass *et al*., 2003) and an attempt to build a general model of software architecture design based on these approaches (Hofmeister *et al*., 2007).

As expected, academic research related to architecture representation has emphasized correctness, consistency and coherence (Simpson *et al*., 1998; O'Neil *et al*., 2000) of structural representation, a concern that is important to specialists who deal with implementation (i.e., software architects and designers). An unintended by-product of this emphasis (Bichler *et al*., 1998) has been the exclusion of other stakeholders (such as managers and customers) from the process even though they are likely to engage in the creation and use of software architecture. As a result, although considerable progress has been made in understanding and representing structural properties of software, the outcomes have remained largely limited to academia. Sustained benefits from academia to practice are still difficult to identify, and investments of time and effort are considered unacceptable for software projects unless they represent very large undertakings. The bulk of research in architecture representation and ADLs has, therefore, not found widespread acceptance (Dashofy *et al*., 2005). This lack of acceptance raises questions regarding whether the concept of software architecture, and the research trajectory suggested by a focus on structure, can lead to breakthrough results that can significantly influence software engineering practice. Dashofy *et al*. (2005) in their early work make a similar observation, but stop short of asking for alternative research trajectories. Instead, they call for faster ways of describing architectures that continue to use ADLs. In their later work (Medvidovic *et al*., 2007), they part ways with this stance, and instead, explicitly recognize multiple stakeholders for software architecture with what they call a lamppost model. The often-cited study by Wynekoop & Russo (1995) raised similar concerns about use of software development methods, such as whether they are really used, how they are selected, and whether they work in practice or are only researcher tools. The recent Medvidovic *et al*.'s study (2007) is different from the Wynekoop and Russo's study (1995) in that the latter presents empirical evidence for their arguments. The lamppost model presented by Medvidovic *et al*. (2007) appears to have substantial face validity. The arguments, however, lack empirical foundation. In this paper, we extend this academic research based on findings from an empirical study.

Design and use of software architecture

Contrary to the assumption that underlies research on representing software architectures (i.e., that users, managers and customers are seen as passive participants), other streams of research acknowledge a stronger and more active role for these stakeholders in the design and use of software architecture (Bass *et al*., 2003; Kruchten *et al*., 2006). For example, Kazman implicitly accounts for multiple organizational stakeholders and their needs when designing or analyzing architecture (Kazman *et al*., 1999). Bosch (2000) describes possibilities for stakeholder involvement in the production of software product lines, although he does not fully account for the needs of different stakeholders. Jazayeri *et al*. (2000) also recognize the need to expand the scope of the term, when they suggest that software architecture includes a set of concepts and design decisions about the structure and texture of software that enables effective satisfaction of explicit functional and quality requirements as well as implicit requirements of the product family, the problem and the solution domains (Jazayeri *et al*., 2000). Their definition implicitly accounts for multiple stakeholders, who must participate in the design and use of software architecture. Instead of making singular representations of the overriding concern, they suggest multiple concrete perspectives that may enable negotiations, and hence, consensus and coherence.

Few researchers have developed these ideas further. Grinter (1999) observes that the role of an architect in an organization was more diverse than merely the specifier of the structure of a system. Smolander & Päivärinta (2002a) find a large set of stakeholders, including external customers, managers and eventual users, who must participate in the development of software architecture. Both studies suggest that software architects are engaged not only in designing a solution, but also in significant communication with other stakeholders who contribute to the design process. Grinter (1999) also finds that architects need to coordinate with problem owners to devise architecture descriptions to ensure that resources are earmarked in response to the project schedule. Smolander & Päivärinta (2002a) identify more than 20 stakeholder groups involved in producing architectural descriptions. Smolander & Rossi (2008) outline how design techniques could support these groups. Findings from these studies illustrate the myriad roles that a software architect must play, including those with social and political overtones, and confirm that other stakeholders must also have significant participation in the architecture development process. Finally and more recently, Medvidovic *et al*. (2007) acknowledge multiple perspectives – technology, domain and business – that must be acknowledged to 'illuminate' software architectural descriptions. They also notice the process- and decision-oriented nature of much of the use of software architecture in practice. Taylor & van der Hoek (2007) also recognize the importance of decisions and evolution management. Findings from this stream of research suggest a possible reason for the problem observed in the previous section: lack of sustained acceptance of ADLs in practice.

Together, this brief review of research provides the underpinning for our research question: *How is software architecture developed and used in an organization?* Specifically, we focus on arriving at an *empirical* understanding of the many meanings that stakeholders attach to the

term 'software architecture,' and how they work with these meanings through its creation and use.

Research method

We followed an immersive research approach similar to that followed by Curtis *et al*. (1988). As the review of prior work shows, no overarching theories are available about our phenomenon of interest (although many assertions have been made about possible uses of architecture in practice (e.g., Garlan, 2000; IEEE, 2000; Bass *et al*., 2003)). As a result, no *a priori* theoretical position was assumed nor were any hypotheses stated for testing. The research, instead, proceeded as an exploratory study (Yin, 1994) with the objective of generating preliminary theoretical constructs about the phenomenon of interest based on empirical observations. The research method, therefore, followed the grounded theory development approach (Glaser & Strauss, 1967; Strauss & Corbin, 1990).

Grounded theory development uses qualitative content analysis of data about the phenomenon under study to construct a theory. Grounded theory approaches have been shown to be useful when dealing with phenomena that are new or not well understood. The meanings of architecture in practice exemplify such an area. Because theory creation following this approach is strongly grounded to the data (instead of researcher's intuition), 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 & Ariav, 1991; Orlikowski, 1993; Volkoff *et al*., 2005). Software engineering research also recognizes the need for qualitative approaches in the areas related to human behavior (Seaman, 1999; Shaw, 2003; Sjöberg *et al*., 2007).

The setting for the study consisted of three software-producing organizations (Table 2). The first organization was a telecom service developer; the second, a software developer for handheld devices; the third, a developer of tailored IT solutions. They described themselves as advanced users of state-of-the-art techniques and methods for software development practices. This claim was supported in that they all made extensive use of Unified Modeling Language for design and of Java and components during implementation.

Data collection, analysis and validation

A pre-study was conducted with each organization with the intent of gathering background information. This was done as informal interviews and served as the basis for interpretations of later data collection efforts (Smolander *et al*., 2002). Following the pre-study, the research proceeded in four broad phases: (a) data collection in the form of interviews, (b) initial data analysis following open coding, (c) qualitative analysis of the codes and identification of theoretical constructs and (d) confirmatory interviews and focus groups (see Figure 1).

During the *first* phase, one of the researchers conducted interviews in each organization using a theoretical sampling strategy (Strauss & Corbin, 1990). The interviews were accompanied by questionnaires and prepared presentations by the chief architects working in these organizations. The data gathering followed a dynamic strategy, where the sample was extended and focused according to emerging needs (Glaser & Strauss, 1967). Table 3 outlines the number of individuals interviewed in each organization and their roles.

The *second* phase, which was intertwined with the first, included transcription and data analysis, that is, open coding. The open coding thus proceeded in parallel, which allowed treating each interview as confirmation or further development of results from earlier interviews. Open coding (Strauss & Corbin, 1990) was done using ATLAS.ti (Scientific Software, 2005) (see Figure 2) by seeding it with high-level categories (Miles & Huberman, 1984) derived from the research question. These included: stakeholders, problems in architecture design and description, and rationale for architecture description. The seed categories were extended, and existing ones were merged as new evidence and interpretations emerged from the analysis.

The open coding resulted in 179 categories. These were grouped to simplify further analysis. This grouping resulted in eight super-categories named 'communication,' 'general features,' 'problems,' 'rationales,' 'solutions,' 'stakeholders,' 'tools' and 'viewpoints.' Each super-category consisted of multiple (4–36) specific categories. The *third* phase, axial and selective coding, involved analysis of the categories and super-categories, including their values, with a view to building linkages among them. A significant output of this analysis was identifica-

Table 2 Study setting

Organization	*Business*	*Description of operations*	*Employees*[a]
Organization A	Telecom service developer	Development of software-based telecom services and platforms for in-house use	200
Organization B	Handheld software producer	Software and tool development for mobile terminals and handheld devices	200
Organization C	IT solution provider	Development of tailored information systems as dictated by customers	400 in one division, 600 in another

[a]Number engaged in software development.

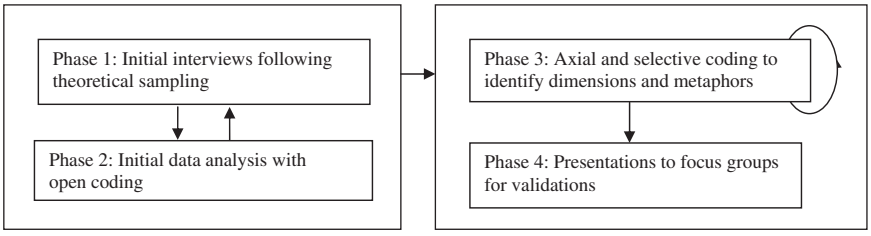

Figure 1 The research process.

...	
Q: Do you mean that you can describe [architecture] better with PowerPoint?	
A: You can do it much better with it. You can draw empty boxes with Rational Rose, but it isn't as visual.	Problem: tool constraints
Q: Is it a problem of looking good?	
A: When you are presenting it to salesmen and customers, yes it is.	Problem: visual appearance Stakeholder: customer management & marketing Stakeholder: customer
Q: Is it important that you show pretty pictures to customers?	
A: Yes. Especially when you can tell with that picture what you have been thinking. In addition, many times when we are making requirements documents, the customer wants architecture documentation as a PowerPoint presentation. They present the architecture also to other possible suppliers and they do not want to redraw the pictures. It is little like giving a tool to the customer too.	Problem: communicating meanings Rationale: communicating Stakeholder: customer Stakeholder: other suppliers

Figure 2 An example of open coding during Phase 2.

Table 3 Interviews conducted during Phase 1

Organization	*Personnel interviewed*
Organization A	2 architects, 2 designers, 2 managers, 1 manager of an in-house customer
Organization B	1 architect, 1 designer, 4 managers
Organization C	3 architects, 1 designer, 2 managers

tion of multiple dimensions that would allow greater understanding of the concept of software architecture. For instance, the super-category 'viewpoints' was leveraged along with the super-categories 'stakeholders' and 'rationale' to identify how different stakeholders used different perspectives to achieve their objectives. A specific example of this described how the 'stakeholder' manager used the 'perspective' of selecting from among available alternatives when the 'rationale' for architecture description was making strategic decisions about technology. This process included comparative analysis (Glaser & Strauss, 1967) to determine relationships among code categories. For example, the researchers compared how 'rationales' provided by different 'stakeholders' emphasized different 'viewpoints,' and how this related to different 'problems' they perceived. This detailed analysis (Smolander & Päivärinta, 2002b) resulted in discovery of important descriptive attributes (dimensions) that underlie the concept of software architecture (see Table 4).

Table 4 Dimensions discovered during Phase 3

Dimensions	*Description*
Time-orientation	Descriptions of past architectural solutions *vs* current design situation *vs* prescriptions about future implementations
Formality	Descriptions for enabling understanding *vs* 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 *vs* negotiating *vs* sense-making
Objective	The objective of architectural design and description
Customer focus	Frequency and strength of interaction between the development organization and customers utilizing the software architecture
Business focus	Extent of reasoning the development group must make about the business area of the system

Careful analysis of the results followed to ensure that the dimensions were not overlapping. The dimensions were finally used to characterize several metaphors that the stakeholders use to understand software architecture. As one may expect, the overall research process, however, did not unfold as smoothly as outlined above. Significant iterations were part of the process, which lend further credibility to the analysis and results obtained following the grounded theory development strategy (Strauss & Corbin, 1990).

As the *fourth* and final phase of the analysis process, a specific validation step was used to further corroborate the results. This included returning to the research participants to share with them the analysis results. This took the form of presentations of findings to each organization as part of a 1-day workshop and eliciting comments in response. The findings were endorsed by the participants during these workshops. The discussions resulted in only minor adjustments to the dimensions and their eventual synthesis into different metaphors.

The multiple meanings of software architecture

Unlike prior research on ADLs (Dashofy *et al*., 2005) that assumes software architecture to be a unified objective construct, our analysis provides empirical grounding to the claim that the term is ambiguous with varying purposes and interpretations. We report the findings as different metaphors that exemplify how software architecture is perceived in practice. The general idea of a metaphor is useful to appreciate and describe human understanding of complex concepts. Prior research shows that concepts that are not concrete are easier understood with the help of metaphors. Lakoff & Johnson (1980) provide a useful explication of this idea, describing how individuals grasp new concepts they encounter by means of other concepts they know in clearer terms. In fact, the term 'software architecture' itself represents a metaphor that the software engineering community borrows from other fields that describe buildings, landscape and so on (i.e., more concrete elements) – which we can grasp more easily.

Our explication builds on the premise that 'software architecture' is a concept that needs to be understood with the help of metaphors. The use of metaphors in this manner to understand a complex concept is seen in many areas, including organizations (Morgan, 1986), information systems (Kendall & Kendall, 1993; Gallupe, 2000) and workflow management (Carlsen & Gjersvik, 1997). Our choice of metaphors to report and discuss the findings thus reflects an established tradition. Following the analysis described in the previous section, we identify four metaphors that govern the participation of different stakeholder communities in the process of creation and use of software architecture. The dimensions identified during the analysis were largely influential in arriving at these metaphors and provided a clear sense of the different metaphors. Table 5 summarizes the four metaphors with reference to values of these dimensions.

Architecture as Blueprint

The *Blueprint* metaphor is strongest among individuals involved in implementing or programming the system. For these stakeholders, architecture means a high-level description of the system, directly guiding more detailed implementation aimed at the production of individual components. Architecture descriptions are thus used for transferring explicit information from architects to designers and other software engineers. The complete specification of architecture, then, resides in and can be observed from the working implementation of the system. This metaphor is directly associated with ADLs (see Medvidovic & Taylor, 2000; Dashofy *et al*., 2005), and is clearly oriented towards the future. The following excerpts show archetypal uses of this metaphor:

> 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 DLLs. In a way it [the architecture] is the basis for the next phase, which is the DLL design. (Jack, Software Engineer)
>
> The [architecture] is usually made from the viewpoint of a programmer. […] They usually consist of lists of various source code files and instructions of how to build the system that is run in a server. (Tim, Product Manager)
>
> [Architecture includes] the components, the communication between the components, the protocols we were using, the message format itself, the inputs and outputs, what we should … produce and get from the user, the error

Table 5 The multiple meanings of software architecture

Dimensions (see Table 4)	*Blueprint: Specification of the system to be implemented*	*Language: Medium of communication for achieving common understanding*	*Decision: Choices about the system to be implemented and rationale*	*Literature: Documentation for current and future generations of users and developers*
Time	Future	Present/future	Future	Past
Formality	High	Low	Usually low	Varies
Detail	High	Low	Usually low	Usually high
Activity	Implementing	Negotiating	Evaluating choices	Reading, analyzing
Customer focus	Low	Possibly high	High	Usually low
Business focus	Low	Possibly high	High	Usually low

> handlings or how the errors should be handled, … and of course for each of the modules, … the functions of the modules and their place inside the whole system, and … the communications inside the module. (George, Designer)

As might be expected, a typical activity associated with this metaphor is implementation of software artifacts that requires both high formality and high level of detail. As a result, the metaphor clearly responds to the needs of designers and technically oriented architects but scores lower on the dimensions of customer and business focus.

Architecture as Language

The *Language* metaphor suggests that architecture enables common understanding about the system among stakeholders. The role of architecture is not providing a basis for creation of artifacts; instead, it acts as a facilitator of communication across stakeholder groups. This role of architecture is acknowledged in most new works (Bass *et al*., 2003; Medvidovic *et al*., 2007; Rozanski & Woods, 2007), but it has not been operationalized to any degree. The metaphor emphasizes understanding between people about the situation at hand. The time dimension for this metaphor focuses both on describing the present situation and on discussing alternative future scenarios. The following excerpts exemplify how the metaphor is used:

> Usually our projects use quite new technologies and you need to know at a coarse level what our gurus have designed. Therefore, it should not just be in the heads of programmers and architects. The project manager and the project steering group must also know and follow where you are going. (Harry, Project Manager)
>
> The purpose of architecture … is that you must be able to tell the customer and the team what is your idea. (John, Architect)
>
> You should not drown in papers. … From the perspective of a project manager the …[architecture] should be condensed. It should have the main points, the core points clearly described so that you don't need to invite the architect or the chief designer to the project steering group meeting to explain it (Harry, Project Manager)

Stakeholders use architecture to communicate and negotiate, that is, according to this metaphor, architecture requires minimal formality and detail. Too much formality or detail can, in fact, lead to a breakdown in communication because the language must be understood by a diverse set of stakeholders with varying backgrounds and experiences. The metaphor is emphasized by stakeholders with high customer or business focus, such as managers and marketing experts. As the examples above demonstrate, including these stakeholders also means that architecture needs to be concise and easy to understand. Increased diversity among stakeholders also contributes to a greater emphasis on thinking of architecture as language. For instance, in one of the organizations in our study, when customer participation was intense the marketing group was closely involved in the process. The metaphor of architecture as language directly corresponds to the idea of a boundary object (Star, 1989) that serves to facilitate interaction between stakeholder groups.

Architecture as Decision

The third metaphor suggests that architecture represents decision(s) about design trade-offs among properties such as cost, usability, maintainability and performance that have consequences for resources needed for building the system, including work force, special skills and monetary resources. Architecture thus becomes the process and product of decision-making concerning design tactics, strategies and associated resources (Bass *et al*., 2003; Dashofy *et al*., 2005; Rozanski & Woods, 2007). The following excerpts demonstrate use of this metaphor:

> 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 million then how much it must have usage… (Arthur, Architect)
>
> If the expected lifetime of the system is three years and we have only 2 Euros and 50 cents money, then it is useless to pursue highest quality. … If we have other objectives, like if the expected lifetime is ten years or more, or if the usability must be top class, then I know where to put the stakes and what problems must be solved. (Richard, Architect)
>
> We must get as quickly as possible an understanding of the environment and the system that we are offering into. […] With … [the architecture] … organizational decisions are put across [and]…the construction of next systems is guided so that they are obeying the same architecture. (Tom, Sales Manager)
>
> What choices are possible, what do we have now, what do we have after five years or after two years? …requirements are defined, that is features, usability, and other things that lead to certain technological choice … (Thomas, Quality Manager)

The metaphor is clearly oriented towards the future and uses architecture as a vehicle to decide attributes of a future solution based on expected resource commitments and strategies. Typical activities associated with this metaphor include evaluating alternatives and making choices about the technical solutions.

As with the Language metaphor, too much formality and detail is not desirable, particularly for resource commitment decisions. On the other hand, for technical trade-off situations, higher levels of detail may be necessary. This metaphor suggests that for stakeholders such as managers and resource planners (like project managers), architecture represents decisions that are used not only for earmarking resources but also as the basis for division of work between working units. As a decision, architecture represents the explicit commitment of a group of individuals to a course of action and enforces participation of all stakeholders to downstream processes, such as division of tasks, their required execution order and even the structure of the development organization (cf. Conway, 1968).

Architecture as Literature

The Literature metaphor is closely related to the documentation of technical structures that aid in transferring knowledge over time. Following this metaphor, architecture is seen as documentation of solutions constructed in the past, meant for future readers. In the current software architecture research, the terms analogous to this metaphor (although broader in intent) include 'reference architectures' (e.g., Batory *et al*., 1995), 'product line architectures' (Clements & Northrop, 1999) and 'architectural frameworks' (Fayad & Johnson, 2000). The following excerpts exemplify this metaphor:

> I think the purpose of architecture … is to keep the knowledge of what kind of a system we have. … It is nice to have such a document … that you can take a view and see …those who will possibly make further development or maintenance can learn the system easily with it. (Michael, Software Engineer)
>
> Later they [architecture] are good just when the maintenance is switched to the next team, which can be brought in with these pictures showing that we have this and that kind of building blocks in these computers and they are doing this and that in this case. (Hannah, Software Engineer)
>
> There is this time perspective that when people change, even organizations can change … Therefore we must be able to pass over the essential things about the systems fast. … It is very important that you can have such descriptions that you can quickly get a grasp on the system from. (Tom, Sales Manager)

As expected, typical activities associated with this metaphor include reading and analyzing (from the recipients of this 'literature'), because it supports not only construction of artifacts but also the transfer of explicit knowledge between designers and maintainers. Bass *et al*. (2003) point out that in fact most development work is maintenance, and thus this metaphor is vital for longevity of the architecture. This also dictates the level of detail that tends to be high although the formality of descriptions varies. An interesting variation of the use of this metaphor was also related to the concern of reuse of components and designs. As literature, the metaphor focuses on the past and tends to have an internal focus (i.e., less of a customer or business focus).

Using the metaphors

The metaphors described above do overlap and are sometimes used simultaneously by different stakeholders. A stakeholder may also use multiple metaphors although the emphasis varies across stakeholders and over time. For example, among the participants we interviewed, the role of an architect often meant simultaneous and balanced use of all four metaphors. The metaphors clearly show how each stakeholder group perceives and uses software architecture for satisfying its own informational requirements that differ from the requirements of other groups. These requirements relate to concerns such as buying or selling the system, running the system, understanding its operation, understanding the project scope and estimating its progress, designing the high-level structure of the system or programming the components of the system. The metaphors thus serve to highlight multiple *uses* of architecture by the different stakeholders. Medvidovic *et al*. (2007) suggest a useful perspective to think about different stakeholders using the lampposts analogy. Specifically, they suggest three lampposts: technology (that deals with providing pointers to software engineers for implementation), domain (that illuminates problems faced by users in specific domains) and business (that is concerned with fulfilling a market need and responding to complementary functionality elsewhere). The metaphors that result from our analysis can be mapped against these three lampposts to describe how each addresses the needs articulated by Medvidovic *et al*. (2007) (see Table 6).

As the table demonstrates, each metaphor not only captures a set of properties but also provides a clue to what it enables, that is, its eventual 'use.' For example, consider the Language metaphor that facilitates communication and understanding. A language follows a

Table 6 Mapping the metaphors against the lampposts of Medvidovic *et al*. (2007)

Lamppost	*Metaphors*			
	Blueprint	*Language*	*Decision*	*Literature*
Technology	Captures layers, systems and dependencies among components			Includes detailed structural descriptions that are used for maintaining and refining the architecture
Domain		Utilizes application domain concepts to communicate across the stakeholder groups		Used to learn about software architecture as implemented and deployed
Business		Includes user- and customer-focused terms to communicate intent with executives and users	Allows evaluation of alternatives, recording of rationale and building commitment to the chosen architecture	

structure and has a vocabulary, making expression and composition possible. On the other hand, the Literature metaphor suggests exposition of style, intent and narrative as it allows transfer of knowledge to future generations. The key use of architecture following the Literature metaphor is, then, the enabling of learning in the future. The Blueprint metaphor requires understanding of representation techniques and rules for applying these techniques. Representation techniques must allow dealing with multiple levels of detail, and blueprints constructed with these techniques must provide clear prescriptions for implementations. Finally, the Decision metaphor suggests that choices must be apparent from the architecture that should also permit analysis of choices by reflecting their consequences.

A process perspective on the design and use of software architecture provides another window into how the metaphors may be useful. Software architecture development can be described as a process, where informal initial descriptions aimed at understanding and objective-setting give way to formal and detailed specifications as the process moves from conceptualization to implementation. The emphasis on the four metaphors varies through the process. At the beginning of a project, the Language and Decision metaphors are more prominent as internal and external stakeholders negotiate to achieve a common understanding and assess the consequences of choices. As the project proceeds to implementation, technically oriented stakeholders such as architects, designers and programmers take over, and the emphasis shifts to the Blueprint metaphor. After the system is deployed, the Literature metaphor gains more emphasis as new participants (users and maintainers) must operate or maintain the system, which requires understanding its structure and principles through documentation.

All four metaphors endure through the lifecycle: the system structure is constantly communicated and understood (Language), new choices are made and implications of old choices are understood (Decision), implementation with existing and new components is guided (Blueprint) and documents are updated to enable learning (Literature). Architecture, thus serves as a shared boundary object (Star, 1989; Star & Griesemer, 1989; Bowker & Star, 1999) between various stakeholder groups engaged in systems development, satisfying their varying informational needs during the systems development process.

Discussion and implications

An obvious implication of our work is that there are multiple perspectives on software architecture. This is hardly a surprising finding in itself. The value of our work is that it provides empirical grounding for this assertion and suggests specific metaphors that can be useful to interpret prior research and provide directions for future work. Our goal was to find how architecture is used in organizations. The identification of the four metaphors and suggestions for their use extend the current understanding of software architecture by explicitly recognizing the information needs of business users and customers.

Whereas much prior work on software architecture is focused on a view of architecture as a Blueprint or Literature, more research is needed to support the view of architecture as Decision (Kazman *et al*., 2001; Rozanski & Woods, 2007). There is scant research that views architecture as Language, a key metaphor to facilitate interaction among stakeholder groups. Writings such as that by Medvidovic *et al*. (2007) and Rozanski & Woods (2007) are beginning to realize that there are multiple areas that architecture should illuminate (see Table 6). Our findings are orthogonal to their arguments and extend this stream of research by proposing four metaphors that may define how areas under lampposts of Medvidovic *et al*. (2007) can be illuminated. The lampposts then present different aspects of architecture, whereas the metaphors provide an answer to the views needed for each aspect. We now outline implications of these findings for research and practice.

Implications for research

We may speculate that the current, narrowly focused vision of ADLs may be traced to the emergent nature of the discipline, or because current research on architecture is dominated by certain perspectives at the expense of others. Although debatable, one argument holds that ADLs are beginning to respond to this challenge, and as a result, are beginning to find greater acceptance with practitioners (Medvidovic *et al*., 2007). If true, the results reported in this paper may provide a possible explanation for this trajectory. The early and exclusive focus in software architecture (as a Blueprint) may have served the technical audience well, but it resulted in lack of openness needed by other stakeholders (similar to arguments about boundary objects suggested by Bowker & Star (1999)). As the discipline is maturing (Shaw & Clements, 2006), software architecture representations are moving towards greater recognition of domain concepts and business needs (Bass *et al*., 2003; Hofmann, 2003). This bodes well for a progression towards addressing other stakeholders, such as users and managers (Rozanski & Woods, 2007). In this view software architecture emerges from the cooperation of different stakeholders while recognizing the situational constraints such as legacy systems, available technology, organizational conflicts, resource constraints, skills and experience.

Ensuring that architecture plays a role as an enabler of communication between a diverse set of participants (including various levels of management and technical experts) will, however, require informal and expressive approaches, which are non-existent so far. Although approaches such as the $4+1$ architectural view (Kruchten, 1995) are available, they represent a forced compromise with a primary focus on technological requirements without squarely addressing these communication needs.

Research on software architecture may, therefore, benefit from work in related fields such as CSCW and information systems planning (e.g., Tellioglu *et al.*, 1998; Ward & Peppard, 2002), which suggest reconciling different perspectives, and the work by Weill & Broadbent (1998), which suggests viewing architectures and investments as digital options that organizations can exercise which form the scaffolding of the current architecture (Orlikowski, 2006). In addition, architectural thinking is at the core of many recent advances in information systems development, such as service-orientation (Papazoglou & Georgakopoulos, 2003), enterprise resource planning (Kumar & Hillegersberg, 2000), business process management (van der Aalst *et al.*, 2003) and enterprise architectures (Ross *et al.*, 2006) and therefore we must invent new and better ways to design, represent and develop architectures. We believe that the results presented here enable researchers to better structure the concept of architecture.

Other, specific responses to the results we have outlined can include techniques that allow architecture specifications to be used in ways that can cross boundaries across stakeholders without sacrificing the need to help downstream activities such as implementation and deployment (e.g., the conformance between architecture and code (Bass *et al.*, 2003; Shaw & Clements, 2006)). For a more managerial perspective the business and strategy development methods suggested by Eden & Ackerman (1998) may provide additional avenues to create a common understanding and vision of business strategy. Finally, the work by Ciborra (2000) may provide another pointer to deriving architectural specifications that take into account their emergent and improvized nature as opposed to carefully planned common road maps (i.e., maintaining multiple and even conflicting views of architecture simultaneously).

Implications for practice

Architectural specifications should be able to span the spectrum from vague and noble ideals to stringent decisions about technical platforms and data interchange formats. A useful adjunct to this requirement would be the ability to use these representations to make explicit the consequences, conflicts and problems that the stakeholders are likely to face. A significant challenge that this presents to the research community and practitioners is how to combine the two, sometimes conflicting requirements for architectural descriptions: (a) they should support development at a concrete level, (b) they should be intelligible to various stakeholders participating in the process and (c) they should allow reasoning following technological as well as political concerns.

Our observations also revealed preliminary patterns of emphasis over the four metaphors based on organizational characteristics (Smolander & Päivärinta, 2002a, b). When the organization was homogenous, the Blueprint metaphor prevailed as exemplified by Organization A (the handheld software producer), where developers were mostly engineers with similar education and required little contact with external stakeholders such as customers. On the other hand, the role of customers and other external stakeholders highlighted the use of the Language metaphor that we observed in Organization C (the IT solution provider). Their diverse set of customers and other external partners emphasized the use of architecture descriptions as communication tools instead of tools for detailed design and implementation.

These observations of organizational differences suggest that no generic model is likely to fit the needs for architecture design languages. The notion of goodness of models for a homogenous group of engineers would not be the same as that for a heterogeneous set of developers, customers and other external partners with diverse backgrounds. The practice of architecture development should, therefore, reflect the needs of the organization and its development practices. We speculate that techniques such as sketching and a succession of more detailed specifications, similar to those used by architects in the early phases of building planning (Morris, 2006), may provide a useful alternative to current prescriptions.

The recent developments in information systems design towards higher level and more business-oriented specification languages and approaches (e.g., SOA, BPM, etc.) could benefit from our metaphors by providing different perspectives for different stakeholders. With notations that follow (for example) the Decision metaphor, these technologies could be more immediately accessible to managers, customers and other stakeholders.

Concluding remarks

In this paper, we have explored the concept of software architecture in three real-life organizations engaged in software development. We used a research method intended to produce empirically grounded constructs that describe different *meanings* of software architecture as perceived by various actors, who participate in its creation and use. The analysis we have described follows the tradition of reporting results as metaphors that distill the findings into evocative phrases. Specifically, the study resulted in four metaphors for architecture: Blueprint, Literature, Language and Decision. The results support and extend recent work that has argued for multiple areas that architecture must illuminate with a lamppost analogy (Medvidovic *et al.*, 2007). In doing so, we hope that we have provided a first step towards understanding key meanings different stakeholders attach to software architecture and providing evidence for these. We hope that the metaphors and the corresponding discussion will spur further research to refine our understanding of software architecture as well as developing additional architectural representations that reflect this understanding.

Acknowledgements

We would like to acknowledge feedback on an early version of the manuscript from Shawn Clark.

About the authors

Kari Smolander is a Professor of Software Engineering in the Department of Information Technology, Lappeenranta University of Technology, Finland. He has a Ph.D. (2003) in Computer Science from Lappeenranta University of Technology and a Licentiate (1993) and Master (1988) degree from University of Jyväskylä, Finland. In addition to his long teaching experience, he has worked for several years in the industry and in the 1990s he was the main architect in the development of the MetaEdit CASE tool. He has published more than 50 refereed research papers in international journals and conferences. His current research interests include the architectural aspects of systems development and the organizational view of software development.

Matti Rossi is a Professor of Information Systems at Helsinki School of Economics. He has worked as research fellow at Erasmus University Rotterdam, visiting assistant professor at Georgia State University, Atlanta and visiting professor at Claremont Graduate University. He received his Ph.D. degree in Business Administration from the University of Jyväskylä in 1998. He has been the principal investigator in several major research projects funded by the Technological Development Center of Finland and Academy of Finland. He is the coordinating editor of *Scandinavian Journal of Information Systems*. His research papers have appeared in journals such as *CACM*, *Journal of AIS*, *Information and Management* and *Information Systems*, and over 30 of them have appeared in conferences such as ICIS, HICSS and CAiSE.

Sandeep Purao is an Associate Professor at the College of Information Sciences and Technology and part of the Enterprise Informatics and Integration Center at Penn State University, University Park. His research focuses on the design, evolution and management of techno-organizational systems, blending research methods from social science and software engineering. Outcomes from these efforts include publications in archival journals such as *Information Systems Research*, papers in conference proceedings such as WITS, and the design and implementation of software artifacts such as APSARA, tied to empirical assessments. He also enjoys reflecting on the craft of teaching in this domain, and has published on pedagogical aspects of organizational informatics.

References

BARAGRY J and REED K (2001) Why we need a different view of software architecture. In *Proceedings of the Working IFIP/IEEE Conference on Software Architecture (WICSA 2001)*, pp 125–134, IEEE Computer Society, Amsterdam, The Netherlands.

BASS L, CLEMENTS P and KAZMAN R (2003) *Software Architecture in Practice* 2nd edn. Addison-Wesley, Boston.

BATORY D, COGLIANESE L, GOODWIN M and SHAFER S (1995) Creating reference architectures: an example from avionics. In *Proceedings of the Symposium on Software Reusability*, pp 27–37 ACM, Seattle, Washington, April 1995.

BICHLER M, SEGEV A and ZHAO JL (1998) Component-based e-commerce: assessment of current practices and future directions. *SIGMOD Record* **27(4),** 7–14.

BINNS P, ENGLEHART M, JACKSON M and VESTAL S (1996) Domain-specific software architectures for guidance, navigation, and control. *Journal of Software Engineering and Knowledge Engineering* **6(2),** 201–227.

BOSCH J (2000) *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. Addison-Wesley, Harlow.

BOWKER GC and STAR SL (1999) *Sorting Things Out: Classification and its Consequences*. MIT Press, Cambridge, MA.

CALLOWAY LJ and ARIAV G (1991) Developing and using a qualitative method to study relationships among designers and tools. In *Information Systems Research: Contemporary Approaches and Emergent Traditions* (NISSEN HE, KLEIN HK and HIRSCHHEIM R, Eds), pp 175–193, North-Holland, Amsterdam.

CARLILE PR (2002) A pragmatic view of knowledge and boundaries: boundary objects in new product development. *Organization Science* **13(4),** 442–455.

CARLSEN S and GJERSVIK R (1997) Organizational metaphors as lenses for analyzing workflow technology. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge*, pp 261–270, ACM Press, Phoenix, AZ, USA.

CIBORRA C (2000) Drifting: from control to drift. In *Planet Internet* (BRAA K, SORENSEN C and DAHLBOM B, Eds), Studentlitteratur, Lund.

CLEMENTS P, BACHMANN F, BASS L, GARLAN D, IVERS J, LITTLE R, NORD R and STAFFORD J (2002) *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, Boston.

CLEMENTS P and NORTHROP LM (1999) *A Framework for Software Product Line Practice – Version 2.0*. Software Engineering Institute, Pittsburgh.

CONWAY ME (1968) How do committees invent? *Datamation* **14(4),** 28–31.

CURTIS B, KRASNER H and ISCOE N (1988) A field study of the software design process for large systems. *Communications of the ACM* **31(11),** 1268–1287.

DASHOFY EM, VAN DER HOEK A and TAYLOR RN (2005) A comprehensive approach for the development of modular software architecture description languages. *ACM Transactions on Software Engineering and Methodology* **14(2),** 199–245.

EDEN C and ACKERMAN F (1998) *Making Strategy: The Journey of Strategic Management*. Sage Publications, London.

ERL T (2004) *Service-Oriented Architecture: A Field Guide to Integrating Xml and Web Services*. Prentice-Hall, Upper Saddle River.

FAN M, STALLAERT J and WHINSTON AB (2000) The adoption and design methodologies of component-based enterprise systems. *European Journal of Information Systems* **9(1),** 25–35.

FAYAD EM and JOHNSON ER (2000) *Domain-specific Application Frameworks*. John Wiley & Sons, New York.

GALLUPE RB (2000) Images of information systems in the early 21st century. *Communications of the AIS* **3(1),** 1–16.

GARLAN D (2000) Software architecture: a roadmap. In *The Future of Software Engineering* (FINKELSTEIN A, Ed.), ACM Press, New York.

GARLAN D, MONROE RT and WILE D (1997) Acme: an architecture description interchange language. In *Proceedings of CASCON'97*, pp 169–183, Toronto, Canada.

GARLAN D and SHAW M (1993) An introduction to software architecture. In *Advances in Software Engineering and Knowledge Engineering, Series on Software Engineering and Knowledge Engineering*, (AMBRIOLA V and TORTORA G, Eds), Vol. 2, pp 1–39, World Scientific Publishing Co., Singapore.

GLASER B and STRAUSS AL (1967) *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine, Chicago.

GRINTER RE (1999) Systems architecture: product designing and social engineering. *ACM SIGSOFT Software Engineering Notes* **24(2),** 11–18.

HEVNER AR and MILLS HD (1993) Box-structured methods for systems development with objects. *IBM Systems Journal* **32(2),** 232–251.

HOFMANN L (2003) *Beyond Software Architecture – Creating and Sustaining Winning Solutions*. Addison-Wesley, Boston.

HOFMEISTER C, KRUCHTEN P, NORD RL, OBBINK H and AMERICA P (2007) A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software* **80(1),** 106–126.

HOFMEISTER C, NORD R and SONI D (1999) *Applied Software Architecture*. Addison-Wesley, Reading, MA.

HUHNS MN and SINGH MP (2005) Service-oriented computing: key concepts and principles. *IEEE Internet Computing* **9(1),** 75–81.

IBM CORPORATION (1975) *Business Systems Planning: Information Systems Planning Guide*. IBM, #GE20-0527-4, New York.

IEEE (2000) *IEEE Std 1471-2000: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. IEEE, New York.

JAZAYERI M, RAN A and LINDEN F (2000) *Software Architecture for Product Families: Principles and Practice*. Addison-Wesley, Upper Saddle River.

KARIMI J (1988) Strategic planning for information systems: requirements and information engineering methods. *Journal of Management Information Systems* **4(4),** 5–24.

KAZMAN R, ASUNDI J and KLEIN M (2001) Quantifying the costs and benefits of architectural decisions. In *Proceedings of 23rd International Conference on Software Engineering (ICSE 2001)*, pp 297–306, IEEE Computer Society.

KAZMAN R, BARBACCI M, KLEIN M, CARRIÈRE SJ and WOODS SG (1999) Experience with performing architecture tradeoff analysis. In *Proceedings of the 1999 International Conference on Software Engineering*, pp 54–63, IEEE Computer Society Press.

KENDALL JE and KENDALL KE (1993) Metaphors and methodologies: living beyond the systems machine. *MIS Quarterly* **17(2),** 149–171.

KRUCHTEN P, OBBINK H and STAFFORD J (2006) The past, present, and future for software architecture. *IEEE Software* **23(2),** 22–30.

KRUCHTEN PB (1995) The 4+1 view model of architecture. *IEEE Software* **12(6),** 42–50.

KRUCHTEN PB (1999) *Rational Unified Process: An Introduction*. Addison-Wesley, Upper Saddle River.

KUMAR K and HILLEGERSBERG JV (2000) Enterprise resource planning: introduction. *Communications of the ACM* **43(4),** 22–26.

LAKOFF G and JOHNSON M (1980) *Metaphors We Live by*. The University of Chicago Press, Chicago.

LINTHICUM D (2000) *Enterprise Application Integration*. Addison-Wesley, Upper Saddle River.

LUCKHAM DC, KENNEY JC, AUGUSTIN LM, VERA J, BRYAN D and MANN W (1995) Specification and analysis of system architecture using rapide. *IEEE Transactions on Software Engineering* **21(4),** 336–355.

MAIER MW and RECHTIN E (2002) *The Art of Systems Architecting*, 2nd edn. CRC Press, Boca Raton.

MEDVIDOVIC N, DASHOFY EM and TAYLOR RN (2007) Moving architectural description from under the technology lamppost. *Information and Software Technology* **49(1),** 12–31.

MEDVIDOVIC N, ROSENBLUM DS, REDMILES DF and ROBBINS JE (2002) Modeling software architectures in the unified modeling language. *ACM Transactions on Software Engineering and Methodology* **11(1),** 2–57.

MEDVIDOVIC N and TAYLOR RN (2000) A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering* **26(1),** 70–93.

MERRIAM-WEBSTER (2002) *Merriam Webster's Collegiate Dictionary* 10th edn (accessed 12 October 2002).

MILES MB and HUBERMAN AM (1984) *Qualitative Data Analysis: A Sourcebook of New Methods*. Sage, Beverly Hills.

MONROE RT, KOMPANEK A, MELTON R and GARLAN D (1997) Architectural styles, design patterns, and objects. *IEEE Software* **14(1),** 43–52.

MONTEIRO E and HANSETH O (1995) Social shaping of information infrastructure: on being specific about the technology. In *Information Technology and Changes in Organizational Work* (ORLIKOWSKI WJ, WALSHAM G, JONES MR and Degross JI, Eds), pp 325–343, Chapman & Hall, London.

MORGAN G (1986) *Images of Organization*. Sage, Beverly Hills.

MORRIS M (2006) *Models: Architecture and the Miniature (Architecture in Practice*. Academy Press, Wiley, New York.

O'NEIL T, LEANEY J, ROWE D, SIMPSON H, RANGARAJAN M, WEISS J, PAPP Z, BAPTY T, PURVES B, HORVATH G and DE JONG E (2000) IEEE ECBS'99 TC Architecture Working Group (AWG) Report. In *Proceedings of the Seventh IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*. pp 383–389.

ORLIKOWSKI WJ (1993) Case tools as organizational change: investigating incremental and radical changes in systems development. *MIS Quarterly* **17(3),** 309–340.

ORLIKOWSKI WJ (2006) Material knowing: the scaffolding of human knowledgeability. *European Journal of Information Systems* **15,** 460–466.

PAPAZOGLOU MP and GEORGAKOPOULOS D (2003) Service-oriented computing. *Communications of the ACM* **46(10),** 25–28.

RECHTIN E (1992) The art of systems architecting. *IEEE Spectrum* **29(10),** 66–69.

ROSS JW, WEILL P and ROBERTSON DC (2006) *Enterprise Architecture as Strategy: Creating a Foundation for Business Execution*. Harvard Business School Press, Boston.

ROZANSKI N and WOODS E (2007) *Software Systems Architecture – Working with Stakeholders using Viewpoints and Perspectives*. Addison-Wesley, Upper Saddle River.

SCIENTIFIC SOFTWARE (2005) *Atlas.Ti – The Knowledge Workbench*. Scientific Software Development, Berlin, Germany.

SEAMAN CB (1999) Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering* **25(4),** 557–572.

SHAW M (2003) Writing good software engineering research papers: minitutorial. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, pp 726–736, IEEE Computer Society, Portland, Oregon.

SHAW M and CLEMENTS P (2006) The golden age of software architecture. *IEEE Software* **23(2),** 31–39.

SHAW M, DELINE R, KLEIN DV, ROSS TL, YOUNG DM and ZELESNIK G (1995) Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering* **21(4),** 314–335.

SHAW M and GARLAN D (1996) *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Upper Saddle River.

SIMPSON H, ROSSAK W, SCHAFFER C, HAMMER D, ROZENBLIT J, BOASSON M, ROWE D, LEANEY J, KIROVA V, KRADJEL H and LAWSON B (1998) *IEEE ECBS TC Architecture Focus Group Discussion Paper*. (accessed 6 September 2002).

SJÖBERG DIK, DYBÅ T and JORGENSEN M (2007) The future of empirical methods in software engineering research. In *FOSE'07: 2007 Future of Software Engineering*, (BRIAND L and WOLF A, Eds), pp 358–378, IEEE Computer Society.

SMOLANDER K, HOIKKA K, ISOKALLIO J, KATAIKKO M and MÄKELÄ T (2002) What is included in software architecture? A case study in three software organizations. In *Proceedings of 9th Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS)*, pp 131–138 IEEE Computer Society, Lund, Sweden, 8–11 April 2002.

SMOLANDER K and PÄIVÄRINTA T (2002a) Describing and communicating software architecture in practice: observations on stakeholders and rationale. In *Proceedings of CaiSE'02 – The Fourteenth International Conference on Advanced Information Systems Engineering* (BANKS PIDDUCK A, MYLOPOULOS J, WOO CC and OZSU MT, Eds), pp 117–133, Springer-Verlag, Toronto, Canada 27–31 May 2002.

SMOLANDER K and PÄIVÄRINTA T (2002b) Practical rationale for describing software architecture: beyond programming-in-the-large. In *Software Architecture: System Design, Development and Maintenance – IFIP 17th World Computer Congress – TC2 Stream/3rd Working IEEE/IFIP Conference on Software Architecture (WICSA3)* (BOSCH J, GENTLEMAN M, HOFMEISTER C and KUUSELA J, Eds), pp 113–126 Kluwer Academic Publishers, Montréal, Québec, Canada, 25–30 August 2002.

SMOLANDER K and ROSSI M (2008) Conflicts, compromises, and political decisions: methodological challenges of enterprise-wide e-business architecture creation. *Journal of Database Management* **19(1),** 19–40.

SOFTWARE ENGINEERING INSTITUTE (2006) *How do you define software architecture*. (accessed 29 September 2006).

SONI D, NORD RL and HOFMEISTER C (1995) Software architecture in industrial application. In *Proceedings of the 17th International Conference on Software Engineering*, pp 196–207, ACM Press, Seattle, WA, USA, 24–28 April 1995.

SOWA JF and ZACHMAN JA (1992) Extending and formalizing the framework for information systems architecture. *IBM Systems Journal* **31(3),** 590–616.

STAR SL (1989) The structure of ill-structured solutions: heterogeneous problem-solving, boundary objects and distributed artificial intelligence. In *Distributed Artificial Intelligence* (HUHNS M and GASSER L, Eds), Vol. 2, pp 37–54, Morgan Kauffmann, Menlo Park, CA.

STAR SL and GRIESEMER JR (1989) Institutional ecology, ''Translations'' and boundary objects: amateurs and professionals in Berkeley's Museum of Vertebrate Zoology, 1907–39. *Social Studies of Science* **19,** 387–420.

STRAUSS AL and CORBIN J (1990) *Basics of Qualitative Research: Grounded Theory Procedures and Applications*. Sage Publications, Newbury Park, CA.

TAYLOR RN and VAN DER HOEK A (2007) Software design and architecture the once and future focus of software engineering. In *Future of Software Engineering, 2007. FOSE '07*, (BRIAND L and WOLF A, Eds), pp 226–243.

TELLIOGLU H, WAGNER I and LAINER R (1998) Open design methodologies. Exploring architectural practice for systems design. In *Proceedings of the Participatory Design Conference PDC'98*, pp 19–28 Seattle, WA, USA, 12–14 November 1998.

VAN DER AALST WMP, TER HOFSTEDE AHM and WESKE M (2003) Business process management: a survey. In *BPM'2003*, (VAN DER AALST W, TER HOFSTEDE A and WESKE M, Eds), pp 1–12, Springer, Berlin.

VOLKOFF O, STRONG DM and ELMES MB (2005) Understanding enterprise systems-enabled integration. *European Journal of Information Systems* **14(2),** 110–120.

WARD J and PEPPARD J (2002) *Strategic Planning for Information Systems*. John Wiley & Sons, Chichester.

WEILL P and BROADBENT M (1998) *Leveraging the New Infrastructure: How Market Leaders Capitalize on Information Technology*. Harvard Business School Press, Boston, MA.

WILE D (1999) AML: an architecture meta-language. In *Proceedings of the 14th International Conference on Automated Software Engineering*, pp 183–190.

WYNEKOOP JL and RUSSO NL (1995) Systems development methodologies: unanswered questions. *Journal of Information Technology* **10,** 65–73.

YIN RK (1994) *Case Study Research: Design and Methods* 2nd edn. Sage Publications, Thousand Oaks.

ZACHMAN JA (1987) A framework for information systems architecture. *IBM Systems Journal* **26(3),** 276–292.