Chapter 5

A Theory of Dissemination and Coordination of Software Architecture Design in Global Software Development

In this chapter an analysis is presented of software architecture coordination and dissemination practices in the context of large scale, global software development. To this end, the case findings discussed in Chapter 4 are contrasted by means of a synthesis of the results of interviews with a group of experts. In addition, recommendations for software architecture documentation and dissemination are outlined.

5.1 Introduction and Objectives

In the previous chapter, we analyzed three specific industrial cases for dissemination of software architecture design in the context of offshore software development. In this chapter we outline and discuss the best practices distilled from these cases. In addition, this chapter presents a synthesis of the factors that underlie the phenomena that were observed in the previous chapter. In other words: we set out to explain *why* these best practices seem to work. The resulting grounded theory provides an answer to research question **RQ1** (Section 1.3).

We aim to improve the process of software architecture dissemination in the context of global software development and therefore pose the following research question:

How do the factors that shape how software architecture is disseminated and coordinated in large, industrial, custom, global software development projects relate?

To this end we use the results of the three case studies of large, industrial custom GSD projects presented in Chapter 4. In this chapter, we compare these case findings to a synthesis of a series of interviews with industrial experts in the fields of software architecture and GSD from various organizations. These were not the same people as those interviewed for the case studies presented in Chapter 4.

The outline of this chapter is as follows. Section 5.2 describes data collection and sections 5.3 and 5.4 outline the main factors and their main implications. Finally, sections 5.5 and 5.6 discuss best practices and conclusions and future work.

5.2 Data Collection

In addition to the case-related interviews that were reported on in Chapter 4, a group of 19 software architecture experts were interviewed at various organizations involved in offshore software development (GSD). These semi-structured interviews were all conducted on-site and face-to-face (in India). All participants were drawn from three major, international IT organizations. They were either senior developers, software architects or project managers. An overview of their function titles and experience is presented in Table 5.1. This particular group of respondents was not associated with the cases described in Chapter 4. The topics discussed during the interviews were similar to those discussed in the case-specific interviews. However, instead of concentrating on a particular case, the interviewees were specifically encouraged to reflect on their entire body of practical software development experience. The transcriptions of these interviews were used to reflect on the themes that were uncovered in the case analyses in Chapter 4. During the interview and the interview analysis, the concepts relating to case-specific phenomena (such as shared mental model measurements) were omitted as the focus of this chapter is relating concepts to factors beyond the case-level. We obtained our respondent population by means of introductions through Chain-Referral Sampling (CRS — also known as “snowball sampling”) (Heckathorn, 1997, 2002). CRS is normally used to obtain access to hard-to-find subjects in hidden populations (such as in studies regarding substance dependence as in e.g. Wang et al., 2005). While software professionals are not necessarily imperceivable in hidden populations, CRS helps to deal with some of the problems common to research in industrial software engineering practice. As software professionals, especially architects, tend to be expensive resources, their time for non-productive hours is limited. This problem is compounded by the fact that cost reductions are often part of the motivation to employ global software development practices and people are expected to deliver at short notice. The resulting work pressure at the offshore development location can therefore be relatively high. Being introduced by a potential interviewee’s peer increases the chances of getting a

Table 5.1: *Overview of expert interview respondent characteristics*

function type	function title	total experience (years)	years in role
developers	senior developer	7	
	senior developer	6	4
	senior developer	6	1
architects	solution architect	7	5
	enterprise architect	8	8
	enterprise architect	8	5
	senior technical architect	16	9
	solution architect		7
	technical architect	13	3
	senior technical architect	11	8
	senior technical architect		9
	lead architect	10	3
project managers	project manager	8	4
	project manager	11	2
	project manager	9	4
	project manager	12	8
	project manager	11	1
	project manager	9	4
	project manager	12	8
	average experience	*11*	

meeting accepted. In addition, in relationship-oriented cultures, it is easier to obtain interviews by means of a personal introduction. An excerpt of the relation between the respondents involved in this study is depicted in Figure 5.1.

5.3 Theory Building

In this section, we discuss the factors that play a role in the process of software architecture design, coordination and dissemination. Based on the concepts that were identified from the labels and their respective narrative as discussed in the three cases, we arrived at the grounded theory depicted in Figure 5.2. Three main drivers were discerned that (eventually) negatively influence project success in terms of schedule and budget overrun:

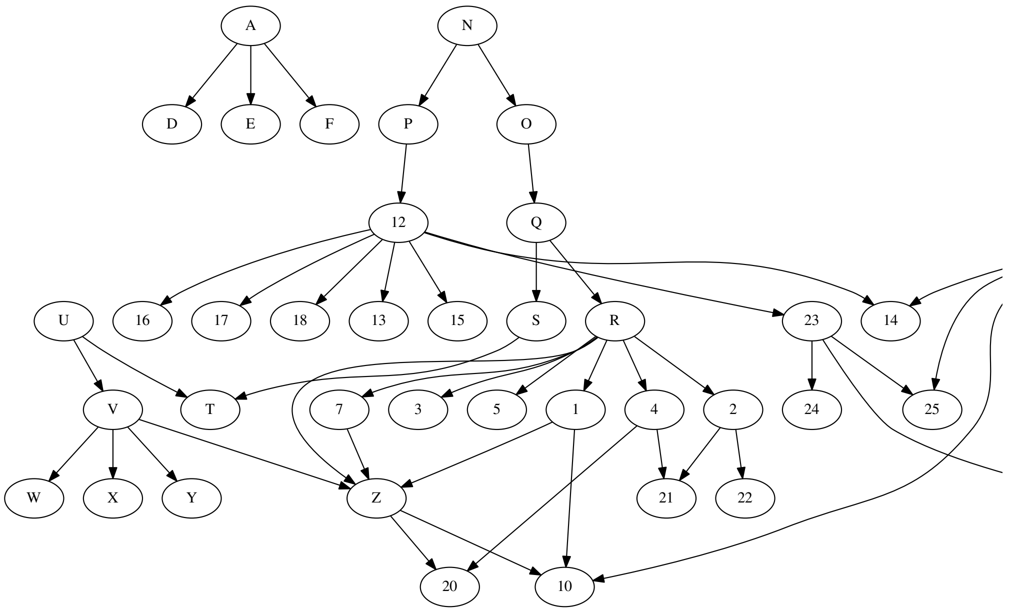

Figure 5.1: *Excerpt of (anonymized) chain-referral sampling graph of interview respondents*

1. The strong **implementation focus** of software development project management prematurely forces projects into the construction phase.
2. A **knowledge gap** exists between the onshore and offshore location regarding software architecture and its role during the software development life cycle.
3. **Cost reduction** forces a move of responsibilities towards the offshore software development location. This compounds the "knowledge gap" problems as less resources are available for knowledge improvement (training) and more work is required of less experienced team members. In addition, the added value of activities related to implementation is more tangible than that of design-related activities. As a result, the "'implementation focus" problem is aggravated.

In the following sections, each of these main drivers will be discussed.

5.3.1 Cost Reduction

Cost reduction is the most important driver for offshoring software development work (e.g. Šmite et al., 2010, Carmel and Agarwal, 2001, Ebert and De Neve, 2001). However, it remains to be seen whether cost reduction benefits materialize. Ample evidence suggests that GSD increases development cost (e.g. Conchúir et al., 2006, Herbsleb et al., 2001, Ebert et al., 2001, Espinosa and Carmel, 2003). Nevertheless,

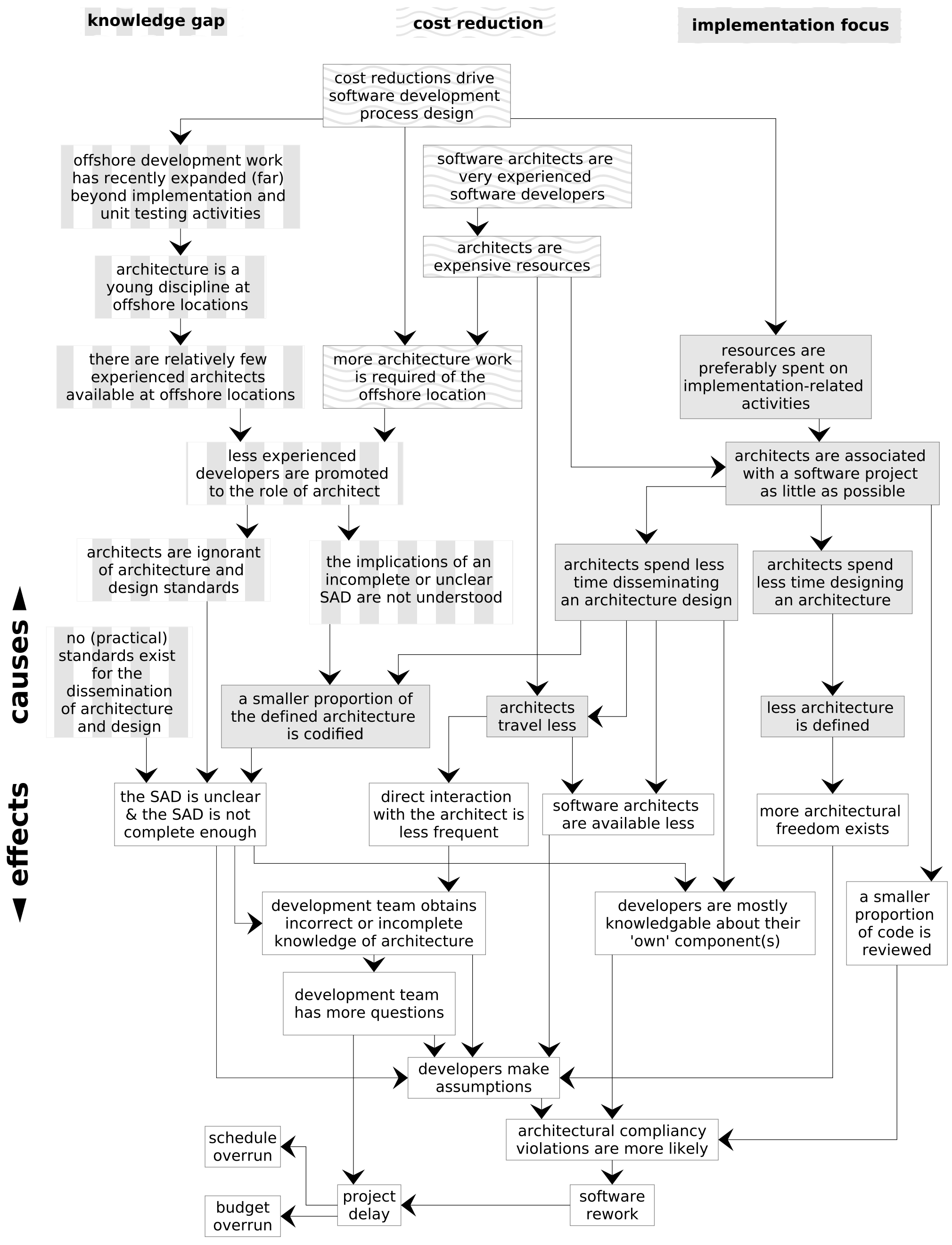
knowledge gap
cost reduction
implementation focus
cost reductions drive software development process design
offshore development work has recently expanded (far) beyond implementation and unit testing activities
software architects are very experienced software developers
architects are expensive resources
architecture is a young discipline at offshore locations
there are relatively few experienced architects available at offshore locations
more architecture work is required of the offshore location
resources are preferably spent on implementation-related activities
less experienced developers are promoted to the role of architect
architects are associated with a software project as little as possible
architects are ignorant of architecture and design standards
the implications of an incomplete or unclear SAD are not understood
architects spend less time disseminating an architecture design
architects spend less time designing an architecture
no (practical) standards exist for the dissemination of architecture and design
a smaller proportion of the defined architecture is codified
architects travel less
less architecture is defined
causes
effects
the SAD is unclear & the SAD is not complete enough
direct interaction with the architect is less frequent
software architects are available less
more architectural freedom exists
development team obtains incorrect or incomplete knowledge of architecture
developers are mostly knowledgable about their 'own' component(s)
a smaller proportion of code is reviewed
development team has more questions
developers make assumptions
architectural compliancy violations are more likely
schedule overrun
budget overrun
project delay
software rework

Figure 5.2: *Concept relation graph (⟶ denotes cause and effect)*

we find that the development organization from which cases A, B and C were derived, strives towards minimizing onshore project involvement. As software architects generally are experienced software developers, they tend to be expensive resources. Offshoring software architecture activities is therefore thought to potentially yield sizable cost reductions. The result of this insight is that more architecture work is required of the offshore location.

5.3.2 Knowledge Gap

In earlier projects, before the year 2005, only coding and unit testing work was entrusted to offshore development teams. The onshore location was responsible for requirements engineering, high and low-level design, integration testing and deployment. A recent development is that organizations require projects to offshore more of these "onshore activities" because of perceived potential development cost reductions. Gradually, unit testing, low-level component design and regression testing were added to the portfolio of activities that is commonly offshored.

Consequently, software architecture is a young discipline at this organization's offshore location in particular and at offshore development locations in general. There are therefore relatively few experienced architects available at offshore locations. To be able to meet demand, less experienced software developers are promoted to the role of software architect. An onshore architect typically has over ten years of software development experience. Offshore developers with just two years of experience have been promoted to the software architect role. As a result, offshore architects are aware of architecture design standards to a lesser extent.

5.3.3 Implementation Focus

Software development projects are limited by resources. Allocation of these limited resources over the various software development life cycle phases and activities is a well studied topic. Simulations are used to study the dynamics of resource allocation (e.g. Kellner et al., 1999) and theoretical models have been proposed to optimize resource allocation (e.g. Yiftachel et al., 2011).

Data from industry shows that resources are preferably spent on implementation related activities. Significantly more effort is spent in the construction phase of industrial projects than prescribed in theoretical models (Heijstek and Chaudron, 2007, Yang et al., 2008). Literature does not elaborate on the reasons for this phenomenon. We found a similar tendency towards coding-related activities at the expense of design activities in cases A, B and C. For example, the management tendency to push to move to the implementation phases as soon as possible (and limiting time spent on architecture design and dissemination) is referred to by various team members from cases A and B, as WHISCY or "**WH**y **IS** no-one **C**oding **Y**et?".

Cost reductions often limit an architect's association with a project to the architecture design (elaboration) phase. The focus on implementation shortens that design phase and thereby shortens even more the time an architect is involved with a software project. As a result, architects spend not only less time designing an architecture but also less time on disseminating that architecture design. The direct implications are that less architecture is defined and less architecture is codified. In addition, architects get less opportunity to travel to the offshore location.

5.4 Implications

We found the main problems caused by the drivers discussed in Section 5.3 to be:

- an *unclear and incomplete* software architecture document (knowledge transfer problem),
- that software architects are *available less* time per project (both a knowledge transfer and a control problem),
- that *less direct interaction* with the software architect is possible (both a knowledge transfer and a control problem),
- that *more architectural freedom* exists for developers and (control problem),
- that *less code is reviewed* (control problem).

These problems lead to incorrect and incomplete knowledge of the software architecture. Consequently, developers make assumptions and are mostly knowledgeable about their "own" components. Therefore, software architecture compliance violations are more likely to take place. This, finally, leads to software rework which in turn causes project delays in terms of schedule and budget overrun.

5.5 Recommendations and Best Practices

This section outlines a set of recommendations for coordination and dissemination of software architecture design in the context of GSD. The recommendations are structured as follows: First, the general recommendation are derived from the three main drivers that were discussed in Section 5.3. Second, a set of best practices is discussed that is distilled from the three cases analyzed in Chapter 4.

5.5.1 General Recommendations

First, *architecture should be recognized to be a first-class development concept*. All participants agree that architecture adherence is important. However, often developer knowledge of architecture is incorrect and incomplete and limited resources are allocated to checking architecture compliance during development. The added value of activities related to implementation is more tangible than that of design-related activities. This is an illusion. To adhere to non-functional requirements, extensive rework is often needed towards the end of a project. Therefore, making architecture a central concept (as is at the core of RUP) this type of rework can be largely prevented.

Second, *offshore software developers need to be trained to increase their understanding of software architecture*. The offshore shortage of software architects leads to limited understanding of the importance of software architecture.

Third, *a process needs to be in place for both initial dissemination of architecture design as well as the feedback process that follows it*. In the next section, we will describe some best practices regarding dissemination of architecture design as well as architecture implementation-related feedback. Such best practices should be institutionalized in a process so that common pitfalls may be avoided.

Fourth, *increased allocation of resources in (1) architecture design and representation and (2) guidance during the implementation phases from the onshore architect, is likely to lower budget overrun*. Conversely, we found, limiting onshore design time and architect availability results in budget overruns. Determining how many additional resources should be spent on software architecture design, representation and dissemination is not straightforward. As a guideline, in traditional (non-agile) development projects, between 70 and 80 percent of the architecture should be designed and represented when implementation commences. All architecturally significant use cases must be addressed in that initial architecture design.

5.5.2 Best Practices

Best practices are categorized in architecture design development, architecture design dissemination, the SAD, the architecture feedback process and architecture compliance.

Architecture Design Development

One strategy to ensure that the offshore team is more knowledgeable regarding software architecture design is to write the SAD together with the offshore technical team lead. For both cases A and B, in hindsight, the architects found that the SAD was not mature enough to be transferred to the offshore team. For a typical custom software development project, the majority of the architecture should therefore be stable before the construction phase. To this end, the “biggest mistakes” have to be removed in the elaboration phase. At the very least a POC of the architecture should be made during the inception phase. Investing some extra time into maturing the architecture and creating a more detailed SAD pays off during the construction phase in terms of fewer comments required during code reviews. This also amounts to fewer defects and less required rework.

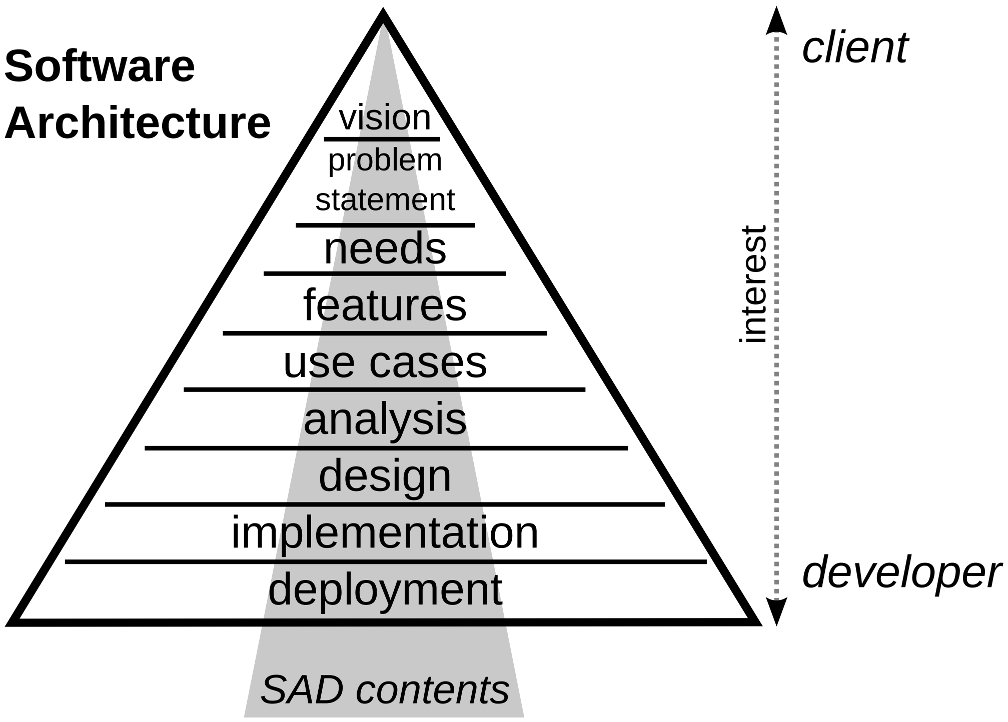

Figure 5.3: *Contents of an SAD and associated stakeholder interest*

Various parts of the same SAD address different audiences (Figure 5.3). A more technical stakeholder is generally interested in the bottom levels of the pyramid whereas a client would be more interested in the top layers. Apart from merely mentioning the information related to each layer, the key to a high quality SAD is the traceability between the layers.

Architecture Design Dissemination

The majority of respondents from all three cases agreed that every onshore architect should visit the offshore team at least once, to transfer in person the SAD and the explain its contents to the offshore development team. One developer explained that it is important for the architect to explain in person the SAD to a development team so that all developers, *"understand the dept or importance of specific requirements — this is something you cannot do from a document alone and it prevents that the solution moves slightly in a different way."* A slightly cheaper solution is to have at least one senior developer of the offshore team to travel to the onshore location during the initial software architecture development phase.

Successful dissemination of architecture requires that multiple communication channels are used to disseminate a single message. In addition to the SAD, the use of personal video conferencing sessions, telephone, e-mail and synchronous messaging should be used to repeat that same message. The frequency of use of these tools should be high. The objective for an architect is to give a lot of guidance after the SAD has been introduced so that all principles are properly understood.

Case C (Chapter 4) used a method which they called "continuous verification." The essence of the method is that short bursts of design information are sent offshore — mostly by means of video conferencing. The offshore team is then asked to summarize the design information sent and to report back to the onshore team. This way, a verification can take place if the design information was sent and received as intended. A by-effect is that the onshore team members need to carefully phrase their design information and that the offshore team members need to listen well and ask critical questions about what is unclear to them.

The iteration approach used in Case C requires a lot of interaction between team members in general and the onshore and offshore locations in particular. However, this strategy requires a client that is able and willing to have (very) frequent meetings throughout the project.

The SAD

As elaborated on earlier, SADs needs to both be clearer and more detailed than they were, found in e.g. cases A and B, because less contact between the development team and the architect takes place in a GSD context. However, an SAD that is too thick is less likely to be read in its entirety. The use of a common template for architecture representation is seen a beneficial. It requires architects to codify information they might otherwise have omitted. The use of UML is not seen by architects as essential for clear architecture representation. Architects argue that one, *"should use what gives most clarity."* However, that architects seem to take intended audience(s) into limited account and that offshore developers do seem to share a preference for UML. It might therefore be beneficial to take the use of UML for architecture representation into consideration. At least a legend should be used to avoid unclarity about the meaning of specific elements used in box-and-line diagrams.

Architecture information should be codified before and during the construction phase. For project-based software development, organizations, budget quickly "disappears" after a system is delivered. In addition, significant architectural drift (Rosik et al., 2010) takes place during the construction phase. Moreover, as employee turnover is relatively high (particularly in Indian development organizations), team members and their knowledge regularly disappear from a project. Therefore documentation in general and the SAD in particular should be kept up-to-date during the project.

Architecture Feedback Process

Almost all developers in cases A an B complained about delays due to the unavailability of software architecture information. Somebody with intimate knowledge of the software architecture should therefore always be available albeit not necessarily physically. This is difficult to explain to project leaders, given that cost reductions are a strong driver for GSD and that software architects are expensive resources due to their seniority. In addition to the trip that an architect should make to explain the SAD, the architect is recommended by some respondents to again travel to the offshore location during the first code reviews session. About this trip, the offshore project leader from Case A explained that, *"this provides a motivation for the developers — it tells them that somebody cares for them, sits besides them and helps them resolve issues."* To create a feeling of working in a single team, a method was used in Case C to make video conferencing sessions more like actual conversations — such as the ones at a "local coffee machine." Team members made it a habit to not get directly to business at the start of such a session: *"Each video conference we spend about ten minutes talking about private things."* In addition he notes that it is important to enrich the conversation with information that is contextual such as *"funny clothing that someone was wearing or [perhaps] spend some time talking about a mistake you made."* The goal of these habits is to build up a relationship like you would with local team members.

Architecture Compliance

Less complex architectures are less difficult to disseminate and are more easily adhered to. For architectures implemented in a GSD setting it is therefore even more pertinent to limit component coupling and increase component cohesion. For very complex architectures, the cost reductions that GSD potentially offers are likely to be offset by the time it costs to represent and disseminate architecture design and attain architectural compliance.

5.6 Conclusions and Future Work

Knowledge codification has the potential to mitigate some of the problems that are the result of the distances that GSD introduces. The opportunity to develop quality documentation, however, is limited. Cost reductions, a focus on implementation-related activities and a general lack of knowledge about software architecture lead to poor architecture design, coordination and dissemination. The lack of knowledge regarding software architecture is a genuine problem in the sense that it is a hurdle to be overcome if GSD projects are to make architecture a central concern. However, cost reductions at the expense of architecture design, coordination and dissemination as well as the tendency to give priority to implementation-related activities, provide a stark contrast with the claim made by all respondents that software architecture is such an important aspect of software development.