Chapter 4

A Multiple Case Study of Dissemination and Coordination of Software Architecture Design in Global Software Development

In this chapter an analysis is presented of software architecture dissemination and coordination practices in the context of large scale, global software development. To this end, a theoretical framework for the software architecture dissemination and coordination process is outlined. Using this framework, the software architecture dissemination and coordination processes of three cases of global software development are analyzed.

4.1 Introduction

Software architecture is an important artifact by means of which non-functional software qualities such as security, maintainability, extendability and portability can be addressed and guaranteed. Software architecture is disseminated to software developers in different ways depending on project characteristics such as project size, software complexity and the technological and organizational maturity of the client. Two main strategies to architecture knowledge dissemination can be discerned:

- A personalisation strategy, where, *"knowledge is closely tied to the person who developed it and is shared mainly through direct person-to-person contacts"* (Hansen et al., 1999).

- A codification strategy, *"centers on the computer and where knowledge is carefully codified and stored in databases, where it can be accessed and used easily by anyone in the company"* (Hansen et al., 1999).

Software architecture is generally disseminated through a mix of both these strategies. This is referred to as a *hybrid strategy* to knowledge sharing (Desouza et al., 2006). In traditional waterfall-approaches, this hybrid strategy will have a large element of codification. In agile (Highsmith and Fowler, 2001) approaches, the personalisation strategy is generally more dominant. In both cases, architecture documentation plays a central role. Software architecture documentation facilitates stakeholder communication and is instrumental in ensuring that essential design principles are adhered to by the code. Developer understanding of software architecture and its rationale is believed to be beneficial for software quality. Most development methodologies prescribe creation of a document that holds software architecture information. For example, in RUP, a methodology widely applied in industry, architecture is central and captured in a Software Architecture Document (SAD) for which a detailed template is used. This template is structured around Kruchten's 4+1 layer view on software architecture (Kruchten, 1995). While we are aware that this document is very often created in RUP projects, we are unsure what role such architecture documentation has in the software development practice. For example, how much of developer knowledge regarding a software architecture stems from this software architecture documentation as opposed to knowledge obtained from other sources such as colleagues? The advent GSD complicates informal personal communication in general and dissemination of software design in particular. A common approach to GSD is a type of "transfer by development stage" (Mockus and Weiss, 2001) where requirements gathering and analysis and design activities take place at a different geographical location compared to where the implementation work and unit testing activities are carried out. Complicating matters further, offshore developers are often unable to directly contact members of the architecture team due to geographical separation. Synchronous communication is often difficult due to time zone differences. Even if an architect can be contacted, communication can be hampered by socio-cultural differences and language barriers. GSD effectively mitigates informal communication while this type of communication is generally seen as an important element to disseminate architectural knowledge. This chapter reports on three case studies conducted to elicit architecture dissemination practices and the role of software architecture documentation in sizable GSD projects.

This chapter is structured as follows: The study objective is discussed in Section 4.2. Subsequently, related work and the research methodology are addressed in Sections 4.3 and 4.4. Cases A, B and C are discussed in Sections 4.5, 4.6 and 4.7, respectively. Conclusions and Future Work are outlined in Section 4.8.

4.2 Objectives

Understanding the processes and practices used for dissemination of architecture in GSD projects yields potential benefits in terms of more unequivocally conveying design decisions fundamental to a software system. Better understanding of a system's intended architecture by developers is of manifold importance. It directly benefits a project in terms of compliance to functional and non-functional requirements and thereby avoids expensive rework. It also benefits the structural integrity of the software in terms of aspects such a security, maintainability and portability.

This chapter addresses **RQ1** (Section 1.3). In Chapter 2 we analyzed software development process descriptions and in Chapter 3 we analyzed industrial instances of software development processes. In this chapter, we use a more detailed, qualitative approach to analyze industrial instances of software development processes. We aim to understand the process of software architecture dissemination in the context of global software development and therefore pose the following research question:

> *How is software architecture coordinated and disseminated in large, industrial, custom, global software development projects?*

As knowledge dissemination takes place via multiple methods we need to define sub-questions:

1. How is software architecture design and dissemination organized?
2. How is software architecture documentation used?
3. What is the role of the architect(s) during the software development life cycle ?
4. How is architecture compliance organized?

To this end we conducted three case studies of large, industrial custom GSD projects for which the architecture design was made in the Netherlands and the implementation was made in India.

4.3 Related Work

In this section we specifically address related work on the role of documentation in software develop processes, software architecture understanding and knowledge transfer in GSD.

4.3.1 The Role of Documentation

Work by Huysman and Wulf (2005) outlines that people have a preference to utilize personal networks over electronic networks to obtain knowledge in the context of

sharing experience. However, as mentioned, software architecture documentation plays a role in development methods of different natures. Studies reporting on the use of software documentation during software development in general are few and report mixed results. In studies analyzing developer preferences regarding documentation, Forward and Lethbridge (2002) and Lethbridge et al. (2003) find a preference for simple and powerful documentation and conclude that documentation is an important tool for communication, even if it is not up-to-date.

Already in earlier case studies, such as a study by Walz et al. (1993), we find a general reluctance for creating documentation. In general, documentation is seen as distracting from the actual work of developing software. The popularity of agile development approaches attests to this in part. Because the second of the four main "commandments" mentioned in the agile manifesto (Highsmith and Fowler, 2001) reads that, *"working software [is valued over] over comprehensive documentation,"* agile development methodologies are generally seen as document-light. As a result, popular agile derivatives such as Scrum (Schwaber and Beedle, 2001) prefer direct communication over documentation (Abrahamsson et al., 2003, Dybå and Dingsøyr, 2008, Clear, 2003, Rubin and Rubin, 2011) and therefore tend to be interpreted (and applied) as documentation-averse (Stettina and Heijstek, 2011a). While in their manual for Scrum software development, Pries and Quigley (2010) emphasize that, *"the scrum approach is not document-averse but, rather, seeks a leaner solution to formulating the requirements for the product,"* adopting an agile approach to software development often implies that little to no documentation is developed. In fact, agile development is described as an antonym for so called "document-driven" software development (Sillitti et al., 2005). In one of the few contributions to understanding of documentation in agile projects, Clear (2003) points at the behavior of students and observes documentation being perceived to be something external. Instead of being produced in-line with the system as natural part of the development process, documentation was often hurriedly pieced together at the end of a project.

The small role that documentation plays in agile development is underlined by Dybå and Dingsøyr (2008). In their thorough structured literature review on evidence on agile software development practices, they find that documentation is addressed in just one of the identified studies. So little documentation is apparently used in agile projects that in an international study of agile teams, practitioners noted that they found that documentation was important but that too little of it was available in their projects (Stettina and Heijstek, 2011b). This ambivalent position towards documentation might be well explained by Parnas' observation that, "*The solution is neither to add more documentation nor to abandon documentation — it is to get better documentation,*" (Ågerfalk and Fitzgerald, 2006). In fact, as noted, even code-centric, lightweight methodologies advise to create a form of architectural documentation (Smith, 2001).

4.3.2 Software Architecture Understanding

Software architecture can be complex and empirical studies show that developers generally understand mostly "their own" specific components of the application (Curtis et al., 1988). Reasons for a lack of understanding of the software architecture as a whole can be time constraints, poor documentation or a high turnover of staff. These factors can lead to absence of a culture of collective code ownership (Nordberg III, 2003), which is desirable to attain because it yields benefits in terms of software quality and team building. Also, documentation of the architecture is essential for maintenance activities which typically involve different engineers from the ones who developed the system (hence non-verbal transfer of design is needed). Software architecture, *"captures and preserves designer intentions about system structure, thereby providing a defense against design decay as a system ages, and it is the key to achieving intellectual control over the enormous complexity of a sophisticated system"* (Hofmeister, 2000). Existing literature fails to confirm that architecture documentation is used in the maintenance phase (de Souza et al., 2005).

In addition, the effectiveness of using UML to disseminate software architecture design is not yet clear. UML is a commonly used language for software design representation. In their standard work on documenting software architectures, Clements et al. (2002) provide strategies for applying UML for representing various architecture constructs but also note that for e.g. the "component and connector construct", *"all of the strategies exhibit some form of semantic incompleteness or mismatch."*

4.3.3 Knowledge Transfer in GSD

Dissemination of software architecture design is thought to benefit from frequent informal communication, preferably with those who have intimate knowledge of that architecture. Practitioners also feel the need for informal communication (Schneider et al., 2008) and informal communication has been found to be very important (Herbsleb and Grinter, 1999a, Damian et al., 2007, Nielson, 1998), particularly in organizations with fast-changing environments (Kraut and Streeter, 1995, Galbraith, 1977). Even though various solutions for distant informal communication such as instant messaging and video techniques (Fish et al., 1993) exist, Herbsleb et al. (2001) found that informal communication is very different for local versus remote site communication and that people at remote sites are found to be difficult to contact. The use of collaborative tool sets seems to mitigate some of the negative impacts introduced by geographical, temporal and socio-cultural distances (Nguyen et al., 2008). However, these tools are not always implemented or used to their full extent. Codification strategies (Hansen et al., 1999) for sharing software architecture information (Babar et al., 2007) are therefore likely to be more commonly used in GSD settings. This places demands on the clarity, completeness and consistency of software architecture documentation.

4.4 Research Method

In this section, the method for this study is described. This study is reported on in two chapters. This chapter contains a multiple case study. In Chapter 5, we present a synthesis of the factors involved in software architecture dissemination and coordination in the context in GSD as derived from these cases as well as recommendations and distilled best practices.

4.4.1 Data Collection

We applied data source triangulation (Stake, 1995) by collecting data by means of several methods. We employed software repository mining to obtain project data such as functional size estimation, cost structure, project planning and information pertaining to team member time registration. The software repositories furthermore gave us insight in the contents of the requirements, software architecture and design documentation and supplementary specifications. These repositories often contained useful information regarding team communication in the form of forums, discussions on defects reports or change requests and saved e-mails. In addition, we conducted a series of on-site, structured and semi-structured interviews with team members who were or had been active in each one of the projects. For these interviews we spent several weeks in India. We obtained a copy of the SAD which we reviewed. We then interviewed the primary architect, located at the onshore location, asking for:

- the coordination of the software architecture process
- methods employed for dissemination of software design and architecture within the team
- the role of the architect(s) in the development life cycle and the processes
- clarifications regarding the SAD
- reflection upon the above mentioned topics

In the next paragraph we explain the justification for the interview topics in more detail.

4.4.2 Interview Design

We developed a theoretical framework that is based on the "correction system" described by Shannon and Weaver (1949). This framework is depicted in Figure 4.1. In this theoretical framework, we depict our perception of the relation between an architect and a developer in the context of dissemination of software architecture. We structure the framework around the formats or states in which software architecture exist. These states are:

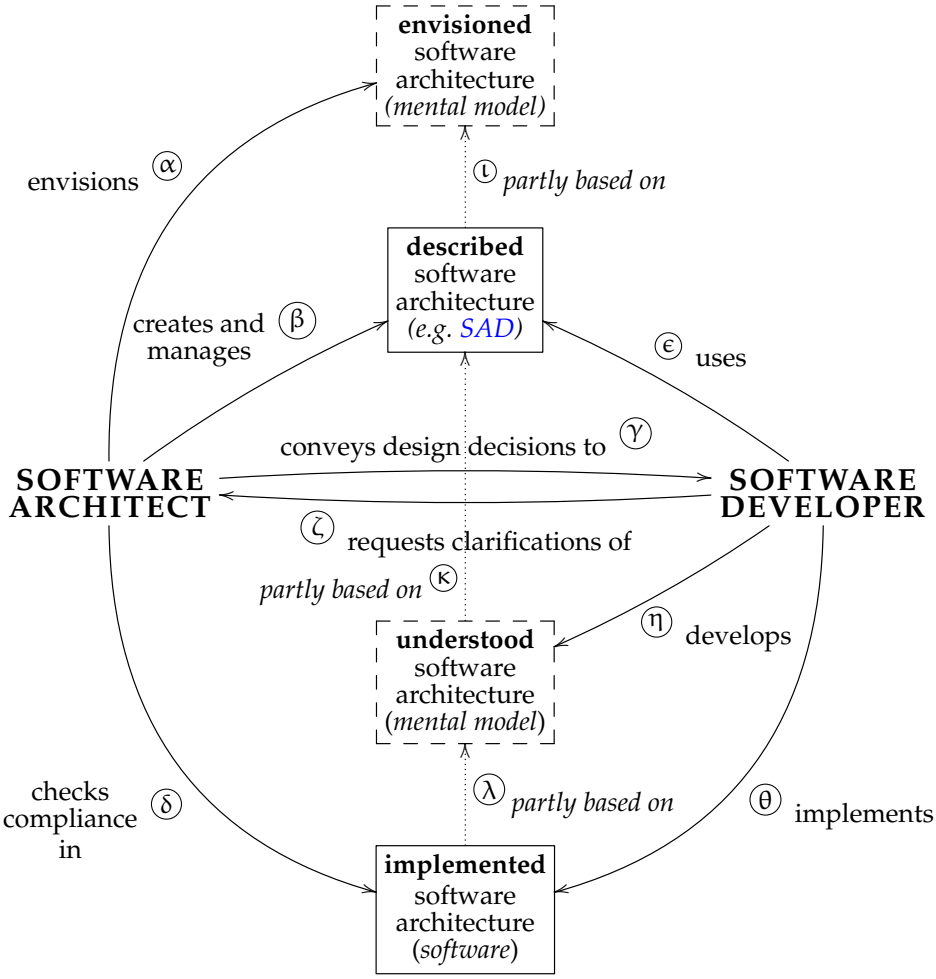

Figure 4.1: *Theoretical framework of software architecture design dissemination*

1. **envisioned architecture** — a *concept* that takes the form of a mental model that exists in a software architect's mind

2. **described architecture** — an *artifact* that takes the form of an architectural representation (such as found in an SAD)

3. **understood architecture** — a *concept* that takes the form of a mental model that exists in a software developer's mind

4. **implemented architecture** — an *artifact* that takes the form of software

The interview questions were targeted to the relations between the concepts and the artifacts. An overview of the question topics related to each link is depicted in Table 4.1. In addition, we asked the architect (1) which parts of the architecture document were important to understand for all developers and (2) to explain some of the most fundamental design decisions with which every developer should be familiar. Project managers were questioned regarding project characteristics, on topics on which they were knowledgeable and we verified information obtained in interviews with other team members. Interviews were taken in closed meeting rooms and recorded. All interviews took place face-to-face and we traveled to India for several weeks to be able to conduct extensive interviews locally. An overview of the team members we interviewed can be found in Table 4.2. Interviewees were asked for their permission for the audio recordings. The recordings were treated confidentially. The average interview length was approximately 60 minutes.

4.4.3 Data Analysis

For analysis, we employed two different techniques: (1) We followed the principles of grounded theory (Strauss and Corbin, 1990) and (2) an adapted form of shared mental model measurement. The use of these methods is described in more detail in the following sections.

4.4.4 Grounded Theory

Audio recordings were transcribed and labeled as follows: We summarized each point that the subject would make into a single sentence. Often participants use multiple sentences to convey a single point and sometimes multiple points are made in a single sentence. These points would be separated into multiple sentences. Transcriptions were done in such a way that each sentence in the transcription would contain a single point in such a way that the sentence could be properly understood in isolation. This one sentence would then be tagged by means of one or multiple tags. The use of these tags is inspired on their use on Twitter[1]. For example:

> *"In my experience, described software architecture, the SAD, is little adhered to by offshore development teams* `#sad_compliance` `#sad_relevance.`*"*

Advantages of using this text-based method are the reliability and modifiability of the data and the abundance of tools that can be used to analyze the data. We employed GNU Emacs[2], GNU Bash[3] and a wide selection of "GNU utilities", all stable and highly customizable, open source and cross-platform tools. For example, a continuously

[1] http://twitter.com/
[2] http://www.gnu.org/software/emacs/
[3] http://www.gnu.org/software/bash/

Table 4.1: *Interview Questions as Derived from Theoretical Framework (Figure 4.1)*

link	core question	interview topics
ⓐ	How does a software architect decide on a software architecture ?	**(out of scope)**
ⓑ	How does a software architect decide what to describe in an SAD?	architecture design process, intended audience, usefulness of SAD, templates, use of text, UML and box-and-line diagrams, perceptions of SAD quality
ⓒ	How does a software architect disseminate architectural knowledge (to developers)?	dissemination process, perceived understandability of architecture and clarity of SAD, factors in understandability
ⓓ	How does a software architect verify architectural compliance?	compliance methods, compliance in practice, factors in compliance
ⓔ	How does a developer use an SAD?	perceived role of SAD, consultation frequency during the project lifecycle, developer input to SAD
ⓕ	How does a developer request clarifications?	dissemination process, feedback coordination
ⓖ	How does a developer arrive at an understanding of SA?	communication methods, perceptions of SAD quality
ⓗ	To what extent does a developer implementation adhere to the intended software architecture?	compliance measurement methods
ⓘ	Which parts of the envisioned architecture are noted in the SAD?	use of templates, architect preferences
ⓚ	Which parts of the SAD are understood by a developer?	prescribed developer knowledge, mental model measurement
ⓛ	To what extent are developers able to implement the intended software architecture?	**(out of scope)**

updated overview of all labels currently used in all interviews, including a count of how often they appear, can be obtained using the following simple Bash script:

```
while true;
do clear;
cat /interviewfiles/*
| awk '{for ( i = 1; i<=NF; i++ ) if ( $i ~ /^#\w/ ) _[$i]++ } END{ for ( i in
    _ ) printf "%s (%d)\n",i,_[i]}'
| sort -u
| pr --columns 4 -w 200 -l 80;
sleep 2;
done
```

For traceability purposes, a separate text file for each transcription was used. No more than two interviews were transcribed per day to avoid mistakes due to fatigue. After transcribing all case-related interviews, double labels were removed. After correction for typographic errors, 132 labels remained. Of these labels, 57 were only used once. We then used grep[4] to extract and group all sentences related to a single label like so:

```
grep -r "#label" /interviewfiles/case_a/*
```

This grouping provided us with a comprehensive view on each label. We could then proceed to identify the main concepts related to the label. In grounded theory, this is referred to as reduction. This resulted in the following seven different categories:

- case-specific problems and generalizability
- architecture development process
- architecture dissemination and clarification process
- software architecture document
- architecture compliance
- shared mental model deviations
- best practices

4.4.5 Shared Mental Model Analysis

Shared mental models (SMMs) provide teams *"a common set of expectations that enable accurate and timely predictions of approaching needs and issues"* (Cannon-Bowers et al., 2001). Holt (2002) describes software architecture as a mental model shared among

[4] http://www.gnu.org/software/grep/

the people responsible for software. SMMs have been found to positively impact work-team adaptability and performance (Cannon-Bowers et al., 2001, Mathieu et al., 2000). There is little empirical evidence regarding how SMMs affect coordination in more asynchronous and geographically distributed collaboration (Espinosa et al., 2001). To address SMMs, in all interviews, we addressed the following set of topics:

- General project characteristics (e.g. *How far along is the project?*)
- The role of the architects (e.g. *Who is responsible for architecture compliance?*)
- Important design decisions in the system to be build (e.g. *What are the most important design decisions in the architecture?*)

In the following sections, we present and discuss the case findings, structured along these categories. The last step of grounded theory, integration of the concepts by means of induction, will be discussed in Chapter 5.

4.4.6 Case Studies

Three case studies of global, custom software development projects were executed. Each project was executed by the same global IT service provider. Each project had a team in the Netherlands and a team in India. All clients were Dutch. A summary of relevant project characteristics as well as the interviewed team members can be found in Table 4.2. The onshore organization (the front office) is responsible for the project and hires developers in the offshore organization (the back office). The back office is a so-called "cost center" which means that it is paid for the amount of hours that they work. The front office takes the risk for the project in that they take the loss when a fixed-price project goes over budget. Conversely, they take the profit when a project is a success. The following three sections contain the three case analyses.

4.5 Case A

The goal of the project that makes up case A is (1) the expansion of an existing system for indexing and (2) for making information of various governmental organizations searchable. The existing system held in the order of millions of documents and processes millions of queries per month. The new system was to replace the old and was to introduce some new functionality such as moving the responsibility for the (technical) presentation of search results to the client and capabilities for separate management of knowledge models and thesauri. The interfaces to the existing information sources were to be maintained. The system is structured around Microsoft SharePoint (Table 4.2). The system was to be delivered as Software as a Service (SaaS, Papazoglou and Georgakopoulos, 2003). The application is to be maintained by the same organization that is building the system. Functional maintenance, however, will be the (organizational) responsibility of the client.

Table 4.2: *Case Characteristics*

	Case A	**Case B**	**Case C**
client domain	government	private	private
functional size	34 use cases	70 use cases	800 function pts.
plan. duration	10 months	3 months	4 months
process meth.	RUP	RUP	Agile / Scrum
budget	€ 800,000	€ 210,000	€ 400,000
offshore dev's	6	5	4
technology	.Net + Microsoft FAST Search	.Net + Microsoft Office SharePoint Server 2007	.Net
GSD type	*transfer by development stage* (low level design & implementation in India)	*transfer by development stage* (low level design & implementation in India)	*transfer by development stage* (low level design & implementation in India)
project objective	expansion of an existing system for indexing and making searchable information from Dutch government organizations	centralization of a human resources portal for a large, multinational industrial firm	rebuild of an existing Visual Basic 6 application
team members interviewed onshore	delivery mgr. architect project mgr. arch. reviewer test lead	delivery mgr. architect arch. reviewer	delivery mgr. project mgr.
team members interviewed offshore	project mgr. #1 project mgr. #2 architect developer #1 developer #2 developer #3 developer #4	project mgr. sr. developer developer	sr. developer developer

4.5.1 Case-Specific Problems and Generalizability

We first sketch an overview of the main problems that were encountered during the development life cycle:

- For this project, a hard deadline and a fixed budget were agreed upon with the client. Time pressure was high because the strict required date of delivery was overly ambitious according to project leaders and other team members at both the onshore and the offshore location.
- No proof of concept (POC) was built for the solution for this project due to time and budgetary constraints. Various offshore team members referred to how useful a POC would have been to better understand the solution and therewith prevent certain delays.
- Requirements were changed late during the project. Specifically, an authentication system for all external interfaces was required.
- The system under development was to embed a proprietary, external search component for which external training from a third party was required. The training was difficult for team members as not all requirements regarding this component were clear at the time of the training.

With regard to the external validity of any insights we obtain from this particular case, we note that late requirements and the problems regarding a lack of a solid design and POC that arise from schedule pressure, are not at all uncommon. On this matter, offshore or back-office architect (BOA) noted that this project, *"was a typical high-time pressure project."* The front office architect (FOA) remarked that the problems that he encountered during this project are very similar to the problems he encountered in other GSD projects in which he either worked or heard about. The offshore project leader insisted that the project adhered to internal quality regulations and that it scored a three out of three on quality audit.

4.5.2 Architecture Development Process

Originally, the offshore team was to deliver the project's software architect in order to save costs. A FOA was intended to be associated with the project part-time to function as a coach for the BOA. This BOA visited the onsite (or: client-) location and had three weeks to develop an architecture. However, the onshore project leader and FOA decided that the quality of the work of the BOA was insufficient and that the FOA was to finish the architecture. The offshore team presented a different view on why the BOA was not able to create an architecture during his time in the Netherlands: *"The time I spent in [onshore] was not enough because of the use of an external component that was completely new for us. In addition, the learning time that was planned during the bid phase was*

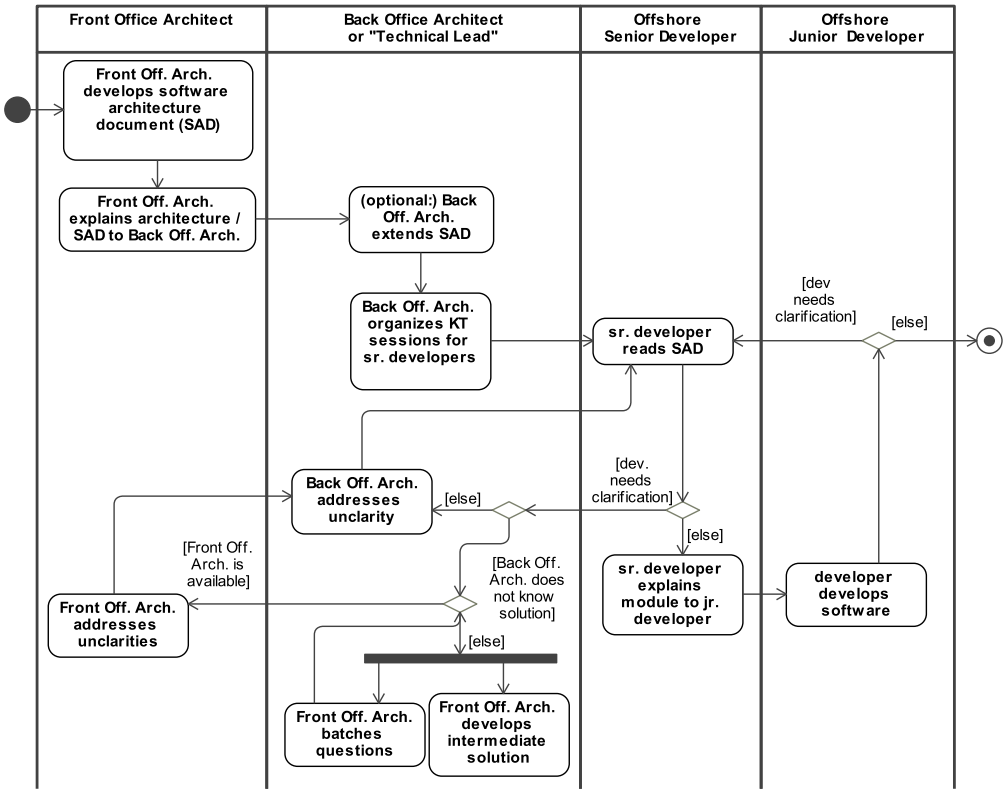

Figure 4.2: *Communication of the software architecture design from the onshore to the offshore location*

not enough." (BOA). The Dutch side of the project explained this as the incapability of the architect to create an architecture. The BOA and the offshore project leader explain that, *"the lack of knowledge of [a specific search engine] was also a problem as we had to design an architecture with a black box in the middle."* It is difficult to make out which team's narrative is right. In any case, with the shift of architecture responsibility, some of the tasks that were originally intended for the offshore team therewith moved to from India to the Netherlands. However, no extra budget was allocated to the FOA for these tasks.

4.5.3 Architecture Dissemination and Clarification Process

The process used to communicate the architecture from the FOA to the development team at the offshore location is outlined in the activity diagram in Figure 4.2.

Initial Architecture Communication Process

The architecture design was initially communicated from the FOA to the BOA and the development team by means of this SAD and two knowledge transfer sessions. The first of these sessions took place between the FOA and the BOA by means of a video connection, right after the delivery of the SAD. The BOA then held a knowledge transfer session with the development team, in knowledge transfer session, the BOA gave a high-level overview of the system. Subsequently, the development team obtained the SAD and then a second session took place with the BOA to discuss details and answers of developers and to explain how to proceed. The offshore project leader noted that the FOA was never involved in these sessions, he only talked to developers later, by means of video-communication sessions based on reviews of their code. A senior developer noted that, his previous projects were similar in terms of dissemination of architecture. A developer describes that, *"the architecture was discussed in a session, then we read through the documentation and came back for questions."* About this second meeting, another developer remarked that, *"in the SAD not all diagrams were written down using UML, that provided us with some difficulties but because we had [a] meeting with the FOA, he could explain to us what he meant [in those diagrams]."* A developer noted that, *"for most knowledge we needed, the knowledge transfer sessions we had were [. . .] enough."*

Feedback Process

The BOA works as a proxy between the FOA and the offshore developers. This situation was said to reduce the time that would otherwise be claimed of the FOA. However, the limited knowledge of the architecture of the BOA rendered him a message proxy. One developer would explain that, *"[our] team lead is too busy to properly communicate software architecture decisions — he has no time for coaching."* As a result, developers lack a deep understanding of the system's design. According to the offshore project leader, regular video meeting sessions directly between client and offshore team, took place. These sessions were not meant for the development team as the client was not seen as technically mature enough to talk directly to developers. A developer noted that, *"in my previous projects, we would have direct contact with our clients in the US — that is much better."* Codified knowledge does seem to play a major role in this project. One developer noted that, *"for a problem, I will first use the documentation and then talk to either the BOA or the FOA."* The FOA's perception of the team interaction on the offshore location is that, *"the technical team lead in India is less communicative with his team of developers than I am with him."* The FOA suspected that information got lost in the communication to developers through the BOA. As an explanation, the FOA suggested a lack of time for explanation or reviewing or a general lack of expertise on the part of the BOA. However, regarding the meetings that took place between the FOA and the BOA, the FOA claimed that *"[he] never said that he did not understand something."*

Requirements Maturity

A senior developer noted that, *"often, we worked more than 12 hours,"* and explained that, *"that had to do with unclarity regarding the functionality that the client requested."* As noted in Section 4.5.1, this project experienced problems with late changing requirements. The client was not the only source of late requirement changes as another example of late requirement changes resulted from poor requirement analysis. The BOA explained that:

> *"In the bid phase, one requirement pertaining to the reading of an input string was formatted as a simple one-line statement which later turned out to be a small project in itself — namely the construction of a parser."*

The requirement problems can partly be attributed to the level of technical maturity within the client organization. A developer described the client as, *"totally functional, not technical."*

4.5.4 The Software Architecture Document

All team members agree that the SAD is, *"very important,"* because, according to a developer, *"it is the base of the software and when it is in place, it is useful to a developer [...] with it, better software is made."*

The supplier organization provides an SAD template that contains extensive guidelines based on the RUP SAD template. Regarding this guideline, the FOA noted that, *"the SAD [of this project] was based on [this] template."* The FOA was critical of this template, though. He noted that he finds it allows too much freedom. While some sections of the SAD template had been filled out by the BOA, almost the entire SAD had been written by the FOA. The BOA explained that, *"there was little involvement from the client with the architecture."* According to the FOA, the software architecture template that is prescribed by his organization does not prescribe how to disseminate a software architecture design

In the next sections, we will discuss the purpose and the intended audience, the use and content of the SAD, perceptions on its quality and additional software architecture documentation used in this case.

Purpose and Intended Audience of the SAD

Regarding the purpose of the SAD, the FOA noted that, *"in essence,"* the client does not read the SAD. He added that the SAD audience is inherently a technical one and should have a basic level of technical knowledge to understand the SAD. Surprisingly, we found a rather extensive use of the Dutch language in certain UML models in the SAD. All use cases and the conceptual architecture models were written in Dutch. When confronted with this observation, the FOA explained that the use of Dutch was

easier for client communication. The FOA noted that, *"translations are done in India"* and pointed out that, there exists a list with translations in the SAD. When interviewing the offshore development team, we did find this language issue to be a hindrance to the developers. A developer noted:

> *"another problem that we have is a language problem, for example in the SAD, parts were in Dutch, we use Google Translate to translate that text to English and if Google is wrong, that would be a problem."*

The development team did not seem to be aware of why Dutch was used in the SAD. One developer noted that, *"[the SAD] was in Dutch because the FOA was Dutch."* Answers regarding the choice of languages in the SAD were polite but hinted at frustration. A developer said, *"although the FOA speaks both English and Dutch, he used Dutch to create certain SAD diagrams[. Still,] it is not his job to translate Dutch to English."* The question who is responsible for translations remained unanswered. Another developer explained that, *"I would have preferred to have the documentation in English."* A senior developer noted that *"we cannot use Google Translate or a professional translator because of the technical nature of the project — we, however, [were forced to] use Google Translate."* The FOA said that, even if he was not pressured for time, he would still prefer to use Dutch for the parts of the SAD *"so not to confuse the client with different terminology."* The FOA noted that he is sure the BOA understood the SAD.

Use of the SAD

Developers said that they refer more often to the functional requirements described in use case documentation, than to the SAD. The available documentation is said to be a first point of reference for the developers who explicitly note they consult documentation before they ask the BOA or FOA. There seems to be a difference in the perception of when the SAD is used. For this particular project, the FOA noted that he did not expect the SAD to play a big role during the construction phase of the project. Yet architecturally significant components such as a locking mechanism, were added during the very late construction phases of the project. And the FOA explained earlier that, *"the SAD helped the developers to understand the [architecture layers], the different ideas in the presentation layer and the web services."* As a result, the offshore developers noted that the SAD played a major role during the construction phase of this project. For every new construction iteration, the requirements are gathered for another external connection of the system. The SAD is then needed to understand the flow of information to and from this new connection. Some developers explicitly noted: *"I even use the SAD later in the development stages."* Developers thus expected much more from the SAD than the FOA aimed to communicate with it. This is a mismatch in the perceived role of the SAD.

SAD Contents

Problems regarding the SAD that were mentioned include a remark by the offshore project leader who noted that he found lacking a clear description of *"the interaction with the external parties."* After analyzing the SAD, we observed that only a part of this model was visible. This implied that without having access to the digital source of the image, not all diagram elements in the component model could be seen. According to the FOA, this had to do with the manner in which the image was exported from the modeling tool, IBM Rational Enterprise Architect. As a solution, the FOA advised that the model source file could be opened with this tool. However, the development team lacked the licenses to use the modeling tool.

Most diagrams in the SAD were non-UML ("box-and-line" types of) diagrams. No legend was used for most of these diagrams. The offshore project leader noted that he preferred the use of UML in SADs because he found it to be *"a universal language."* Also, most developers explained that they preferred the use of UML in the SAD. One developer directly linked the limited use of UML in the SAD to project problems: *"In the SAD not all diagrams [were modeled using UML], that provided us with some difficulties but because we had two to three meetings with the FOA, he could explain to us what he meant with that."* Regarding the freedom the FOA took to develop the SAD we also found that certain protocols are described in more details than others. The FOA said that he choose himself which protocols prescribed in the SAD warranted further explanation. Regarding the choice for mentioning and explaining design patterns in the SAD, the FOA noted that he assumed that they were understood and that, *"some were particularly addressed."* An external architecture reviewer, however, noted that he found that the employed design decisions were not properly, if at all, described in the SAD: *"The design patterns in [this] SAD are not described in enough detail and seem to have been added just to show that some thought went into design patterns rather than to be used as a guide for developers."*

Perceptions of SAD Quality

Whether an SAD is good enough for a developer to use is influenced by what a developer is accustomed to. One developer noted that, *"compared to my other projects the documentation for [this project] was very good, everything was very well described in detail and that is needed when you need to build a low level design based on it."* Other developers noted that, *"this SAD is one of the better ones I have seen,"* and, *"the SAD in [this project] is better than the SADs I had to use in previous projects."* In contrast, a senior developer noted that, *"I have seen better SADs than the one we use in [this project]."*

One developer found that the incompleteness of the SAD led to a greater involvement of developers in the design of the architecture:

> *"Previous projects in which I was involved usually did not involve so many external components and were less complex, The SAD used in [this project] therefore is much*

better than the SAD in [my] previous projects because we were much involved in its creation and therefore we know how the architecture has evolved to its current form to understand the architecture much better."

The desire for increased inclusiveness in software architecture design is a recurring theme in all three cases and will be discussed in more detail in the discussion section.

Additional Architecture Documentation

During the third construction iteration another design document, the *Supplementary Specification* (SS) was created to capture the late requirements and their impact upon the architecture. According to its definition, the SS artifact *"captures system requirements that are not readily captured in behavioral requirements artifacts such as use-case specifications"* (Kruchten, 2003b). The SS is normally prescribed to be used as input for the architectural analysis activity that leads to the SAD, in the inception phase. However, in this case, the SS is used as an update of the SAD. The SS contained information about logging, authentication, authorization and performance. The sources for the SS were the existing SAD and the Service Level Agreement (SLA) made for the project (which was to be delivered as a service). The FOA summarizes the rationale of the SS as follows: *"One of the goals of the SS is to prove to the client that we did certain things [because] in essence, they do not read the SAD."* A senior developer noted that, *"the supplementary specifications are created for the maintainers."*

4.5.5 Architecture Compliance

The non-functional requirements that an architecture addresses can only be guaranteed if an architecture is implemented according to plan. In this project, however, the FOA complained that, *"the developers think it is good enough when their work resembles the architecture,"* and explained that, *"as an architect for an offshore team, you have far less control over this."* A senior developer noted that, *"code should fully adhere to the architecture,"* and that even, *"when you find that some wrong decisions were made earlier and the architecture should then be readjusted,"* still, *"the architecture document should be leading."* When, during the interview, confronted with a choice between working code and architectural compliance, often developers would initially state that adherence to architecture is more important than working code. When pushed, developers would often admit that in practice, this plays out differently. Says one developer: *"this is somewhat diplomatic as when you find the architecture is not correct, you change the architecture and then align the code with this new architecture."* An architecture reviewer noted that an important reason for creating an SAD is to *"cover your ass."* In this sense, an SAD serves as an overview of non-functional requirements that where addressed that a project team can show to the client. In this case, not the code, but the documentation serves as a proof of work.

Architecture Understanding

Regarding developer knowledge of the SAD, the BOA explained that, *"knowledge was divided up per [external system] that was added — at least two people were knowledgeable per [system]."* This confirms findings by Curtis et al. (1988) who found that developers mainly understand "their" component. For example, when one developer was asked about certain parts of the architecture, he replied, *"I work with "the pushing scenario" so my knowledge is mostly confined to that functionality."*

Code Reviews

The FOA is seen by the offshore developers first and foremost as the person who knows how the system should work. To ensure that the system meets set requirements, developers must obtain information from the FOA. The FOA, in turn, needs to check the extent to which that information is understood. This check is done by means of code reviews (Figure 4.1, line δ). Code reviews are explained to be the only means of checking whether design decisions have been adhered to. Code reviews are prescribed by the development process to be first done by fellow developers, second by a senior developer (peer reviews), third by the BOA and fourth by the FOA. In this project the FOA did not have enough time to review all code. In total, he reviewed approximately half of all code. Due to resource constraints and tight schedules, the development team did not have enough time to perform code reviews. For example, no code reviews took place during the first construction iteration. Not all developers agreed that the code review process was adhered to strictly. Some developers referred to the *"incidental internal code reviews"* that would take place at the peer review level. A developer was dissatisfied with the code review procedures and noted:

> *"I would say that in this project the development team had less understanding of the architecture and of what to do than we had in my previous projects as there, a code review would be automatically done at check-in time, so we would immediately get feedback line-by-line, at that detail and those were mistakes you would not make again."*

Here, a connection between code review procedures and understanding of software architecture is made. Developers also imply that stricter code reviews would infer that their work or at least the quality thereof is important. Developers did not have a feeling that all their work was checked in detail. A developer noted, *"the FOA does not do code reviews on any scheduled time, when he had time he reviewed code and provides review comments more regarding architecture compliance than naming conventions."*

4.5.6 Shared Mental Model Deviations

We found the following differences in perceptions between team members in this project. First, we found differences regarding the software architecture process:

- Developers feel that they only interact with the FOA regarding software architecture matters. The FOA does not share this view: *"[more than half of my time I spend on] implementation related problems that are not concerned with the software architecture."*
- Some developers saw the BOA as the main source of information concerning software architecture while others saw the FOA in that role.
- There were great differences in how the development process was perceived. For example, some developers claimed that significant time losses occurred due to waiting for the onshore location while others claimed that, *"we never had to wait for something in particular from the front office."* This can be explained by the fact that not all developers were involved in discussions with the front office.

Second, we found differences regarding the artifacts created in the development process:

- Offshore developers found that they completely adhered to the design principles while the onshore team members did not share this view at all. (Developer: *"The code as it exists now is completely compliant to the SAD."* Another developer: *"currently the implementation is fully architecturally compliant."*)
- Some developers noted that the SAD was complete, some said it was not complete at all. This could be explained by the respective components that these developers "owned." One developer's component might have been more completely described in the SAD than another developer's component.

Third, we found differences regarding the project as a whole:

- Some developers remarked that the project was going to be delivered on time and that the system was almost done. The project was delivered 6 months too late and at the onshore location, the team members were much better aware that this would happen.
- The project is deemed to be a success by some offshore team members and *"very problematic"* by others. Location did not play an important role in the divergent opinions regarding this topic. The onshore team members consistently mention poor architectural compliance as an important reason for the project failure.

The FOA explained that every developer should at least have an understanding of the design patterns and the SAD chapter in which the "implementation view" (Kruchten, 1995) was outlined. This chapter contains information on design packages, the component model, process flows and architecture, naming and modeling guidelines. In each interview, we asked two questions pertaining to this content:

- *What are the most significant design decisions in the architecture?*

- *What are the most important parts of the SAD?*

Very different answers to this question were given:

- FOA: *"The most important design decisions in [this project] are the application of pushing and the use of chain of responsibilities [...] another important decision is the use of a manager so that the system is so flexible so that we can easily add a new connection."*
- Developer: *"The most important design decisions are the functional and non-functional requirements of the client and those decisions pertaining to the external systems that we are using and also the infrastructure."*
- Developer: *" The web service part was the most important part of the architecture."*
- Developer: *"Another important aspect of the architecture was that the flow of the system has to be able to be changed as easy as possible."*
- Developer: *"Other important aspects of the design are logging and exception handling because we have seven external connections."*
- Developer: *"The use of web services was another fundamental architectural design decision that allows us to not tightly couple the connections to the system to allow easy extendability"*

These answers always corresponded to the element of the system with which the developer was concerned. The FOA concluded that he, *"would give developer knowledge of the architecture a 6 (just enough) but there is a clear difference between the quality of individual developers."*

4.6 Case B

The objective of this project was to integrate various Human Resources (HR) systems into a single system. To this end, 70 use cases were implemented, 50 of which were electronic forms that were custom built in Microsoft InfoPath. The team members in this project were all different people from the team members for cases A and C. One exception is the offshore project leader who was briefly involved in the inception phase of case A. The client was a large, multinational industrial firm. Key characteristics of this case are summarized in Table 4.2.

4.6.1 Case-Specific Problems and Generalizability

Again, we first sketch an overview of the main problems that were encountered during the development life cycle of this case.

- This project dealt with changing requirements from the client. For example, many of the InfoPath forms that needed to be developed were already designed when the use cases changed a lot. Most of the form development then had to be redone. In addition, the system's architecture was impacted by these changing requirements.

- Significant delay was introduced due to difficulties related to deployment of the developed software. Various issues were explained to be the cause for these problems by different team members. The FOA noted that, *"the offshore team claimed that it worked in their environment,"* but that, *"they had little regard for the fact that we had to implement it in another environment."* A developer claimed that the cause for the problems was poor collaboration with the front office. He explained that, *"we cannot do deployment on our own as it takes place outside of our vision — we cannot visualize or imagine why things don't work in the production environment — what has been missed, why it is failing."* The end result was that the development team was not able to package their software in such a way that it could be delivered to the client. The front office therefore packaged and deployed the software.

- The development was severely hampered by three experienced developers leaving for other companies. The offshore project leader explained that SharePoint experience was a skill for which employers were willing to pay salaries that were 20 to 30 percent higher. Often changing employers (partly due to being offered such raises) is not uncommon in the competitive and fast-growing software development sector in India. According to the architect, when they knew they would leave, these developers were not very motivated anymore. As a result, they did not properly hand over their work. In addition, not enough information was codified. A senior developer explained: *"When the previous senior left for another company he gave the project a week notice, I came in two days before he left and on those two days he was very busy arranging his departure from the company so not much knowledge transfer happened in that time."* The FOA: *"this cost us a lot of expensive onshore resources."*

As with Case A, regarding the external validity of any insights we obtain from this case, we note that changing requirements and the problems associated with changing team composition in the offshore development team due to people switching companies, are common. The FOA noted that, *"the problems that we had in [this project] were typical."*

4.6.2 Architecture Development Process

As for the previous case, the intention for this project was to have the architecture created by the offshore team. The architecture responsibility was offered to the back office. Before the project commenced, however, the offshore organization made clear

that they wanted the front office to develop the architecture. The reason seems to have been a lack of experienced architects available at the time. About the availability of developers that are able to create a proper software architecture in the back office, the architect was very clear: *"[Some people in our organization will have you believe that] if you just create a use case model, a set of non-functional requirements, some instructions about what it has to do, storyboards and some screenshots, [the back office] should figure out by themselves how to create sub-systems and decide how to build it — but we have found in past projects that that capacity is not available, also not with the seniors — who are good technical specialists but setting up an architecture is just not what they do."* In contrast, the offshore project leader made clear that, given availability, *"there is no reason why an Indian architect cannot talk to a Dutch client from India."*

There was little time available to create an architecture. The architect explained that, *"often clients have an idea of the solution that they want to have implemented and ask us to implement that solution — we then get little time to check such an architecture — this rarely leads to something good and often we need to rework the architecture during the implementation phase."* The offshore project leader expounded on her contradicting view by explaining that, *"I don't feel that the quality of the architecture suffers under grave budget constraints — perhaps the SAD suffers but I have seen good architecture frameworks made under pressure."* Not so for this architecture, though, as she continued to explain that, *"the architecture [that was] made [was] not very mature."* The initial lack of maturity of the architecture was confirmed by developers as they complained that they had to rework a large batch of InfoPath forms as the use cases and architecture were fundamentally changed during development. This problem could partly be attributed to the requirement changes described earlier.

In conclusion we observe that the architecture was not ready when it was communicated to the back office. Another mistake that was made here, was that all 50 use cases were developed immediately while limited knowledge was available regarding the target technology, InfoPath. The offshore project leader: *"In the first few weeks of a project, we [normally] go very slow deliberately — we don't aim for the sky — In [case A, for example, they] started ramping up very slowly — first [they] let developers create a small set of use cases so that [they] could assess the quality of the architecture but also the quality of the people."* Lastly, the offshore development team was not satisfied with the architecture design. A prominent critique of that design was that it was not robust enough for the client requirements. A senior developer explained that, *"SharePoint has a limitation on the amount of entries in a list — that should be no more than 10,000 — in a single month, the client overrun this by 5,000 — this should have been known by the architect."*

4.6.3 Architecture Dissemination and Clarification Process

The process of communicating the architecture from the architect to the development team in the back office is very similar to the activity diagram that described the same process for case A (Figure 4.2). One difference is that no back office architect was

involved, but that the most senior developer took a similar role. The other difference is that the SAD was not extended by anyone in the back office. The knowledge transfer sessions took place by video conference and were only attended by the most senior developer — referred to as the "technical lead." The senior developer did not like the use of conferencing tools: *"communication on the architectural level should take place face to face because it is so fundamental."* The offshore project leader also complained about the architecture dissemination process and added that she did not know why the architect did not travel to India as she prefers that. The front office explained that the architect did not travel to India due to budgetary constraints. The offshore project lead responded by noting that that would have been an investment that would surely have paid off.

Developer Hierarchy

Similar to case A, the front office complained about the use of a single point of contact to the back office team. The architect complained that they, like in case A, had no say in the composition of the offshore team and that as a result, *"the developers were [. . .] not very visible to us from the front office — we got a point of contact offshore, but not a list of developers."* He continued to explain that, *"in bigger projects, Indian teams make use of a senior developer as a proxy and this does not work very well as the senior developer will understand you but [not] the junior developers."* One of the problems that resulted from this was that the front office had no influence in work distribution strategies. They noted that work was poorly distributed. The architect: *"A substantial part of [this project] consists of electronic forms which can thus be set up in a similar way - however, these forms were distributed by the senior in India and three different developers worked on these forms [and they were so dissimilar that] we could see by the implementation of the form, who made it."*

The use of a senior developers and a technical lead as proxies for the rest of the development team was also found in case A. A senior developer from this project explained that *"junior developers should not be given or even know the architecture because it is too much pressure and they will get confused — they are still learning and architecture is too much of a challenge for them — juniors should just implement whatever they are told too in the manner that is told to them."* He continued to explain that, *"As a senior developer you are a sort of translator between the architect and the junior developer."* He added that this hierarchical nature of work distribution is commonly found in software projects. The offshore project leader defended in a different way the use of a technical lead and senior developer as a proxy. She noted that, *"[while] it might be partly rooted in [our] culture; as long as a development team is bigger than three or four people, I don't see any other way than to ask questions from junior to senior developer to technical lead to front office in order to reduce the workload and to avoid double questions."* There might be more than one way to avoid double questions. Also, a question that is asked multiple times might serve as a bellwether for possible unclarities or complexities in the architecture

design. Only when the more senior developers all left the project did the architect communicate to the developers directly. One developer noted that he liked this direct communication. The hierarchical nature of the team composition was still maintained, though, as even in this new situation, the most senior of the junior developers would note that *"the developers communicated directly the onshore architect but they would have less communication with him than I would have, because they would usually ask me a question first."*

Architecture Understanding

The clarification process, the steps that were taken when junior developers had questions, was similar to the description in Figure 4.2. A developer explained that, *"the SAD was provided to us and gone through with the technical lead — we then knew what was needed and did not use the SAD."* The front office noted that because the development team was not at all visible to them, they were not sure that they understood the architecture design.

We find that the offshore development team did not find understanding of the architecture and architectural compliance as important as the front office found it to be. The architect explained that, *"because of a lack of understanding in the Indian team, [I] was e-mailing whole chunks of self-written and self-found source code to outline what [we] meant — that is too much detail in my opinion."* He explained that this happened in all of the three offshored project in which he was involved as an architect. In contrast, a developer clearly outlined that he did not use the SAD as *"it was not so important for my own component."* This disregard for the relation that a "developer's functionality" has with the rest of the system, is a recurring theme in this case as well as the previous case. The architect referred to this problem as well: *"The mentality I always see with juniors in India is "I am here to execute a job, tell me what to build and I'll build it" — the idea that they also have to study the overarching structure is very rarely adopted."* As a result, the back office team admitted that only during the later stages in the project did they understand the software architecture.

The Notion of One Team

The notion of forming a single team, despite the geographical, temporal and socio-cultural distances is regarded as a best practice in literature (Lee et al., 2006) and in the case organization. The idea behind this notion is that communication technology is used to organize a team to simulate that it is co-located. In practice, however, it is not easy to accomplish "one team" as it mainly constitutes a feeling among team members — which is not straightforward to foster. For example, the architect explained that, *"we are supposed to be 'one team' but when problems arise such as a missed deadline, or who did not deliver what or are the specifications correct or hasn't something been tested well, and before you know it is one team versus the other."* He adds that, *"this feeling remains dominant in*

the rest of the project." As noted in the previous section, the development team was not at all visible to the front office, which made creating a single team a challenge from the outset. A senior developer put across that, *"when the architecture is not developed offshore, it is not so clear what the architecture rationale is — you are just asked to implement, this model, this model and this model — if we knew how these models came to be, we can also contribute our ideas."* This is in concordance with remarks made by various back office team members in case A. Developers seem to find it important to be included in the architecture design process. They feel more like full partners than in a situation in which they are just told what to do. This contrasts with the front office view that the backoffice team simply needs to be told what to do to be able to function as team members. This follows for example, from this quote from the architect: *"The SAD does not contain many motivations for the design decisions that it describes [. . .] we most often do include these decisions but not so much for India, as they do not care much but more for the client."*

4.6.4 Software Architecture Document

In this project, as in Case A, limited time was available to create the SAD upfront. According to the architect, the SAD was not written with offshore development in mind. A GSD SAD, he notes, should have more explicit diagrams for clarity. However, Case B's SAD is "too immature" to allow for such distinctions. In hindsight, the level of detail of the SAD was too low. Initially, the front-office team offered to model up to the class-level and to make sequence diagrams. The architect said that the offshore senior developer *"laughed and said that was really not necessary."* Now, the architect notes, it turned out that we had better done that. At the end of the project, the SAD no longer reflected the implemented architecture — this phenomenon is referred to in literature as "architectural drift" (Rosik et al., 2010). The architect noted that he, at the time of the interview, still intended to (rudimentary) update the SAD.

Role of the SAD

The architect found that in GSD, it is important to make things very explicit. The SAD is an important means towards this end. The offshore project manager likened the SAD to be a *"semi-contractual document"* in relation to the client. Still, the architect is unsure about the role of the SAD in practice. He stresses that he thinks it is "very important" for a developer to understand the overarching structure of the architecture and the place of his own part of the system in that whole. He nevertheless adds that he observes that junior developers do not read the SAD. Apart from the hierarchical nature of the organization, he feels that an SAD might be too voluminous and complex for most developers in the offshore team. As a result, he notices, developers (at most) read the parts that are relevant to their component. From experiences in earlier projects, he notes that he found that visual materials, such as a clickable demo, work well. He

adds that documents are just not read when they are too thick. He admits that the SAD might be more important in GSD but that it is a very limited vehicle for dissemination of design. The role of the SAD is less that of the primary conveyor of design decisions and more one of a reference document. From that perspective, the architect notes that, *"a super-well written SAD will also really help a lot [towards successful GSD projects, but] mostly as a reference."*

SAD Quality

The offshore project leader found the SAD to be incomplete and of low quality. She explains that the SAD is incomplete even though she feels that, *"not all aspects of the SAD as it is derived from the RUP, are needed to be filled out to be able to communicate it from onshore to offshore."* She finds the quality of SAD depends both on an architect's experience with architecture and his experience with collaboration with offshore teams. She adds that Dutch SADs suffer from language issues: *"The English in the SADs written by Dutch architects is often weird."* Also, offshore team members complained about the quality of the SAD: *"The SAD is not as good as SADs we used in other projects,"* and, *"it was not up to the mark."*

4.6.5 Architecture Compliance

According to the architect, the implemented architecture only resembled the prescribed architecture. He found that understanding of the prescribed software architecture was low among developers. He particularly found that developers did not test their own work, did not keep to coding standards and did little to ensure that their components work with other components. In line with these observations, one developer explicitly noted that he found architecture unimportant: *"The system was not complex enough to warrant a big role for architecture."* As a result, architecture compliance was low. A senior developer remarked that the implementation was *"fully compliant to the architecture."* Another developer said that the system was approximately 80 percent compliant to the prescribed architecture. The architect maintains that it was much lower.

Architecture Understanding

The architect finds that developers focus more on appearances than on "the inner workings." He adds that the idea that the overarching structure must be understood, is rarely adopted. He illustrated his point as follows:

> *"The offshore team is positioned far away on an island, and we send them a package of paper — the SAD. That creates an attitude: "What we have build may be interpreted as what you specified in that SAD, so we are done." [In my opinion, this does not stem from cultural differences at all.] The exact same attitude would*

exist if you would send a Dutch team to an island to collaborate with another onshore Dutch team."

He added that according to him, the hierarchical nature of offshore development organizational culture did harm architectural compliance. Developers are, in his view, not encouraged to look beyond their own component. The architect: *"If I'd have to rate the knowledge that the offshore developers [in this project] had of the architecture I'd give them 6 out of 10."*

Code Review

Within the offshore team, the senior developers reviewed, as one explained, *"most of the code that the juniors write."* He explained that they specifically check whether, *"the layers that we want are used correctly."* Not enough time was available for full code reviews. Exceptions were made for very complex functions. In addition, a sample of the written code was reviewed by the onshore team. When mistakes were found, developers were told to repair similar problems for the entire code.

4.6.6 Shared Mental Model Deviations

When asked to describe the fundamental design decisions of the system architecture, the architect mentioned:

- The design of multi-language support
- Communication with external databases
- The interface with other external systems
- The translation of the client department workflow to the system

One developer noted that the most fundamental design decision were "long list policies." This was a design decisions specifically related to "his" components. A senior developer stated that, *"there are many important design decisions such as handling of the list, integrating InfoPath and integrating the company facebook."* Again, these components were that developer's "own components." I already discussed the discrepancies in the perceptions on the extent to which the implemented architecture complied to the prescribed architecture. The offshore team thought compliance was high to perfect while the onshore team thought the system was not compliant at all.

4.7 Case C

The objective of this project was to rebuild an existing Visual Basic 6 application in .Net. The size of the application was estimated to be approximately 800 function points.

Important characteristics of this case are summarized in Table 4.2. Compared to the cases A and B, this case had the smallest development team — only four offshore developers were involved. Where both cases A and B followed the RUP development process, case C used a distinctively agile approach. Another distinct aspect of this project is that a Model-Driven Development (MDD) approach was used to generate significant parts of the code.

4.7.1 Case-Specific Problems and Generalizability

The use of an existing MDD framework, implies that instead of one, multiple systems are being developed (Heijstek and Chaudron, 2010). The existing code generator needs the be developed in parallel to the system that is required by the client. One developer explained it by noting that, *"when we find a mistake while working with the framework, we fix it for the project, but also for the framework. This introduced extra effort."* In particular, at the start of the project, many things were changed to the framework, this led to many defects in the developed system and required extra effort to repair in both the generator and the meta-model of the system that was being built. In addition, not all developers were experienced with the MDD paradigm. As a result, at least one developer made changes to generated code. This code got overwritten after another generation cycle. He noted that he *"learned this the hard way."* This project had no problems with late requirements because, they say, of the weekly interaction with the client. Late requirements were expected and anticipated by using an agile design process (described in Section 4.7.2). This was possible in part because the complexity of this project was lower than that of cases A and B. GSD that employs MDD tools and techniques and that used agile development methods, is not very common. Growing bodies of literature exist on the application of agile methods in GSD projects (e.g. Taylor et al., 2006) and the integration of MDD in agile development approaches (Zhang and Patel, 2011). Studies on the application of MDD in a GSD (e.g. Heijstek and Chaudron, 2010, 2009) setting are rarer. Studies of the combination of all three paradigms are (to the best of our knowledge) unavailable in literature. The results obtained from analysis of this case are therefore hard to generalize and mainly serve to provide contrast to the more "traditional" cases A and B. This is a valuable addition to the analysis also because all three project were executed by the same organization.

4.7.2 Architecture Development

This project lacks a team member that was explicitly assigned the role of software architect. The project leader explains that there are relatively few complicated patterns in the system. In addition, the size of the project stems more from the quantity of the screens than from the complexity of the screens. The project had a start-up phase of three to four weeks in which scope was agreed upon with client and a high-level functionality document was made by using smart use case points. This document

contains functional and non-function requirements. During these weeks, the non-functional requirements where reviewed to see whether it was possible to use the MDD approach they used earlier without having to change it fundamentally. They found this was the case.

Model-Driven Development Approach

The project makes use of a home-grown MDD platform that had been used on several earlier projects. This platform is based on Model-Driven Architecture (MDA) and therefore uses UML models as input. The class and use case models must adhere to strict modeling guidelines. By making use of stereotypes that are related to classes and use cases, patterns are used to generate code. The UML diagrams are translated to the XML Metadata Interchange format (XMI, Object Management Group, 2007) which is in turn used as input for the code generator.

In previous projects that used this MDD approach, approximately eight hours were needed to implement a single function point. Although little data is available in literature, this is at least a factor 1.5 to 2 faster than comparable traditional projects (which do not employ code generation techniques)[5]. The project leader, who also led these previous projects, explained that for this project, they were even faster as most people in both the on- and offshore teams were involved in the previous projects.

Applying MDD tools and techniques is limited to applications that can be described using the defined domain language. A trade-off needs to be made between flexibility and code generation coverage. The less "custom" functionality (that is, functionality that cannot be described using the domain language) is required, the higher the proportion of code that can be generated. A developer explains this by noting that, *"In an earlier project, lots of business rules were needed, many security constraints and lots of complications – [our framework] would not have been useful for such a project."*

Design Development Process

Implementation, design and testing is done in short iterations that last one week. In each iteration, a predetermined set of functionality is designed. In the next week, this functionality is implemented. The week thereafter, this functionality is tested.

[5] In the dataset accompanying ISBSG Benchmark Release 9 (International Software Benchmarking Standards Group, 2006), the average amount of (adjusted) function points per (normalized) person-hour for all 1001 "new development" projects was 15.3 h/fp. Jones (2000) reports the average productivity of 1065 projects (of which 505 are new and the rest are enhancement projects) in the US was 14.3 (h/fp). The smaller the data sets are, the higher the productivity seems to be become: Premraj et al. (2005) report that the productivity of 401 projects completed between 1997 and 2003 in Finland to be 4.3 h/fp. Tsunoda et al. (2009) report the average productivity of 211 enterprise application development projects in Japan to be 6.7 h/fp. However, the company average (for the organization from which all three cases in this chapter are derived), calculated from a dataset of 51 software development projects, is 20.5 h/fp. In an (unpublished) dataset in our possession containing 40 software development projects executed at another large Dutch organization, the average productivity is 27.7 h/fp.

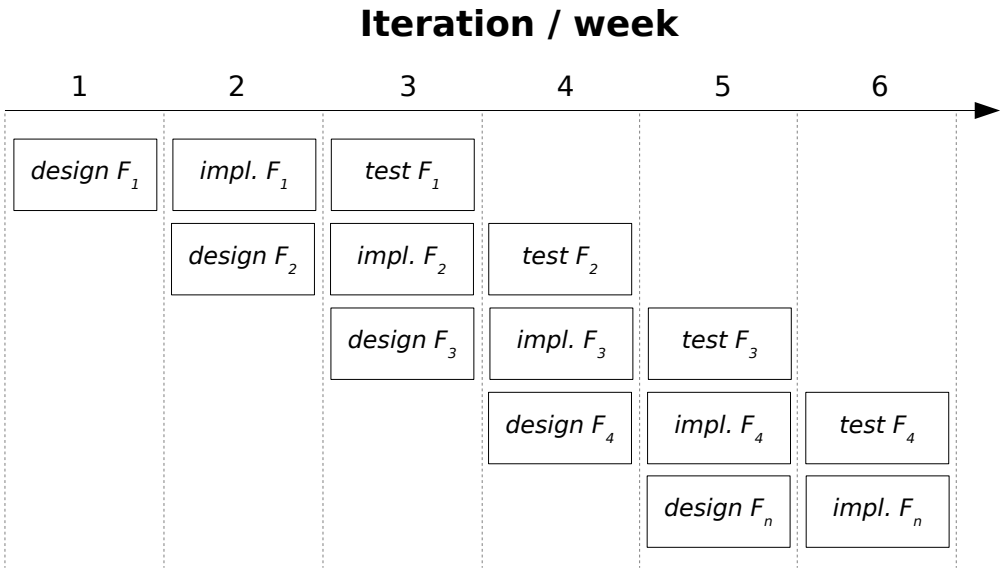

Figure 4.3: *Case 3 iteration strategy*

During every iteration (except the first two and the last two) one set of functionality is designed, another is implemented and yet another is tested. This strategy is depicted in Figure 4.3. This way, 17 iterations have taken place.

4.7.3 Architecture Dissemination and Clarification Process

Because of the use of MDD, the architecture used for this project is mostly defined by the design choices on the MDD framework level. The architecture is proven and mature as it was used for previous projects. The project leader therefore referred to the project as, "not such a high-risk project." Even though it is agile and that is because of the fact that the MDD approach is used, it has a proven architecture. In fact, this stable architecture is prerequisite to enable code generation.

Various interviewees involved in cases A and B were familiar with the MDD approach that was used by this project. The offshore project leader involved in case B explained that mature and reusable architectures such as the one used in case C, are easier for offshore development as it is clear how these need to be implemented.

Design Communication Process

Since a significant part of the architecture is already implemented in the framework that enables code generation, most design communication was related to lower-level design. The senior developer of the offshore team was involved in the weekly requirements gathering workshop by means of video conferencing. A design for implementation of

these requirements would be first made onshore and then disseminated to the offshore team. The project leader explained that from his experience with offshore GSD projects, he noticed that, *"too much is being 'sent' from the onshore team to the offshore team and it is checked too little whether the offshore team has understood everything correctly."* As a result, in this project a method was used that the team 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 whether 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 more questions. For all the benefits that frequent communication offers in this case, at least one developer experienced similar waiting problems found in cases A and B. That developer explained that not all developers are usually part of all communication calls with the front-office. As a result, until there is time to address one of that developer's problems, he has to make assumptions and, *"wait and when my assumption turns out to be wrong, I have to rework my solution."* Still, the developer preferred the communication methods used in this project, over the methods he used in the previous projects in which he was involved.

Agile and Model-Driven Communication

The offshore project leader explained why he feels agile practices aid in communicating design:

> *"I contend that agile is a better way of approaching GSD projects as daily communication between the shores means that you can't play "hide and seek any more" — in addition, because of the short iteration cycles, you are being forced to know every single moment what is happening — when you need to give a demo to the client every week, you will make sure that it works. Nobody likes to look like a fool."*

The project leader has experience in non-agile GSD projects and explains that, *"the main disadvantage of these big RUP GSD projects is that you give offshore developers a chance to disappear underwater without knowing what they do and without feedback loops - effectively creating more than one team."* There is, however, a limit to the size of the projects you can tackle with such intensive communication required. The project leader noted that for this 800 function point particular project, they were "stretching it".

The offshore developers were very positive about the use of MDD in particular. One remarked that, *"we were only two to three months into the project and most of the important parts of the application were ready."* The same developer found it much easier to communicate using a model as opposed to talking about code with colleagues and the client. This MDD-related benefit is found in other studies as well (e.g. Heijstek and Chaudron, 2010). The frequent interaction with the client was also regarded as useful.

A developer explained that the increased client interaction led to a reduced total defect count. As a result, the development team could focus on new functionality.

Team Composition

The project leader explained that he found many GSD projects, especially those staffed with many skilled and experienced technicians, are unsuccessful. He explains that, *"most people in onshore development teams are selected based on their technical skills while these matter to a lesser extent in GSD projects than communication skills, being able to delegate, having trust in other team members."* He explains that he staffs his onshore team with young people who are relatively inexperienced and who are willing to learn. In addition, dedicated meetings are planned in which both onshore and offshore team members are required to review the development process.

4.7.4 Software Architecture Document

As most of the architecture was defined in the MDD framework, no SAD was used. The project leader planned to have an SAD made after system delivery, for the client. Instead of using a custom SAD, developers have to understand the architecture of the MDD platform. Most team members had experience with the framework. One developer explained that he learned how the MDD framework works during a three week "on-boarding training" that was organized by colleagues that were already knowledgeable regarding the framework. He noted that he was not a very good modeler at the start of the project but that he learned quickly on the job. He said that the use of MDD made the development process more formal and that, *"there are many things we have to do in a specific way."* He found that this made things simpler.

4.7.5 Architecture Compliance

As the project had no designated software architect, the project leader asked an external MDD expert to do a review of the code at several points during the implementation of the system. Compliance of handwritten additions to the code was found to be good.

4.7.6 Shared Mental Model Deviations

No explicit shared mental model deviation could be found during the interview. Some disagreement existed as to the percentage of code that was generated versus the percentage of hand-written code. One developer reckoned it to be 30 to 50 percent of the code where another claimed it to be 60 to 70 percent of the code.

4.8 Conclusions and Future Work

The software architecture design and coordination processes are important factors that determine GSD project success. We found that two of the cases were seen as unsuccessful projects. In case A, the onshore and offshore team have very different perspectives on how successful their project was. In both cases A and B, onshore team members consistently mention poor architectural compliance as an important reason for project slowdown. Case C's team application of code generation techniques automated much of the architecture compliance. Much fewer project problems were reported and the project was seen as successful by both onshore and offshore team members.

The rest of the conclusion is structured around the sub-research questions posed in Section 4.2.

4.8.1 How is Software Architecture Design and Dissemination Organized?

The main finding is that *specifically the dissemination of software architecture does not seem to be formalized while this might benefit the development process*. The SAD seems to be typically created onshore. A cost-related incentive exist to have software architecture created offshore. This failed in cases A and B. The main reason given was the novelty of software architecture as a discipline and the resulting lack of experienced architects at the offshore location. After the architecture is codified it is sent offshore to a senior software developer. This senior software developer then acts as a proxy for work distribution for more junior developers. If architecture-related questions arise, junior developers are expected to ask their senior peers. Developers continue working based on assumptions while they wait for an answer. This leads to re-work and project delay. In addition, this proxy work method obfuscates the intentions of the onshore team. Vice-versa, it "hides" the offshore development team for the onshore team. Direct interaction between all team members is seen as beneficial for architecture understanding. It is, however, hard to organize in larger teams. In case C we find evidence that in an MDD context, when architecture plays a smaller role (because it is (1) proven, (2) not complex and (3) mostly already implemented to enable code generation), less problems arise associated with architecture dissemination.

4.8.2 How Is Software Architecture Documentation Used?

The main findings are that (1) *the SAD is intended to be used extensively but developers use the SAD sparingly if at all* and that (2) *while a template is used for development of the SAD, architects enjoy (and use) a large degree of freedom in choosing software architecture representation*. The SAD has multiple audiences, chiefly the developers and the client. A first version of the SAD is typically for the developers. An updated version of the SAD is created so that it covers all client requirements. Perceptions on what or who is

the main source for architecture information, differ widely. The onshore team find the SAD the main source of information while developers use the SAD to a limited extent. Some developers rated the SAD quality in cases A and B to be low. When used, the SAD does not seem to be specifically tailored for GSD.

4.8.3 What is the Role of the Architect in the Development Life Cycle ?

This question is harder to answer. The main finding seems to be that *the role of the architect in GSD is not clearly defined*. This role is therefore open to interpretation. From the previous question, it follows that according to the offshore team members, the architect is the main source of information regarding the architecture. In addition, the architect is also seen as a hurdle in terms of accepting developed source code. Architect travel was noted to be a best practice by most interviewees. However, in none of the cases, onshore team members traveled to the offshore location to coach the development team regarding the SAD.

4.8.4 How is Architecture Compliance Organized?

The main finding is that *while developers say that they find architecture very important, they seem to be mostly knowledgeable about "their own" component*. Code reviews are used to check architectural compliance. Developers and architects often disagree on whether code is compliant. The prescribed code review process is that (1) developers review each others code first, (2) that code goes to the offshore technical lead and then (3) to the onshore architect who should mainly look at architecture compliance. The onshore architect normally only does elaborate code reviews when there is a feeling that there is a problem. We find that architects do not find it part of their responsibilities to "micro manage" implementations to attain compliance. Interestingly, developers all note that they would like to be more involved in the design process at both the architectural and lower design level. Onshore team members experiment with offshore team member inclusiveness but are generally negative about this.