SOFTWARE PROCESS IMPROVEMENT AND PRACTICE
Softw. Process Improve. Pract. 2003; **8**: 233–247 (DOI: 10.1002/spip.186)

Architecture as a Coordination Tool in Multi-site Software Development

Research Section

Päivi Ovaska[1*,†], Matti Rossi[2] and Pentti Marttiin[2,3]
[1] *South Carelia Polytechnic, Koulukatu 5B, FIN-55120, Imatra, Finland*
[2] *Helsinki School of Economics, P.O. Box 1210, FIN-00101, Helsinki, Finland*
[3] *Nokia Technology Platforms, P.O. Box 407, 00045 Nokia Group, Helsinki, Finland*

A widely held understanding of coordination in software development is that it focuses on coordinating development activities to achieve a common goal. Our study, a case study in an international ICT company, suggests that in multi-site environment, it is not enough to coordinate development activities to achieve a common goal. Rather, more emphasis should be put on coordinating interdependencies between activities. Shifting the interest from activities (and subsystems) toward system-level dependencies requires software architects and developers to have a common understanding of the software architecture. Our findings reflect coordination challenges in multi-site environment with geographically dispersed teams. On the basis of the findings, we claim that architecture could be used to coordinate distributed development. However, this requires that the chief architect is capable of maintaining the integrity of the architecture and of communicating it. Furthermore, we list some requirements for a development methodology that uses architecture to support the coordination. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: multi-site software development; software architecture; coordination of software development

1. INTRODUCTION

Today, it is common for software development to take place in multiple or even distributed groups working together on a common system. Recent projections by the Gartner Group suggest that more than 137 million business users worldwide were involved in some form of remote work in 2003 (Finholt *et al.* 1998). Increased importance of knowledge, technological complexity, global competition and the availability of digital information and communication technology are driving the change toward global and networked environment (Castells 1996). Carmel (1999) has introduced a set of catalysts for global software teams. These include deployment of specialized and 'best' expertise, expansion through company acquisitions, reduction in development costs, visibility for the company brand with global presence, reduction in time to market and proximity to customers in high-interaction tasks such as requirements gathering and innovation.

* Correspondence to: Päivi Ovaska, South Carelia Polytechnic, Koulukatu 5 B, FIN-55120, Imatra, Finland
†E-mail: paivi.ovaska@scp.fi

In this kind of distributed environment, the management of uncertainty and distribution of knowledge become focal issues. As the developers have no immediate feedback and as they do not necessarily share the same development culture and values, the coordination and control of the work is paramount.

Coordination is an inherent aspect of work in any organization and takes place in the form of meetings, scheduling, milestones, planning and processes. Kraut and Streeter (1995) argued that coordination becomes much more difficult as project size and complexity increases. Apparently, complexity increases when the project is located in multiple sites. Communication is a salient part of coordination, and it has been observed (e.g. Allen (1977)) that distance affects the frequency of communication. Communication delays and breakdowns taking place in software development projects are discussed in several studies (Curtis *et al*. 1988, Kraut and Streeter 1995, Herbsleb *et al*. 2000).

Much of the research in coordination of software development assumes that coordination is coordinating activities to achieve a common goal (Curtis *et al*. 1988, Kraut and Streeter 1995, Grinter 1999, Grinter *et al*. 1999, Herbsleb and Grinter 1999, Carmel and Agarwal 2001). We go further and propose that in a multi-site development environment, it is not enough to coordinate only the activities but it is also important to coordinate the interdependencies between the activities to achieve a common goal. The main mechanism for coordination is the software architecture, which describes these activities and their interdependencies in terms of components and their relationships (interfaces between components) (Garlan and Perry 1995). This kind of coordination gives the developers in multiple sites a possibility to concentrate more on their own development work and of not having to be so aware of the work in other sites as long as the interfaces between components remain the same (Szyperski 1998). In the case of interface changes, the developers should coordinate mostly their changes, and not so much the whole component change.

To get first hand information about coordination issues in situ, we performed an in-depth study of a software development project of an international information and communications technology (ICT) company. We studied the use of architecture as a coordination mechanism in a multi-site development environment. Geographical, cultural and language distances between development participants were present in the studied organization. The focus of our study was twofold: first, we focused on coordination problems and tried to identify categories of processes that explained most of these problems. Second, we used these categories to identify differences between same-site and multi-site development environments.

The goal of the studied project was to develop a directory service platform for the global telecommunications market. To gain easier management, the project was partitioned into two subprojects on the basis of architecture and technology. The development work in these two subprojects was carried out in different ways, which gave us a possibility to make comparisons between these two subprojects. One subproject was executed in three different sites and the other in one site in Finland.

We used grounded-theory approach according to Strauss and Corbin (Eisenhardt 1989, Strauss and Corbin 1990) to investigate the importance of software architecture in multi-site coordination. An involved case-study approach according to Yin (1994) and Eisenhardt (1989) was used to gain insight into the project. The principal researcher of this study worked as a head of department in the company and participated in the steering group work of the project.

The rest of the article is organized as follows. First, we discuss coordination and architecture in software development literature and take a closer look at two coordination theories (Section 2). Section 3 introduces our case organization and the case project. The research settings and methods along with a project narrative are outlined in Section 4. Section 5 describes the findings of the study. Section 6 discusses and concludes the results, their implications for research and practice as well as topics for further study.

2. RELATED STUDIES

2.1. Coordination

Coordination issues in software development, especially in large software projects, have been identified as one of the main reasons for delays and budget overruns (Kraut and Streeter 1995). Plans, processes, interface specifications and software architecture are typical formal coordination mechanisms in large software projects. However, there is an abundance of research results, which show that informal, unplanned, ad hoc communication is also extremely important in supporting collaboration (Curtis *et al.* 1988, Kraut and Streeter 1995, Herbsleb and Grinter 1999). People need to communicate with each other on details and things omitted from the formal specifications, and their work has to be coordinated on the system issues (Kyng 1991). As Winograd and Flores (1986) observed, collaboration and communication exist in all human actions except for the simplest tasks.

2.2. Coordination Problems

2.2.1. *Same Site*

The formal communication mechanisms can be vulnerable to imperfect foresight and unexpected events, which require communication to coordinate activities (Herbsleb and Grinter 1999). The lack of informal communication channels can lead to problems in software development, which increases the development time. These problems can lead to misunderstandings in design conventions and rationale (Curtis *et al.* 1988) and to software integration problems (Herbsleb and Grinter 1999). As the size and complexity of software increases, the need for supporting informal communications increases dramatically (Kraut and Streeter 1995).

2.2.2. *Multi Site*

Multi-site work often lasts longer than same-site work and requires more people to accomplish a job of equal size and complexity (Herbsleb *et al.* 2001). Kraut and Streeter found that communication barriers (geographic, organizational or social) reduced people's opportunities and eagerness to share information (Kraut and Streeter 1995). Allen (1977) has found that distance affects the frequency of communication.

Coordination of multi-site development can be organized according to standardized processes and written specifications (formal communication), but there is still a need for informal communication channels. Virtual coordination can be supported formally through legal formal contracts or informally through ad hoc relationships (Kraut *et al.* 1999).

Examples of successful multi-sited projects can be found. Open source projects such as Linux and Apache have developed software successfully in highly distributed loose communities. These projects utilize informal communication channels and governance structures and authority of project leaders for coordinating version releases (Markus *et al.* 2000).

In her research on the success factors in a highly distributed organization, Orlikowski (2002) found that there were a number of formal basic principles, a common process methodology, for managing coordination work. This methodology consisted of technical standards and coordination documents that were prepared at all levels of the project and that described the interdependencies between different parts of the system and how these parts were coordinated in the project in order to work together. Winograd and Flores (1986) observed further that coordination needs the participants' commitment to dependencies between activities. This commitment can be generated through 'speaking and listening' (Olson and Teasley 1996).

These communication barriers between teams can be alleviated by an architect who acts as a boundary spanner between teams (Curtis *et al.* 1988). The boundary spanner translates customer needs into terms understood by software developers.

2.3. Coordination Theories

Several coordination theories exist in different disciplines, which consider the coordination from different perspectives (Malone and Crowston 1994, Tolksdorf 2000). However, basically, there are only two kinds of models of coordination: 'traditional' and 'modern'. The 'traditional' model defines coordination as coordinating activities toward a common goal. This understanding is widely used to describe coordination in software development (Curtis *et al.* 1988, Grudin 1994b, Kraut and Streeter 1995, Grinter *et al.* 1999, Herbsleb and Grinter 1999, Herbsleb *et al.* 2001) and assumes a common goal shared by all entities. The 'modern' model sees coordination as the coordination of dependencies between activities toward a common goal. Both models are shown in Figure 1.

We will discuss the 'modern' model in more detail. Our aim is to compare our findings of the case of an international ICT company with this model. In order to do this, we have selected the coordination model developed by Malone and Crowston (1990, 1994). They defined coordination as *management of interdependencies between activities*. They assume that if there are no interdependencies, there is nothing

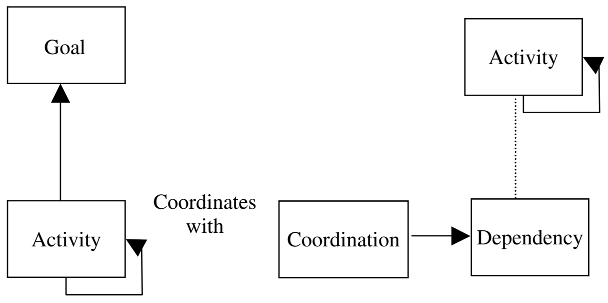


Figure 1. Two models of coordination: traditional (left) and modern model (right). Adapted from (Tolksdorf 2000)

to coordinate. The activities can be activities or objects; everything that has dependencies requires coordination (Malone and Crowston 1994). The model was selected because of its usefulness in characterizing development activities taking place in multiple sites. This kind of coordination does not require activities in different sites to be aware of each other all the time and to be able to cooperate to achieve the common goal. They should only have a common understanding of dependencies between activities and coordinate through them.

The model suggests that interdependencies between activities could be analyzed in terms of common objects that are involved in some way in both activities. These common objects constrain how each activity is performed. Different patterns of use of the common objects by the activities will result in different kinds of interdependencies. Malone and Crowston gave an example of a common object from designing and manufacturing a part, both involving the detailed design of the part: the design activity creates the design and the manufacturing activity uses it. The parts can be manufactured only after the design is complete and the actor doing the manufacturing has received a copy. This kind of pattern of usage can be called a *prerequisite constraint*. In a prerequisite constraint, one task creates an object, which is used by another object. This prerequisite constraint leads to a producer–consumer relationship between activities.

2.4. Coordination and Architecture

One of the first people to realize the relationship between coordination and software architecture was Melvin Conway (Conway 1968), who did so over 30 years ago. Since then, his findings have become known as *Conway's Law*, which suggests that software architecture mirrors the structure of the organization that designed the architecture.

Conway's structural view is the only widely accepted definition of software architecture (Shaw 2001). Structural view defines architecture as the structure of the components of a system, their interrelationships and principles and guidelines governing their design and evolution over time (Garlan and Perry 1995). The other definitions emphasize the configuration and style (Jacobsson *et al.* 1999), the constraints and semantics (Gacek *et al.* 1995) and the analysis and properties of the architecture as well as the different rationale (Perry and Wolf 1992), requirements and stakeholder needs (Kruchten 1995, Shaw 2001), Smolander's research study (2002) emphasizes 'architecture as a mutual reality'. In other words, the architecture is an important communication tool between different stakeholders on high-level structures and solutions of the system.

A few years after Conway's findings, Parnas (1972) recommended that the decomposition of software into modules should be done according to the division of labor rather than on the basis of flowcharts. He viewed a module as 'a responsibility assignment rather than a subprogram'. In large and complex systems, the developers, their division into groups and sites and their communication demands create the need for the coordination of the development effort. The software architecture influences the communication requirements between the project members. Geographically dispersed teams can develop software when the software architecture components are independent of each other (Olson and Teasley 1996).

This kind of software architecture requires communication and coordination for the successful development of the components. Thus, we believe that architecture is in fact a coordination mechanism in multi-site development. According to Malone and Crowston, the following seven types of dependencies are involved in coordinating activities. Basic coordination processes manage the relationships, and support processes serve to disseminate and collect information relevant to coordination processes (Malone and Crowston 1990). The four basic coordination processes are as follows:

Management of shared resources. Whenever multiple activities share some limited resources, they need a resource allocation process to manage the interdependencies between them.

Producer–consumer relationships. When one activity produces something that is used by another activity, they need to manage how to sequence activities, how to transfer the shared product, and how to ensure its usability by the activity that receives it.

Simultaneity constraints. Activities need to occur at the same time. This type of dependency requires activities to be synchronized.

Task–subtask relationships. When a group of activities are all subtasks for achieving some overall goal, they need to be integrated either through top-down goal decomposition or through bottom-up goal identification.

Support processes are orthogonal to these fundamental types of coordination processes. These processes serve dissemination and collection of information relevant to coordination processes. They process relevant information with the purpose of reaching consensus on specific coordination strategies (Malone and Crowston 1990). The three support processes are as follows:

Communication. Actors share the same language and 'common knowledge'.

Decision making. Actors propose alternatives, evaluate them and make choices (e.g. by authority, consensus or voting).

Perception of common objects. Actors see the same physical object in a shared situation or information in a shared database.

Most of the basic coordination processes require a decision to be made and accepted by a group. For example, the question of when actors should be assigned to activities requires a decision. Group decisions, in turn, require members of the group to communicate in some form the goals to be achieved, the alternatives being considered, the evaluations of these alternatives and the choices that are made. Finally, the establishment of communication language depends on the ability of actors to perceive common objects in the same way. These support processes can also be thought to coordinate the interdependencies between activities.

3. CASE ORGANIZATION

This study was performed in the software development department of an international ICT company. The whole department was divided into three different units at different geographical locations in Finland. Although the development happened in Finland, there were a couple of foreign developers participating in the development process. The coordination between the different sites had been planned to be carried out by using common processes and written specifications. This was supported by an extensive formal project handbook available in each subproject that described the communication and coordination to be used. In order to support informal communication across the sites, computer-mediated communication devices such as video conferencing, Internet Relay Chat (IRC) channels, shared calendars and electronic mail were in place. Table 1 shows some of the characteristics of the project.

The customer of this project was an internal one. The project was completed in 2001 and followed a traditional waterfall model with distinct requirements of collection, analysis, software design, implementation and testing phases. The aim

Table 1. The characteristics of the project

Characteristic	Project
Project execution start and end dates	Start date: 3.4.2001 End date: 5.12.2001
Software size (Line Of Codes, LOC)	138,000 LOC
Project cost	In total 3500 man days, design and implementation phase 2900 man days
Architecture	Service platform, distributed Server and centralized Client
Projecting	Divided into two subprojects, Server subproject and Client subproject
Work allocation	Server developed in the same site, Client developed in three different sites
The goal of the project	The renewal of old architecture
Maturity of used technology	New technology
Product life cycle phase	In the middle of its life cycle
Targeted markets	International

of the project was to develop a directory service platform for global telecommunications markets. The service had already been in the domestic market for several years. The global market posed such new requirements for service maintenance and delivery that the platform renewal was seen as necessary.

The project was divided into the following phases on the basis of the company's process model: the prestudy, feasibility study, project execution and piloting and maintenance. During the prestudy and feasibility study, requirements were collected, analyzed and the execution of the project was planned. In the execution phase, the software architecture was defined through the respective modules and their interfaces. This was followed by a detailed design, implementation, integration and system testing phases. After the system testing showed the quality of the software to be acceptable, the product went onto piloting and customer acceptance phases. The project was partitioned into two subprojects to facilitate easier management. Partitioning was carried out on the basis of the architecture and technology: one subsystem had a highly distributed, component-based architecture (Server) and the other was a centralized subsystem (Client), which handled authentication, authorization and the user interfaces. The functionality of the services required subsystems to communicate only through an easily extensible and configurable interface. The architecture of the case project, taken from the project architecture description, is shown in Figure 2.

The Server subsystem was responsible for dynamically resolving the information resources to be used by examining a service request and routing it to the right information resource. To do that, Server subsystem used CORBA (Common Object Request Broker) trading service. The main requirement for Server subsystem was high configurability: new information resources and services should be added by simply configuring it and adding the new modules to the system. Server subsystem was geographically distributed. The distribution was implemented using CORBA technology.

The Client subsystem was responsible for the user interfaces, authentication and authorization as well as for the interfaces to external systems. It converts the end-user's requests to a standard request for the Server subsystem, responds to the standard reply from Server subsystem and sends the answer back to the end user. The

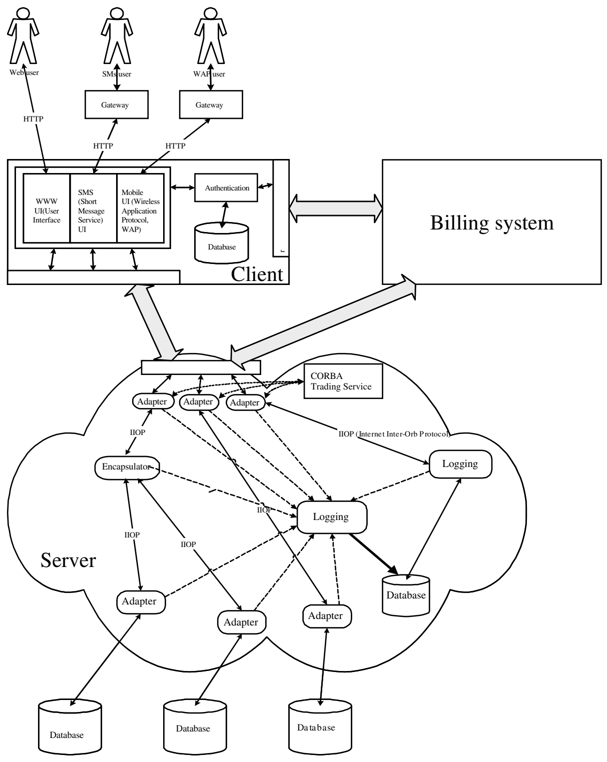


Figure 2. Architecture of the case-study system

Client subsystem was required to be independent of the location of the information maintained by the Server subsystem or how that information is retrieved from the information resources all over the world. This requirement was not well implemented; Client subsystem was dependent on the types of information resources residing in the Server subsystem. The platform was designed for the needs of different international markets and had to support interfaces for different kinds of external systems, e.g. SMS (Short Message) centers, WAP (Wireless Application Protocol) gateways, other platforms and billing systems. The needs of the international market posed additional demands on the user interfaces; e.g. they should be localizable to any language and should show the results in a country-specific manner. This also posed challenges on the Server subsystem: it had to be possible to dissipate the information all over the world and it had to support different kinds of information protocols.

From the user point of view, the service was quite simple. The user requested a service through the user interface that could be either mobile or

World Wide Web (WWW). The service fetched the requested information, processed it and returned the reply to the user.

4. RESEARCH METHOD

4.1. Research Subject and Method

We studied only architecture design, detailed design and implementation phases of the case project. The work allocation of the development activities was done according to the architecture. On the basis of this assumption, the research question of our study was formulated as follows:

Question 1. What kind of coordination problems related to software architecture was present during the system development?
Question 2. How did these problems differ in the same-site and multi-site environments?

The nature of our research problem led us to use a qualitative approach. Among the qualitative research methods, we used the case-study approach according to Yin (1994) and Eisenhardt (1989) for gaining insight into the phenomena and composed the study according to the theory building structure of grounded theory (Strauss and Corbin 1990). According to Strauss and Corbin, the grounded-theory method (Glaser and Strauss 1967, Strauss and Corbin 1990) is a 'qualitative research method that uses a systematic set of procedures to develop and inductively derive a theory about a phenomenon' (Glaser and Strauss 1967, Strauss and Corbin 1990). The notion of common object in Malone and Crowston's coordination theory was used as a priori construct (Eisenhardt 1989).

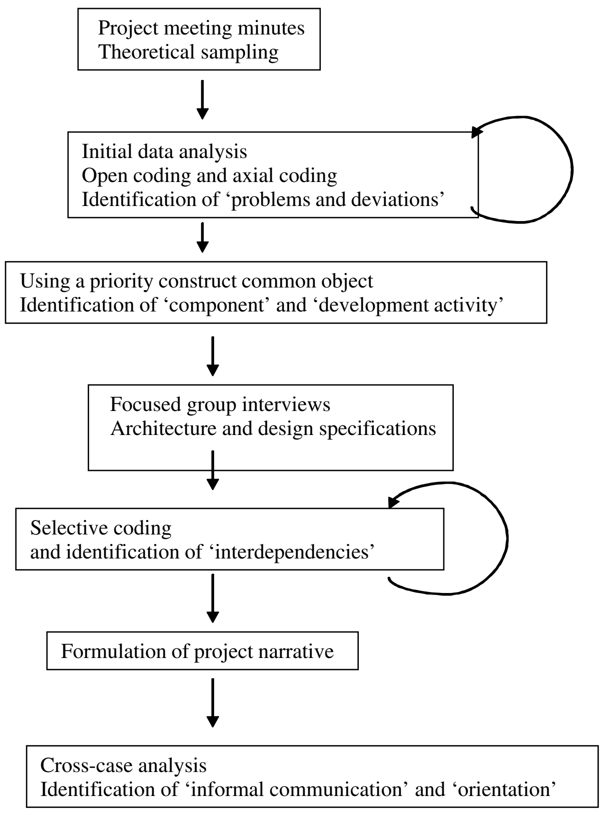
Our qualitative data analysis was performed in three phases following Strauss and Corbin's methodology of open coding, axial coding and selective coding (Strauss and Corbin 1990).

4.2. Research Process

The research process proceeded in four broad phases: data collection, data analysis, formulation of project narrative and cross-case analysis (Figure 3).

Data for the study was gathered from extensive documentation of the project (Table 2), using a theoretical sampling strategy. On the basis of the analysis of documentation, we decided to complement the written project material with focused group interviews among project participants. Glaser and Strauss (1967) call this *dynamic process of data collection* in which the sample is extended and focused according to emerging needs as theoretical sampling.

The open coding started with the identification of problems and deviations related to software architecture and coordination in the project progress.



Project meeting minutes
Theoretical sampling
Initial data analysis
Open coding and axial coding
Identification of 'problems and deviations'
Using a priority construct common object
Identification of 'component' and 'development activity'
Focused group interviews
Architecture and design specifications
Selective coding
and identification of 'interdependencies'
Formulation of project narrative
Cross-case analysis
Identification of 'informal communication' and 'orientation'

Figure 3. Research process

Table 2. Data available from the project

Data/Document
15 Progress report (from Project Manager)
Project managementSoftware: plan vs actual costs
11 Project steering group meeting minutes
46 Project group meeting minutes
Project plan
Functional specifications
Requirement catalog
Risk analysis document
Project quality criteria document
Architecture descriptions
Module specifications
Group interview material

These issues were brought to project meetings for discussion and decision making. The project meeting minutes and the group interview were the main sources for incident data. We used architecture and design specifications to help pinpoint the problems. We could observe in total 329 deviations and problems related to software architecture and coordination.

The concept of common object from Malone and Crowston's coordination theory was used as a starting point to interpret the identified issues in data. First, we divided the development process into software design and implementation phases according to the project management guidelines of the company. We observed that the common object in designing and implementing the software was a component. Software architecture described the structure of the components and the principles and guidelines for their design and implementation (Garlan and Perry 1995). It also described constraints concerning the pattern of use of the components, resulting in possible interdependencies between components.

We also assumed that the main support activity of software development was the management of the development process. It can be divided into two parts in the same way as software development: software process planning and execution management. The common object of these is a development activity. In the next table (Table 3), the processes involved in software development and their common objects are summarized.

We used these two common objects, namely, component and development activity, to find the interdependencies between activities that caused problems in the project. In the analysis of the materials, we identified three interdependencies between components (interfaces, assembly order and interdependence) and three interdependencies between development activities (responsibility, communication and orientation) that could largely explain coordination problems in the project. How these interdependencies appeared in the project is explained in the project narrative (Section 4.3) and the resulting coordination processes in Section 5.1.

Table 3. Processes in software development

Process	Common object
Software design and implementation	Component
Software process planning and execution management	Development activity

After finding the interdependencies, we tried to find the answer to the research question of how the problems in coordination differed in multi-site and same-site environment. Eisenhardt calls this *cross-case analysis* (Eisenhardt 1989). To find these differences, we compared the two subprojects, multi-site Client subproject and same-site Server subproject, and analyzed the differences in the coordination processes between them. The results of this comparison are explained in Section 5.2.

4.3. Project Narrative

The following project narrative traces coordination problems in our case project. Coordination problems are traced both in Client and Server subprojects through four episodes.

4.3.1. Episode 1. The Beginning: Assumptions, Atmosphere and Communication

At the beginning of the project, software architecture was regarded as an important method in the coordination of distributed software development. The allocation of the development work was performed according to the software architecture. The software development processes in the company were also thought to guarantee coordination. During the project, it became apparent that the actual allocation of work did not follow the plans and that the processes did not guide the work at all.

The project participants and the customer representatives were chosen for the project on a technical basis. The project's goal (new architecture of the old system) also emphasized technical matters. In the Client subproject, the social and organizational communication barriers between the foreign consultant architect and the software developers were high. The consultant architect was appreciated as a business-oriented and capable architect, but the language and cultural differences made the conversation difficult. He did not understand his role as a communicator of architecture, and he left many of the design decisions to be resolved by the designers. He concentrated on conversation with the customer about the Client subsystem features and functionalities. They had nine official project meetings during the project. They used e-mail and the project's IRC channel for communication, but not enough to get the common understanding of the architecture of the Client subsystem. The problems of teamwork, communication and coordination were discussed many times in the Client subproject meetings.

In the Server subproject, the architect was appreciated as a technically capable and experienced person. He was not a communicator either, but the designers resided in the same site and they could discuss with the architect and other developers every day. There were also problems with communication within the Server subproject, but they concerned the formal communication. They did not have any official project meetings, only those that concerned the whole project. The project participants did not raise issues in the project meetings and project managers had problems when they tried to find out the situation of the subproject. Still, the designers knew what to do because they used informal communication channels actively, especially face-to-face conversations.

4.3.2. *Episode 2. Architecture Design: Orientation and Responsibilities*

The orientation of the project architects affected their interest in the project activities. The architect of the Server subproject concentrated on the coding and technical matters of Server subsystem, and the architect of the Client subsystem concentrated on the functionalities and features of the Client subsystem. In the architecture design phase, they did not see that the two subsystems would be so interdependent on the project. They considered the interface between the subsystems to be a technical 'request–reply tube'.

The interface between the subsystems was defined at the technical level, using terms such as 'XML' (Extensible Markup Language) and 'TCP/IP'. The information that passed between these subsystems in XML documents seemed to be unimportant. The promises of the new XML technique were so fascinating that the main issue, the information that passed within the XML documents, was forgotten.

In the Client subproject, the architecture design document was more an analysis of the different alternatives for doing business with the Client subsystem. In the Server subproject, this document contained technical viewpoints of the Server architecture. Although the document was limited, this was still considered adequate by the Server subproject participants. Furthermore, there was no document describing the overall architecture in either project. This was despite the need to communicate the complex architectural structures and solutions between the teams. The architect of the Server subproject had an intrinsic responsibility for the whole architecture. However, there was no official decision about it and, in practice, nobody assumed this responsibility.

4.3.3. *Episode 3. Different Interpretations of the System*

Problems described in the previous Episodes heavily affected the project stakeholders' interpretation of what kind of system they were developing in the project. They did not have a common understanding of the architecture of the system. Every designer saw the architecture of the system and its component dependencies differently. This phenomenon can be seen in Figure 4.

The figure shows component design specifications of four designers and how they visualize their component's communication with the other components of the system. We can observe that each designer has put his/her own component in the middle of the subsystems, thus emphasizing the importance of his/her task. Each component designer has in this way his/her own individual understanding of the architecture. Thus, in practice, there exist as many interpretations of the architecture as there are component developers in the system. In other words, this has led to eight different interpretations of the system architecture.

4.3.4. *Episode 4. Emerging Coordination Problems*

In the later phases of development, the lack of common understanding of the architecture caused

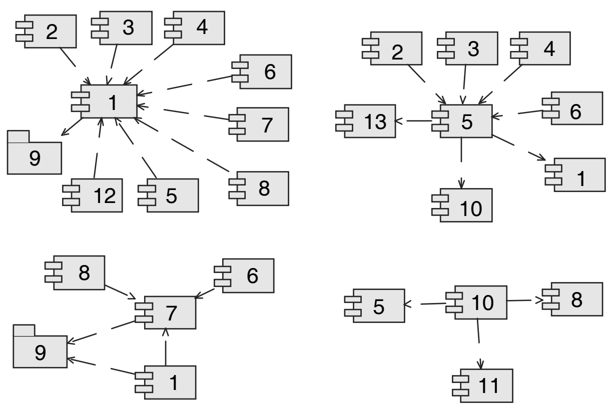


Figure 4. Representations of the system based on designers' interpretations

a lot of coordination problems in the Client subproject. These problems made the realization of the actual system more difficult.

The Client subsystem was tightly coupled with the Server subsystem, which complicated the development of the Client subsystem. A change implemented in the information structure of the Server side could have a major influence on the functionality of the Client. The problems became more serious when the Client subproject initiated the implementation phase one month before Server subproject.

The interface specification between the Client and Server was sketched parallel to the other design in the Server subproject, which meant that the interface specification was way too late for the Client project. The design of the Server subsystem should have been ready before the Client and not vice versa. This brought about serious difficulties for the Client module designers and especially for those developing module(s) near the subsystem interface. Their module functionality depended on the structure of information in the interface and the designers were not aware of that. The module designers communicated the situation to the architects, but this was done too late. The specification of the interface was delayed, and a lot of work had already been wasted.

The system integration phase, in which the system was composed from its components, was time consuming in the Client subproject. The integration caused a lot of problems, and many changes to the interfaces and components had to be made before the integration of the components into a working system. These problems surfaced in the testing phase, in which the integration problems made it impossible to run the tests.

5. RESEARCH FINDINGS

5.1. Coordination Processes

During the analysis, we discovered six different coordination processes to explain most of the coordination problems in our case study. These categories were as follows:

Managing interface between system components, i.e. how the functionalities of one component affect the functionality of the other components.

Managing the assembly order of system components, i.e. how the components can be integrated to each other into a working system.

Managing interdependence of the system components, i.e. how the system components are dependent on each other.

Communication, i.e. how the communication between development participants is taking place.

Overall responsibility, i.e. how the responsibility of decisions related to the software architecture is taking place.

Orientation, i.e. how the orientation of development participants affects the interpretation of the architecture in a shared situation.

Each type of coordination process corresponds with a dependency between components or activities. Table 4 shows the dependency types along with an explanation of dependency in the case-study context. The explanations of the dependencies are supported with examples taken from the project material.

5.2. Same-site Versus Multi-site Development

The results of the comparison between same-site and multi-site development problems in our case study are shown in Table 5. The project episode number is noted in italics and indicates where the description of each dependency is described in the project narrative in Section 4.3.

As we can see in the table, the main difference between the two subprojects seemed to be in the informal communication and orientation. Client subproject had problems in informal communication mainly because the subproject architect and designers had cultural differences and they could not communicate with each other. The physical distance between subproject participants resulted in poor informal communication. The situation in the Server subproject was better: they were in the same site, and in this way, informal communication was easier. The architecture description was poor in both subprojects, but the Server subproject participants did not see this as a major problem because they could talk with the architect every day. The difference between architects' orientation in these two subprojects affected, mostly, the content of the architecture description. In the Client subproject, the architect's orientation was business oriented containing analysis of different business cases and in the Server subproject, it was a technical one containing technical terms. Both of the architecture descriptions were partial, one from a purely

Table 4. Dependencies between components and development activities: explanations and examples

Dependency	Explanation	Examples from the project material
Interface between system components	Components of the system share the common functionality, the functionality of one component affects the functionality of other component	*'Interfaces were designed too late in the Client subproject, the implementation of components were already going on'*
Assembly order of system components	System is composed of components. The integration of system components into a working system has some predefined and planned order.	*'According to Build Plan Client 4th build was postponed due to Client-Server integration problems... '* *'Start of testing is late because of integration problems'*
Interdependence of system components	One component produces something that is consumed by another component	*'The functionality of the Client subsystem was dependent on the resources of the Server subsystem'* *'Client-Server interface was dependent on Server resources'* *'The Client subproject initiated implementation one month before the Server subproject'*
Communication	Communication disseminates information between activities	*'In the Client subproject, there is a big need to talk with architect and designers'* *'The language and cultural barriers were huge between architect and designers in the Client subproject'* *'I asked several times the Client subsystem architect help to design my component but I did not understand what he answered to me'*
Overall responsibility	The responsibilities of the architectural decisions. Hierarchy/decision-making approach.	*'No-one in the project took responsibility for the whole architecture and the interface between the two subsystems. The project needed this kind of a 'chief architect.'* *'The Server architect should concentrate more on the architecture and not module coding'*
Orientation	Partial orientation in activities that affect how the common language was shared between activities and how the architecture was interpreted in shared situations	*'Service should be easy to configure and manage with its components'* *'Data transfer between components and configuration should be in XML-format'* *'Client architect concentrated too much on customer requirements, not architecture design'*

Table 5. Differences between two subprojects

Dependency	Client in a multi-site environment	Server in a same-site environment
Interface between system components	Problems exist *(episode 4)*	No problems exist *(episode 4)*
Assembly order of system components	Problems exist *(episode 4)*	No problems exist *(episode 4)*
Interdependence of system components	Problems exist *(episode 4)*	No problems exist *(episode 4)*
Overall responsibility	Problems exist *(episode 2)*	Problems exist *(episode 2)*
Communication	Problems exist in informal communication *(episode 1)*	No problems exist in informal communication *(episode 1)*
	Problems exist in formal communication *(episode 1)*	Problems exist in formal communication *(episode 1)*
Orientation	Business oriented *(episode 2)*	Technically oriented *(episode 2)*

business point of view and the other from a purely technical point of view.

In the multi-site Client subproject, there were several coordination problems in the implementation of the final system. Interfaces between system components were designed too late and the interface design between Client and Server subsystem had the wrong timing. Both of these caused a lot of changes and wasted work in the Client subsystem. This also affected the compatibility of the system components, which made the integration of components and testing difficult and delayed the finalization of the project. In the Server subproject, these problems did not surface.

6. DISCUSSION AND CONCLUSIONS

This paper described the role of software architecture in the coordination of multi-site software development through a case study. Our objective was to better understand the coordination in multi-site environment, characterized by geographical, cultural and language distances between development participants. We used a case-study approach to study two subprojects in an international ICT company.

Architecture was intended to be the tool for coordination in the project. The architecture description was supposed to contain the rules of how the components of the system exchange information with each other. The observed problems in coordination (lack of overall responsibility, communication and orientation) in architecture design resulted in poor or missing architecture description and different interpretations of the architecture by the project members. These problems caused actual system coordination problems in later phases. There were problems with managing the interfaces between system components, their assembly order and interdependencies between them. These problems manifested themselves in the difficulties of the realization of the final system as well as delays in the timetable.

Our study suggests that in the multi-site environment, it is not enough to coordinate activities, but in order to achieve a common goal, it is important to coordinate interdependencies between the activities. This kind of coordination needs a common understanding of the software architecture between software development participants. According to our understanding, participants coordinate their development work through interfaces of their components. Each component can be developed separately, and thus, it is not necessary to take into account development of other components or distance, cultural and language differences between sites. The important issues that matter in this case are the appropriate architecture description and well-defined interfaces between components. Furthermore, these need to be communicated, both informally and through formal descriptions, to all parties involved.

6.1. Implications for Research

The common understanding of coordinating software development activities means, from the multi-site perspective, that activities residing in different sites should have awareness of each other all the time to coordinate their work. Quite often this is not possible, although different kinds of electronic media are seen as remedies to this problem. Also, this approach assumes that all activities are willing and are able to cooperate to achieve a common goal. In multi-site environment with different cultures and languages, this is not a realistic assumption.

We suggest that in multi-site software development, it is not enough to coordinate activities, but in order to achieve a common goal, it is important to coordinate interdependencies between the activities. The interdependencies between components are described by software architecture. When the coordination is done by using architecture, the work allocation is made according to this component structure. All the component designers have to have a common understanding of component interdependencies and they have to have well-defined interfaces to be able to develop their own components. The component developer does not have to be aware of other component development as long as the interfaces remain the same. In the case of interface changes, the component developers should coordinate, mostly, the interface change, not so much the whole component. This principle is widely known as *encapsulation and information hiding* (Parnas 1972).

Correspondences between our findings and Malone and Crowston's model are summarized in

Table 6. Correspondence between coordination processes in this study and Malone and Crowston's study (Malone and Crowston 1994)

Our coordination processes	Malone and Crowston's coordination process
Managing the interface between system components	Managing shared resources
Managing assembly order of system components	Managing task–subtask relationships
Managing interdependence of system components	Managing producer–consumer relationships
Communication	Communication
Overall responsibility	Decision making
Orientation	Perception of common objects

Table 6. The main difference is the importance of software architecture as a coordination tool in multi-site software development. Software architecture was meant to serve as a vehicle for communicating the dependencies between components among the project participants and development activities in our case project.

We extend Malone and Crowston's findings and emphasize that software architecture could serve as a coordination tool, especially in multi-site development, but the usefulness of it depends heavily on the key people. The software architecture in our study served as a commitment into dependencies between the activities. However, we observed that without this commitment in a multi-site environment, coordination did not succeed and caused problems for the project. As discussed in Section 2.2, such a commitment is not so much the result of 'speaking and listening' but, rather, the result of authorized decision making by a chief architect to maintain the conceptual integrity of the system (Brooks 1995).

Our suggestions for the role of the software architecture in multi-site software development extend Smolander's research study (Smolander 2002) about the role of software architecture as a communication tool. The coordination of multi-site work needs a common understanding of the architecture of the system to direct the development work toward a coherent, working system.

6.2. Implications for Practice

To improve the use of architecture in the coordination of practical multi-site development work, organizations should improve both their informal communication capabilities and software development methodologies. One way to improve informal communication in multi-site environment is the effective use of electronic communication channels and CSCW (Computer-supported Cooperative Work) technology (Lee and Malone 1990, Grudin 1994a, 1994b). A software development methodology that supports multi-site development instead of concentrating only on the individual work practices should include at least the following requirements derived from the observations and explanations in our case study.

First, we emphasize the use of multiple viewpoints in architecture design for various stakeholders. Boland, Tenkasi and Te'eni (1994) observed in their study that representations of knowledge need to be distinctive for different individuals and that they need to include various levels of detail. The architecture can be regarded as a representation of knowledge of the system structure.

Different representations of architecture would help organizations get better common understanding of the architecture to satisfy all stakeholders. These viewpoints are dependent on the stakeholders needs. The use of viewpoints helps in finding interdependencies of architecture components, and it is important to choose such viewpoints that reveal these interdependencies. The organization and work division should be made according to software architecture, considering the interdependencies of the system parts. The components that depend heavily on each other should be developed in the same site or sites residing near each other.

Second, the architecture design work in a multi-site development environment requires a 'chief architect' to communicate the structures and solutions of the system to get the common understanding of the architecture of the system and to help in coordination of development work. In Section 2.2, we discussed that architects are boundary spanners through which other groups can sufficiently coordinate the design work in order to accomplish the architectural design. As Brooks (1995) stated in the 'The Mythical Man-Month': 'the architect is like a director and the manager, like a producer of the motion picture'. He also emphasized the chief architect's important role in maintaining the conceptual integrity of the system developed (Brooks 1995). We emphasize that the architect's role as a communicator and coordinator is even more important in multi-site development, in which communication and coordination is more difficult because of long distances.

Third, the interface design should be done early enough in the architecture design phase to guide the designers in their component design work. In this way, the designers can concentrate on their component design independently. A special coordination plan, in which the coordination between different sites is described, is also needed.

6.3. Topics for Further Study

In the future, we will seek, and possibly develop, a software development method that supports the multi-site coordination aspects of organizations that deal with distributed software development on the basis of the requirements derived from our study. This requires a more comprehensive sample of projects. Further, the method should concentrate on multi-site working practices with a set of people collaborating with each other instead of concentrating only on individual work practices. We seek also to combine the research on distributed teamwork with the work on architecture definition languages. This could give us new tools for managing complex system development across geographical, cultural and technical borders.

REFERENCES

Allen T. 1977. *Managing the Flow of Technology*. MIT Press: Cambridge, MA.

Boland RJ, Tenkasi RV, Te'eni D. 1994. Designing information technology to support distributed cognition. *Organization Science* **5**(3): 456–475.

Brooks FPJ. 1995. *The Mythical Man-Month*: *Essays on Software Engineering*, Second Edition, Addison-Wesley: Boston, Massachusetts, USA.

Carmel E. 1999. *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall: New Jersey, USA.

Carmel E, Agarwal R. 2001. Tactical approaches for alleviating distance in global software development. *IEEE Software* **18**(2): 22–29.

Castells M. 1996. *Rise of the Networked Society*. Blackwell Publishers: Cambridge, MA.

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

Curtis B, Krasner H, Iscoe N. 1988. A field study of the software design process for large systems. *Communications of the ACM* **31**(11): 1268–1287.

Eisenhardt KM. 1989. Building Theories from Case Study Research. *Academy of Management Review* **14**(4): 532–550.

Finholt TA, Rocco E, Bree D, Jain N, Herbsleb JD. 1998. NotMeeting: A field trial of NetMeeting in a geographically distributed organization. *SIGGROUP Bulletin* **20**(1): 66–69.

Gacek C, Abd-Allah A, Clark B, Boehm B. 1995. On the definition of software system architecture. ICSE 17 Software Architecture Workshop, Seattle, Washington, April 24-25, 1995; IEEE Press.

Garlan D, Perry D. 1995. Introduction to the special issue on software architecture. *IEEE Transaction on Software Engineering* **21**(4): 269–274.

Glaser B, Strauss AL. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Adline: Chicago, IL.

Grinter RE. 1999. Systems architecture: product designing and social engineering. Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration (WACC '99), San Francisco, CA.

Grinter RE, Herbsleb JD, Perry DE. 1999. The geography of coordination: dealing with distance in R&D work. Proceedings of the International Conference on Supporting Group Work (GROUP '99),: Phoenix, AZ, November 14-17, 1999 ACM Press.

Grudin J. 1994a. Computer-supported cooperative work: history and focus. *IEEE Computer* **27***(5): 19–27*.

Grudin J. 1994b. Groupware and social dynamics: eight challenges for developers. *Communications of the ACM* **37**(1): 93–105.

Herbsleb JD, Grinter RE. 1999. Architectures, coordination, and distance: conway's law and beyond. *IEEE Software* **16**(5): 63–70.

Herbsleb JD, Mockus A, Finholt TA, Grinter RE. 2000. Distance, dependencies and delay in global collaboration. Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '00), Philadelphia, PA, December 2-6, 2000 ACM Press.

Herbsleb JD, Mockus A, Finholt TA, Grinter RE. 2001. An empirical study of global software development: distance and speed. Proceedings of the 23th International Conference on Software Engineering (ICSE '01), Toronto, Canada, May 15-18, 2001 IEEE Press.

Jacobsson I, Booch G, Rumbaugh J. 1999. *The Unified Software Development Process*. Addison-Wesley Longman.

Kraut RE, Steinfield C, Chan AP, Butler B, Hoag A. 1999. Coordination and virtualization: the role of electronic networks and personal relationships. *Organization Science* **10**(6): 722–740.

Kraut RE, Streeter LA. 1995. Coordination in software development. *Communications of the ACM* **38**(3): 69–81.

Kruchten PB. 1995. The 4 + 1 view model of architecture. *IEEE Software* **12**: 42–50.

Kyng M. 1991. Designing for cooperation: cooperating in design. *Communications of the ACM* **34**(12): 64–73.

Lee J, Malone TW. 1990. Partially shared views: a scheme for communicating among groups that use different type hierarchies. *ACM Transactions on Information Systems* **8**(1): 1–26.

Malone TW, Crowston K. 1990 What is coordination theory and how can it help design cooperative work systems? Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '90), Los Angeles, CA, October 7-10, 1990 ACM Press.

Malone TW, Crowston K. 1994. The interdisciplinary study of coordination. *ACM Computing Surveys* **26**(1): 87–110.

Markus ML, Manville B, Agres CE. 2000. What makes a virtual organization work? *Sloan Management Review* **42**(1): 13–26.

Olson JS, Teasley S. 1996. Groupware in the wild: lessons learned from a year of virtual collocation. Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '96), Boston, MA, November 16-20, 1996 ACM Press.

Orlikowski WJ. 2002. Knowing in practice: enacting a collective capability in distributed organizing. *Organization Science* **13**(3): 249–273.

Parnas DL. 1972. On the criteria to be used in decomposing systems into modules. *Communications of the ACM* **15**(12): 1053–1058.

Perry D, Wolf A. 1992. Foundations for the study of software architecture. *ACM Sigsoft Software Engineering Notes* **17**: 40.

Shaw M. 2001. The coming-of-age of software architecture research. Proceedings of the 23rd International Conference on Software Engineering (ICSE '01), Toronto, Canada, May 15-18, 2001 IEEE Press.

Smolander K. 2002. Four metaphors of architecture in software organizations: finding out the meaning of architecture in practice. International Symposium on Empirical Software Engineering (ISESE 2002), Nara, Japan, October 03-04, 2002.

Strauss A, Corbin J. 1990. *Basics of Qualitative Research: Grounded Theory Procedures and Applications*. Sage Publications: Thousand Oaks, CA.

Szyperski C. 1998. *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, Boston, MA, USA.

Tolksdorf R. 2000. Models of coordination. Proceedings of the Engineering Societies in the Agent World (ESAW), Berlin, Germany, August 21, 2000 Springer-Verlag.

Winograd T, Flores F. 1986. *Understanding Computers and Cognition*. Ablex: Norwood, New Jersey.

Yin RK. 1994. *Case Study Research: Design and Methods*, 2nd edn. Sage Publications: Thousand Oaks, CA.