Coordination Implications of Software Architecture in a Global Software Development Project

Alberto Avritzer, Daniel Paulish
Siemens Corporate Research
755 College Road East
Princeton, NJ 08540
{alberto.avritzer, daniel.paulish}@siemens.com

Yuanfang Cai
Dept. of Computer Science
Drexel University
Philadelphia, PA, 19104 USA
yfcai@cs.drexel.edu

Abstract

In this paper, we report on our experience assessing the relationship between the dependency structure of a software architecture and the coordination needs among distributed development teams. We use as a case study for global software development the Global Studio Project Version 3.0, where matrix models were used to represent both architectural dependencies and the coordination structure among the team members. Analysis of data gathered during the Global Studio Project Version 3.0 revealed that design structure matrix (DSM) models representing the modular structure of the software architecture are highly consistent with the social network matrix models that represent the actual coordination structure. We conclude that DSM modeling can help guide the task assignments in global software development projects.

1 Introduction

For the past few years, Siemens has been experimenting with software development processes and practices for globally distributed projects using student-based development teams from universities around the world. The students who make up this development project simulate an industrial software development project using common practices for collaboration among distributed sites. We refer to this experimental global software development project as the Global Studio Project (GSP). Experiences with this project have been reported in a number of papers, and it has been documented as a case study (GSP 2005) within [9]. This paper reports on the GSP experience during the third year of the project, referred to herein as GSP V3.0. We report on our experience assessing the relationships between system architecture, viewed as a set of interdependent decisions, and the social interactions for an experimental software development project. Our approach is to uniformly represent both architectural design decisions and the actual communications among global teams using matrix models, and compare the consistency between them.

For many years, software project organizations have often been structured in accordance with the system architecture design of the product being developed [8]. For globally distributed software development (GSD) projects, coordination and communication among the development teams are more complex than for collocated projects due to time, distance, and cultural differences. For globally distributed software development projects, it is especially important to minimize the need for communication between teams that are not collocated and to maximize the communication within a local team.

Viewing software architecture design as a decision-making process, we hypothesize that the dependency structure among design decisions implies the potential needs for communication. Therefore, it is important to make the potential communication requirements between modules explicit early in the architecture development process such that tasks could be assigned to global teams to maximize the project communication efficiency. To achieve this purpose, we need a suitable representation of the modular structure of the design to reveal the potential needs for communication among architects and developers. Cataldo et al. [5] empirically studied the possibility of identifying the coordination requirements from modification requests.

In this paper we use *design structure matrix* (DSM) [10, 7, 2] modeling to represent the architectural structure as interdependent design decisions. We use DSM modeling because of its ability to explicitly represent design decisions, express *design rules* [2] and reveal interdependencies between architecture modules. Given the GSP V3.0 architectural design, we first represent the components as design decisions, model their relations as logical constraints, and then identify the dominance relation among these decisions, which constitute an *augmented constraint network*

(ACN) [3]. From the GSP ACN, a GSP DSM can be automatically derived [4], which manifests the decoupling effects of design rules and the independent modules within the architecture.

We also use a matrix model to represent the actual communication structure among project teams. Since the GSP V3.0 was managed by software engineering researchers, the project was highly instrumented and all team members were periodically surveyed concerning the practices used and their experiences on the project. One such survey reported in this paper is the Social Network Analysis (SNA), from which we were able to get insights into the communications patterns among project team members during the course of the development. We also derive matrix models from the SNA.

Our analysis showed that the modular structure of the architecture is highly consistent with the communication structure. We observed that during the early architecture design phase the lead architect for each major system component is a focal point for project communications. In addition, after the architectural design rules became more stable, there was less coordination needed among the remote teams.

The conclusions from this experience report are that DSM modeling has the potential to help guide task assignments and team coordination; and that the coordination process of GSP V3.0 is efficient because the lead architects, that is, the design rule decision-makers, are distributed to component teams, each responsible for an architectural module.

The remainder of the paper is organized as follows. In Section 2, we describe the GSP V3.0 and present a brief overview of the tool that was developed during the execution of GSP V3.0. In Section 3, we present the data collection approach. In Section 4, we introduce the GSP architectural DSM and communication matrices. In Section 5, we discuss our results in terms of project architecture, process and management. In Section 6, we present our conclusions and topics for future research.

2 Global Studio Project V3.0

The "extended workbench model" development process used during the first two years of the GSP had the following characteristics [1].

- Centralized Project Management and Control: The software process was managed centrally at a headquarters location. System requirements, architecture design, and system testing were performed at the central site.
- Iterative Development: Continuous integration tools and methods were used such that the product features were added as they were developed.

Member/Arch Role /Team Role	Site	Component
AA/arch/lead	SCR/US	overall project
AB/arch	SCR/US	Performance Engineering
BH/arch	SCR/US	UML Modeler
FD/arch/lead	SCR/US	Tangram
VC/arch/lead	L'Aquila/Italy	Performance Model Generator
GF	Chemtech-Rio/Brazil	Tangram web service
RC//lead	Chemtech-Rio/Brazil	Tangram web service
RL//lead	COPPE/UFRJ-Rio/Bazil	Tangram
ES	COPPE/UFRJ-Rio/Bazil	Tangram
PF/arch/lead	PUC-RS/Brazil	project
EN//lead	PUC-RS/Brazil	PEPS
MC	PUC-RS/Brazil	PEPS
EM	PUC-RS/Brazil	PEPS
JM	Zaragoz/Spain	Performance Engineering
RU/arch/lead	PUC-RS/Brazil	Testing
LM	PUC-RS/Brazil	Testing

Table 1. The Global Team of the GSP V3.0 Project

- Minimization of Cross-team Communications: The central team coordinated communications between the remote development teams for efficient communications. Most of the project communications were between the remote teams and the central team, in a hub and spoke pattern.
- Formality of Requirement Specifications: The central team documented and clarified product requirements for the remote teams. The central team had project roles of chief architect, requirements engineer, integrator, quality assurance, and supplier management.

Although the extended workbench model worked well for the GSP in that project goals were successfully met, the communications burden on the central team was significant, and it's difficult to envision how the process would scale for very large projects without additional delegation of responsibilities from the central site to the remote sites. With a single central site and a limited number of remote sites, our experience has shown that the extended workbench model will work effectively for medium-sized software systems with a maximum total size of 15 MLOCs.

The extended workbench model assumes a well-defined loosely coupled software architecture where components are allocated to individual teams and the components interact with each other through well-defined interfaces. When two teams are assigned components between which there is an interface, the central team will coordinate and encourage communications between the two teams. Such foresight

into planned communications patterns requires early phase design and analysis efforts such that the requirements and software architecture are understood before initiating component development.

For the third year of the Global Studio Project (GSP V3.0), a new product, process, and organizational structure were used as compared to the first two years. For GSP version 3.0, a "system of systems" process was used with the following characteristics:

- Hybrid Centralized/Distributed Management: The software development process is still developed and managed centrally, but the architecture and requirements engineering teams are extended with key domain experts resident at the remote sites. The objective is to use the specialized domain knowledge about the large existing software systems to help steer the overall requirements and software architecture specification efforts.
- Software Integration Testing: This work is done by a specified remote team, rather than the central team. Continuous integration is also used.
- Frequent Communications between Teams: Communication between the development teams and the remote integration testing team is encouraged. The central team is not required to coordinate the communications among teams.
- Formal Testing Specifications: Testing specifications are formal and will be provided by the remote integration testing team to the development teams.
- Less Formal Requirement Specifications: The upfront formal specification of the common interface between components and the availability of domain experts on the existing components reduces the need for formal requirement specifications.

The GSP version 3.0 project team is organized as a central coordinating team, a distributed architecture/requirements team, several remote development teams, and one remote integration testing team. The central coordinating team is responsible for product identification and assignment of components to distributed development. As the components being integrated are existing operational systems, the test cases/use cases can be developed by inspecting the existing system's user interface.

2.1 State-of-the-Practice

Today's software project managers have a large number of possible ways to structure their GSD projects across multiple development sites. If a software architecture design exists at the time when the development work is being allocated among development sites, it will often be a driver for the project's organizational structure. Some of the project organizational approaches that could be considered include product structure, process steps, release, competence center and open source.

In a product structure organizational approach, the architecture decomposes the system into components and the components are allocated as work packages to the different sites. In a process steps structure, work is allocated across the sites in accordance with the phases of the software development process; e.g., design may be done at one site, development at another site, and testing at yet another site. In a release based organization approach, the first product release is developed at one site, the second at another site, etc. Often, the releases will be overlapped to meet time-to-market goals; e.g., one site is testing the next release, another site is developing a later release, and yet another site is defining or designing an even later release. In a platform structure, one site may be developing reusable core assets of the product line and other sites may be developing application-level software that uses the platform. In a competence center organizational approach, work is allocated to sites depending on the technical or domain expertise located at a site. For example, perhaps all user interface design is done at a site where usability engineering experts are located. Lastly, in an open source structure, many independent contributors develop the software product in accordance with a technical integration strategy. Centralized control is minimal except when an independent contributor integrates his code into the product line.

These organizational approaches may change over time. For example, components may be allocated at first with the intent that the remote site will develop the skills over time to become a competence center in the functionality that component provides. In addition to the organizational structure, a global software development process must be selected or created which supports the structure. During the first two years of the GSP, a product structure approach was used to structure the organization and an extended workbench model development process was used [9]. This resulted in a hub and spoke organizational structure where the remote component development teams communicated mostly with the central team roles (e.g., chief architect, project manager, supplier manager) at the headquarters or central site.

2.2 Case Study, UML-PM

For GSP V3.0 we selected as a case study the development of a tool to generate performance models from UML model specifications (UML-PM). The approach used was to identify remote sites as competence centers, and motivate domain experts located in these sites to collaborate by generalizing existing tools that were designed for specific purposes. We list below the identified tools and competence centers:

- UML Editor - Siemens Corporate Research (SCR), Princeton was identified as the center of competence for UML model specifications and for UML editor

development. The TDE (Test Development Environment) tool developed at SCR for the last several years served as the front end for our UML-PM tool.

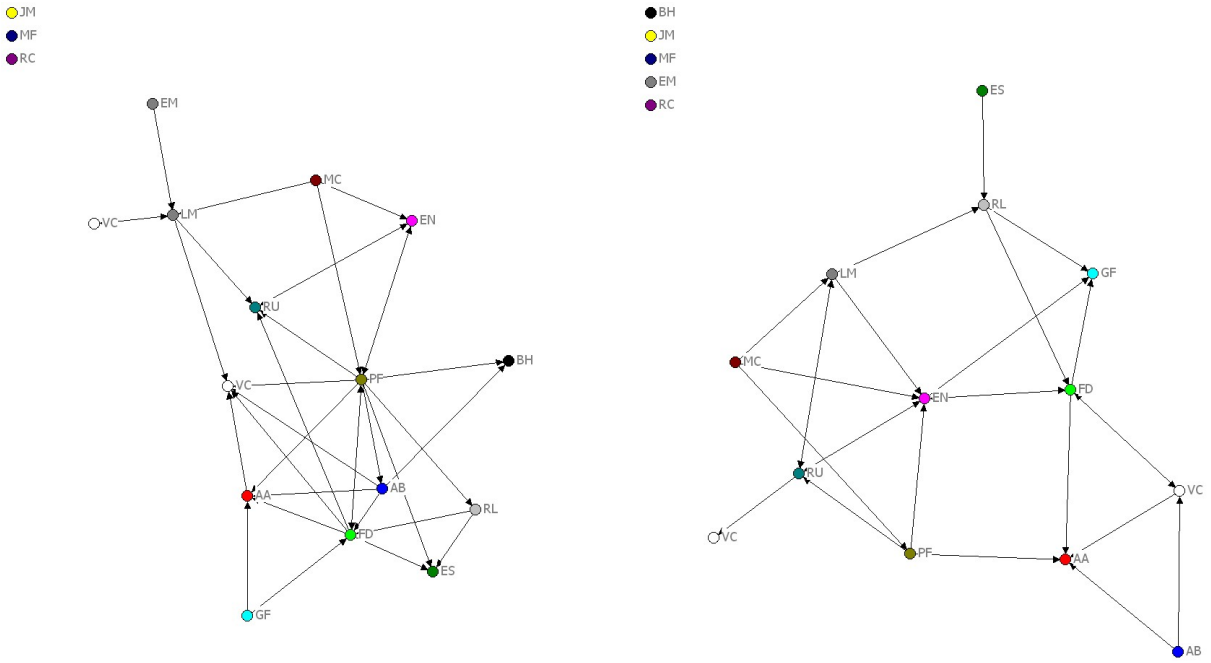
- Performance Model Generator - L'Aquila University in Italy, was identified as the center of competence for model transformations from UML specifications to queueing network representations. The Mosquito tool developed at L'Aquila University [6] over the last several years served as the main focus of architecture activities for the UML-PM tool. The Mosquito tool was wrapped to act as a web-service and adapters were written to upstream tools, i.e. towards the customer, and downstream tools, i.e. towards the performance model solver, for proper integration with the Mosquito tool.
- Performance Modeling Solvers - COPPE/UFRJ and PUC-RS Universities, Brazil were identified as centers of competence for performance modeling solvers. Adapters to the Tangram-II and PEPS tools were written to integrate the output of the Mosquito performance modeling representation with Tangram-II and PEPS input representation of performance models.

Therefore, for GSP V3.0, more of an open source approach was used, with competence centers located at the remote sites, since each university had expertise on the specific performance analysis tool that they had developed which was to be integrated into the performance analysis tool set. The tools had already been individually developed and had to be integrated using a "system of systems" type of process where a central team was used for integration but had much less control than with the extended workbench model. The system of systems approach seems to have worked successfully for GSP V3.0 since the collaborating universities were motivated and successfully integrated their systems into a powerful tool set. The approach depended much on mutual trust among the development teams that a particular system would be available per the overall project plan such that the next team in the workflow would be able to use it. Furthermore, the rich communications among team members worked well for a project of this size and with staff from similar cultures. Table 1 presents the initials of the team members, their roles, sites, and the components assigned. Figure 5 (A) shows the component and connector view of the UML-PM architecture, while Figure 5 (B) shows the DSM view of the UML-PM architecture.

3 Data Collection Approach

In this section, we describe the approach we used to collect the data presented in this paper. A questionnaire was developed to collect data about the social interactions in the project. The questionnaire was composed of 10 questions, where each team member was asked to identify the team members that interacted with him for the last two weeks, the frequency of interaction, and the technical content of the interaction. In addition, some of the questions probed social interactions outside of work. The questionnaire was web based and the answers were stored in a relational database. This *Social Network Analysis* (SNA) data was collected for the four two-week GSP V3.0 project phases described below. We used pre-canned SQL queries to breakdown the data for analysis in two-week ranges. For conciseness of presentation, we only present in Figure 1 the communications patterns between team members for the two-week phases ending on May 04 and May 18 in the form of SNA graphs. An arrow in Figure 1 from *team member A* to *team member B* represents that *A* remembers having communicated with *B* over the last two weeks. In this initial analysis, we don't show weights to represent frequency of communication or the quality of the communication between *A* and *B*.

- March 01, 2007 to April 14, 2007 - In this timeframe, the GSP V3.0 infrastructure deployment was completed, the project plan was developed, and the definition of the interfaces between components was finalized.
- April 15, 2007 to May 4, 2007 - (Figure 1 (A)) In this timeframe, the adapters between the major components were developed. Specifically, the TDE adapter, the Tangram adapter, and the PEPS adapter were developed. In addition, the integration testing team prepared for integration testing, and the Tangram web service was developed. The automated integration testing milestone occurred in this timeframe but was not on schedule.
- May 5, 2007 to May 18, 2007 - (Figure 1 (B)) In this timeframe, the team continued to prepare for the automation of the integration testing activities. The following activities from the previous iteration continued to take place: TDE adapter, Tangram adapter and PEPS adapter development, preparation for integration testing, web service development and deployment for Tangram. The automated integration testing milestone was still not achieved on schedule.
- May 19, 2007 to June 8, 2007 - In this timeframe, the team started integration testing activities. The following activities from the previous iteration continued to take place: TDE adapter, Tangram adapter, and PEPS adapter development, web service development and deployment for Tangram.
- June 9, 2007 to June 28, 2007 - In this timeframe, the team finalized the execution of integration testing.



(A) May 04, 2007 (B) May 18, 2007

Figure 1. GSP V3.0: Social Network Analysis

4 Modeling and Architectural and Coordination using Matrix Models

This section introduces the concept of *design structure matrix* (DSM), a.k.a. *dependency structure matrix*, and how DSM modeling can potentially help the designer understand the relations between the modular structure of the design and coordination among project team members.

4.1 Design Structure Matrix

The design structure matrix (DSM) was initially conceived by Steward [10] and developed by Eppinger et al. [7] as a means of modeling interactions between design variables of engineered systems. A DSM is a square matrix where the rows and columns are labeled with design dimensions in which decisions have to be made. A marked cell models the fact that the decision made on the row depends on the decisions made on the column. Baldwin and Clark proposed a modularity theory [2] in which they represent the dependency structure among design decisions using DSMs. For example, Figure 2 (A) shows a DSM modeling a design consisting of three decisions, A, B, and C. A and B depend on each other, and C depends on A. A and B can be seen as an algorithm and its associated data structure. C can be seen as a client that uses the algorithm that operates on the data structure.

A DSM has the following features that make it a suitable representation for both architectural and communication structures:

	A	B	C
A	.	x	
B	x	.	
C	x		.

(A) Sample DSM

	I	A	B	C
I	.			
A	x	.	x	
B	x	x	.	
C	x			.

(B) Splitting

Figure 2. Splitting Operation

1. A DSM represents design decisions explicitly, modeling both architectural decisions and environmental conditions [11]. For example, in Figure 5 (B), each component is represented as a design decision.
2. A DSM can represent the notion of *design rules*, proposed by Baldwin and Clark [2], as a means of decoupling otherwise coupled design decisions. Design rules specify the interfaces between modules, and appear at the left-hand side of the DSM. Figure 2 (B) shows the introduction of design rule I, which can be seen as the abstract data type that provides the interface for the module that consists of A and B.
3. A DSM shows how the introduction of design rules *decouple* the system into *modules*: after specifying I as an interface, C no longer depends on A and B. Instead, both the A and B module and the C module depend on

I. As a result, A and B become an independent module, and C is another independent module. In a DSM, independent modules appear as blocks along the diagonal. In a truly modularized design, all the dependencies appear between modules and design rules, and there are no dependencies among modules.

Baldwin and Clark define the behavior of introducing design rules that decouple two modules as the *Splitting* operation. Given two modules, A and B, resulting from the Splitting operation, experiments on A and B may be performed independently. In other words, module A can be replaced with a better module with advantageous properties, such as higher performance or lower cost, without influencing B. Module B can be similarly substituted with a better version without disturbing A. The ability to select the best candidate for each module increases the value of the entire system. Baldwin and Clark define the behavior of exchanging an existing module for a new module with advantageous properties as the *Substitution* operation. In other words, each module creates an option (to substitute), which will only be exercised when substitution is advantageous. Increasing the number of substitutable modules increases the number of system configuration options, which (under well-defined assumptions) results in higher value for the system as a whole.

4.2 DSM Modeling for GSP V3.0

To explicitly represent the architectural decisions and their relations in GSP V3.0, we derive its DSM as shown in Figure 5 (B). In this DSM, design rules are the interfaces of these components, modeled with variables starting with "dr_". The components are modeled with variables starting with "com_". The first block of the DSM models the design rules, and the following blocks along the diagonal model the independent component modules. In both Figure 5 (A) and (B), the components and interfaces are similarly numbered.

One assumption behind DSM modeling is that the design structure, composed of design decisions, is isomorphic to the task structure. After we represent a software architecture using a DSM, the DSM implies the coordination structure. A dependency in a cell models the fact that the decision-maker responsible for the decision on the row has to adhere to the decision on the column, implying possible communications between these decision-makers.

Given a DSM with design rules, we map the design rules and independent modules to the team members with different roles. The design rule decision-makers are the architects for the most important system modules. This implies that there was intensive communications among the architects during the architectural design stages. During the initial development stage, developers would have to communicate with the architects to understand the design rules, implying communications between the developers and architects. When the design rules became stable and the system was more ideally modularized, there was less communication among the lead architects.

4.3 Coordination Modeling in GSP V3.0

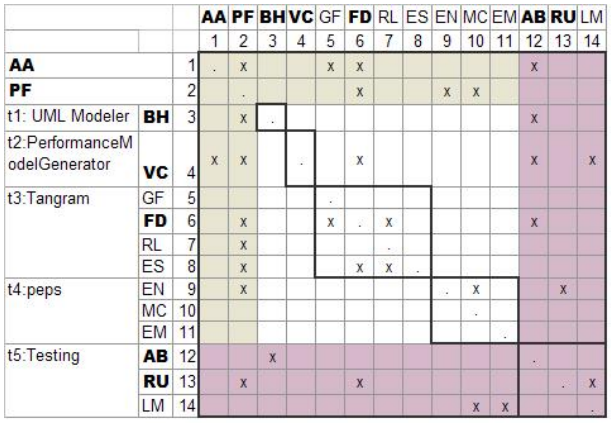
The coordination structure among team members at each process stage can also be represented in a matrix form. The coordination matrix is derived from the social network from the survey and questionnaire. In such a matrix, we label the rows and columns with the teams members. If developer A initiates a communication to developer B, then we mark the cell on row A and column B. We call such a matrix a *coordination matrix* (COM). Figure 3 (A) shows the COM during the initial development stage, when the architecture was designed. Figure 3 (B) shows the COM during the second development stage. Figure 4 (A) shows the COM during the third development stage, at the start of integration testing phase. Figure 4 (B) shows the COM at the end of the integration testing phase.

We cluster these matrix models according to component teams. All the members responsible for the same component are clustered into the same block. AA and PF constitute the first block. They do not belong to any specific teams, but oversee the project. The next blocks are the UML modeling team, PerformanceModelGenerator team, Tangram Team, and Peps Team. The last block is the testing team. We label the rows and columns with the initials of the team members, and distinguish the initials of all the architects, that is, design-rule decision-makers, using bold font (please refer to Table 1 for the role of each team member). From the COMs, we can see that all local teams have at least one architect, except the Peps Team.

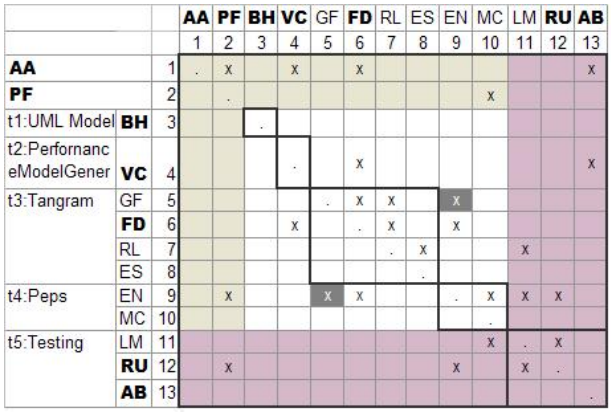
4.4 The Comparison of DSM and COMs

Because a DSM represents the decision structure, we hypothesize that the dependencies among the decisions drive the need for communication. To test this hypothesis, we compare the COMs shown in Figure 3 and 4 with the architectural DSM shown in Figure 5 (B) as described below.

The light grey parts of the COMs model the communications through AA and PF. Since they do not belong to any particular team but provide oversight to the whole project, it is normal that they communicate with all other teams, generating crosscutting marks. The dark grey parts of the COMs model the communications through the integration testing team. Because this team tests all the components and their interactions, their communications with other teams also appear to be highly crosscutting. As a result, we mainly compare the middle part of each COM (with white background), with the architectural DSM. Note that the architectural DSM in Figure 5 (B) is asymmetric: its upper-right part is blank, modeling that the components do not influence the design rules. If a design rule decision-maker and a component team member talk to each other, there will be symmetric marks in the COMs (Figures 3, 4).

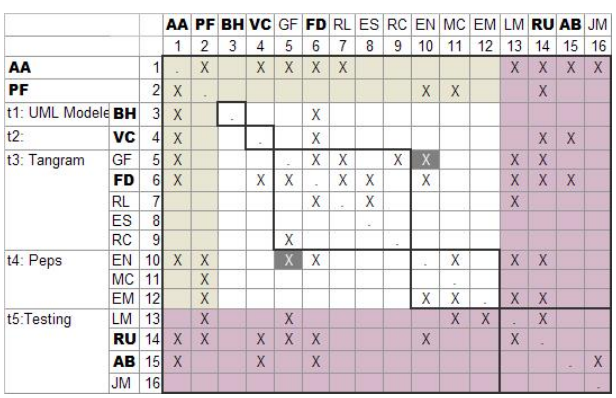


(A) May 04, 2007

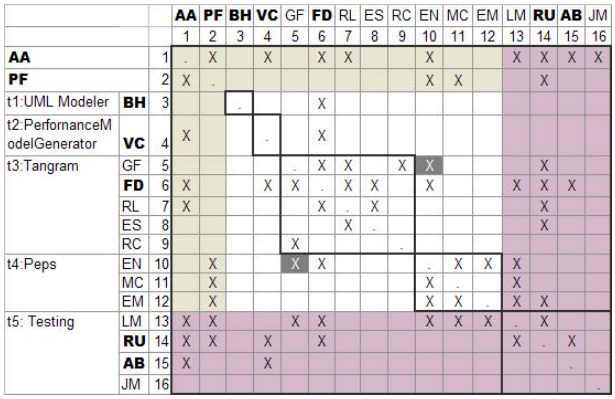


(B) May 18, 2007

Figure 3. The Communication Structure During the First and Second Development Stages



(A) June 08, 2007



(B) June 28, 2007

Figure 4. The Communication Structure During the Third and Fourth Development Stages

The architectural DSM shows a well-modularized pattern: (1) the only off-module dependencies are between components and design rules, and (2) there are no off-block dependencies among these component modules. We consider a COM to be consistent with the architectural DSM if (1) the off-team communications only happen through design rule decision-makers (architects), and (2) there are no off-team communications besides that.

4.5 Comparison Results

We studied the DSM and COMs of the Global studio project to answer the following research questions: **RQ1:** Is the architecture structure consistent with the coordination structure that actually happened? **RA1:** The four COMs appear to be consistent with the architectural DSM with only one exception. We compare the DSM and COMs in detail with the following three research questions.

Design rules in an architectural DSM are the key to decompose the system into modules. Accordingly, we expect to find higher levels of communications centered around the architects, that is, the design rule decision-makers, especially at early development stages, leading to the following research question: **RQ2:** Do the design rule decision-makers have intensive and cross-team communications? **RA2:** The answer is yes. From the COMs, we observe that (1) the project manager, AA, communicated with most other teams, which is consistent with his project leading role; (2) with only one exception, all the off-team communications happened through architects (through the rows/columns labeled with bold font initials.)

In an architectural DSM showing a truly modularized design, each module should be independent in that the dependencies only exist within each module or between design rules, and there should be no dependencies among independent modules. Accordingly, we expect intensive communications within individual component teams, but not between these teams. If a team member is not an architect, he/she should not need to communicate with other non-

architect members of other teams. This analysis leads to the following research question: **RQ3:** Do the developers of the same architecture module only coordinate with each other, and not with non-architect members from other teams? **RA3:** The answer is yes with only one exception. Most of the communications were indeed within local teams, except that GF talked to EN after the first stage, as shown in the cells with dark background and white mark in Figure 3 and 4. These exceptions are caused by the fact that the Tangram team has more experience with web services and has an architect, FD, while the Peps team has less experience and has to learn from the Tangram group by talking to FD and GF. FD is an architect, but EN is located physically closer to GF than to FD (both GF and EN were located in Brazil).

This leads to an interesting observation. A key feature of the GSP V3.0 project process is that the architects were distributed in that they were from different remote sites and competence centers, each having different expertise and responsible for a different component. This is different from previous GSP projects where all the architects were located in a central site. However, we notice that there is no architect (design rule decision-maker) in the Peps team. As a result, the leader of this team (EN) has to communicate with the architect or members from another team. This observation verified our intuition that having a focal architect within each remote site and component team is essential to minimize the communication between teams. We further observe that having a focal architect within each remote site is advantageous when the software architecture is well-modularized in that after the design rules are stabilized, a team member can obtain all the information needed to accomplish the task within the local team, and does not need to send inquiries to members of another team. In the GSP 3.0 process, having a focal architect does not mean that one can only contact another team through the focal architect. Instead, when there is a need, one can contact anyone from any other team. From the communications that actually happened during the project, we observe that cross-team communications were very rare because the architecture was modular.

Finally, the communication pattern may change at various stages of development, leading to the following research question: **RQ4:** How stable is the coordination structure over time? **RA4:** The coordination structure turned out to be very stable in terms of internal and cross-team patterns. We observed the same patterns through all four communication matrices. During the last two stages of the project, more and more communications were among local team members, and less among the distributed architects, indicating that the design rules became more stable over time.

5 Results Discussion

The GSP V3.0 project was completed successfully, as it achieved its main goal on schedule and within cost. We now present the factors we believe contributed to project success.

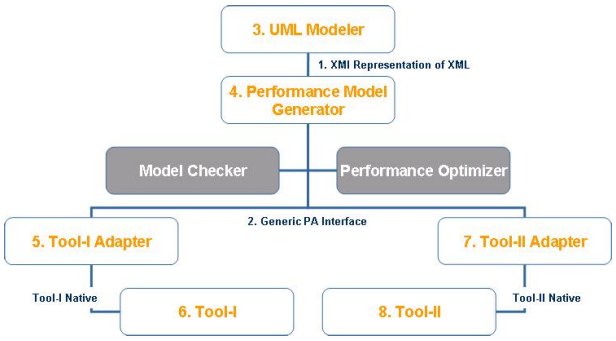
5.1 Architecture

Figure 5 (A) shows the component and connector view of the architecture. It shows that the UML Modeler connects to the Performance Model Generator using the XMI representation of XML. It also shows that the Peformance Model Generator uses the PMIF 2.0 XML connector to communicate to the Tool-I adapter and to the Tool-II adapter. The Tool-I and Tool-II adapters use the native language of the Performance Modeling Solvers to communicate to Tool-I and Tool-II. The lines in Figure 5 (A) represent client/server run-time interactions. The interaction is initiated by the component upstream, i.e. closer to the customer, to the component downstream, i.e. closer to the performance modeling solver tools.

The rationale for the selection of these interfaces was the following:

- The adoption of industry standard interfaces as implemented by Mosquito enabled the identification of each version and each release of each interface to use,
- The development of adapters to the Mosquito interfaces enabled the reuse of large components that existed at the start of the project,
- The implementation of web-services to wrap the existing components enabled each remote site to continue to use its local environment for developement.

The adoption of the XMI and PMIF 2.0 standards identified Mosquito as the focus of the architecture. Once the Mosquito web service was made available to the other components, all the components were able to quickly integrate with Mosquito. Because the Mosquito tool architect had implemented the XMI representation of XML and the PMIF 2.0 interfaces, the project had a resource to call on to resolve technical issues related to Mosquito interfaces. Therefore, the identification of a stable component as the focus of the architecture constrained the technical communications and provided a mechanism to quickly resolve architecture issues. In addition, the use of web services to wrap existing components enabled the use of an heterogenous environment for component development and for component execution in production. For example, Mosquito and TDE run as Eclipse plug-ins, one of the tool adapters runs under Linux, while the other tool adapter runs under .NET. We attribute the selection of these design rules as one of the key enablers for project success from the architecture perspective.



(A) Component and Connector View

			1	2	3	4	5	6	7	8
Design Rules	dr_XMLRepresentationOfUML	1	.							
	dr_GenericPAInterface	2		.						
UML Modeler	com_UMLModeler	3	x		.					
PMG	com_PerformanceModel Generator	4		x		.				
PEP	com_pep_adapter	5		x			.	x		
	com_pep	6					x	.		
Tangram	com_tangram_adapter	7		x					.	x
	com_tangram	8							x	.

(B) DSM View

Figure 5. GSP V3.0 Architecture

5.2 Process

The extended workbench model process [9] for global software development differentiates between resources located in the central and remote sites in the following ways:

- Central site resources are usually more skilled (or experienced) than remote site resources,
- Central site resources are usually assigned requirements and architecture (early phase) related tasks,
- Central site resources are usually more costly (hourly labor rate) than remote site resources,
- Central site resources tend to be more trusted by upper management with important project tasks than remote site resources.

When using the extended workbench model, central site resources tend to become bottleneck resources in large software development efforts, and cost more than the remote site resources. In this paper, we report on initial results of the experimentation with a novel "system of systems" process for global software development which employs domain experts located at the remote sites. This process for global software development attempts to solve some of the problems that arise for coodination of global software development when remote teams are composed of less experienced personel than the central sites. Our initial results are positive and seem to indicate that global software development projects could be structured to take advantage of the existing domain expertise located in the remote sites with the following benefits: more scalable as the central site is not as overwhelmed by remote site requests, enables distribution of important tasks to the remote sites, and creates trust between upper management and remote sites.

In the following subsection we will summarize the actions taken by the GSP V3.0 management to create a strong partnership between the central site and the remote sites.

5.3 Management

The main project management factors for success are usually team building, communications and culture. We now describe how these factors were addressed in GSP V3.0 to ensure project success. The objective of our team building activities was to build a high-performing team with a feeling of parternship. The project was initiated by having a face-to-face kickoff meeting where the key team members were present. After three weeks, a face-to-face architecture meeting was held at the central team's site. The project manager followed up with a face-to-face meeting with a key team member that could not be present at the kick-off meeting. In addition, a key team member in charge of Integration Testing was resident at the central site for one month. The project used a shared document and code repository for sharing information and weekly telecoms to address issues. All project members were invited to attend the weekly telecoms and most telecoms were well attended. We believe the high level of commitment oberved throughout the project was due to the strong personal relationships that were created by the early team building activities. We observed an evolution of the communication patterns among team members depending on the actual phase the project was in. Initially, most of the communication was among collocated team members and between the members resident at or visiting the central team. We concluded that the reason for this pattern of communication was the lack of knowledge of the work and expertise located at the remote teams. As the project gathered momentum, we saw a shift in the pattern of communications from site-based to work-related communications based on need. Finally, as the project reached the critical milestone for deliveries we saw even greater communication based on need and less communication based on site. The project had several micro-cultures of a few similar cultural domains, so culture shock was not as strong as it is usually found when team members are from very distinct cultures. In the few instances that cultural problems did occur, it was quickly resolved through intensive communication.

We now describe some of the strategies used in GSP V3.0 to overcome obstacles that are usually found in global

software development efforts [9]. The central site was located on the East Coast of the USA and the remote sites were located in Brazil and Italy. The time difference between Brazil and the USA varied from 1 to 3 hours as the USA and Brazil changed from Standard Time to Daylight Savings Time. The time difference between Italy and USA was 6 hours. We did not observe a significant impact of these time differences in project productivity as the central site was able to communicate frequently with both teams in Brazil and Italy. In GSP Versions 1 and 2, there was a team in India with a greater time zone difference. Most of the team members were fluent in English, and could understand Portuguese and Italian. Some of the code documentation was originally written in Italian and was studied by Portuguese speaking team members, while these documents were being translated into English. The central team had project management and architecture leadership responsibilities and was easily accessible by all team members. In addition, each local team had local team leaders and a local manager. The project was composed of two Siemens Companies (SCR, Chemtech) and several universities (L'Aquila, PUC-RS, COPPE). The shared feeling of partnership and the understanding of the common goal was apparently enough to overcome the several company and university cultures. Team members traveled at the start of the project to establish personal connections. Good collaboration tools were used (Gforge, SVN, Web-ex, wiki). Teleconferences were short and had a very structured agenda. A sense of urgency was created by defining a short four-month project with a well defined and achievable goal that was understood by all team members and reinforced at the weekly teleconferences. Project management was very focused on managing change requests and avoiding scope creep that would deviate from the project's main goal.

6 Conclusions

In this paper, we have presented an experience report of the assessment of the coordination implications of software architecture in a global software development project.

We conclude that GSP V3.0 employed an architecture that enabled an efficient communication structure. We observe that architectural DSM modeling has the potential to guide task assignments and team coordination. We envision a procedure as follows: given a software architecture model represented using one of the prevailing modeling techniques, such as UML or ADLs, the user can identify the design rules, derive the architectural DSM, and map the DSM to task assignments, assigning each module to a distributed team. Our results are still very preliminary given that they are based on a single experiment. We will continue to conduct comprehensive empirical studies to explore the relation between software architecture and communication requirements.

We have also reported on our experience using a novel process for global software development, which employed domain experts located at remote sites to help enable collaboration among remote teams. This process for global software development attempted to solve some project management problems that arise in the coodination of global software development projects when the remote teams are composed of less experienced personel than the central site. We conclude that global software development projects could be structured to take advantage of domain expertise located in remote sites to create a more scalable environment, as the central site experts would not be overwhelmed by remote site requests. This additional scalability is achieved because our process enables the distribution of important tasks to remote sites. We are continuing and extending our research by evaluating architectures to enable efficient use of the UML to Performance Modeling tool at customer sites and by adding usability features to the UML editor to facilitate performance requirements input.

References

[1] A. Avritzer, W. Hasling, and D. Paulish. Process investigations for the global studio project version 3.0. In *Proc. IEEE International Conference on Global Software Engineering*, Los Alamitos, CA, 2007. IEEE Press.

[2] C. Y. Baldwin and K. B. Clark. *Design Rules, Vol. 1: The Power of Modularity*. The MIT Press, 2000.

[3] Y. Cai. *Modularity in Design: Formal Modeling and Automated Analysis*. PhD thesis, University of Virginia, Aug. 2006.

[4] Y. Cai and K. Sullivan. Simon: A tool for logical design space modeling and analysis. In *20th IEEE/ACM International Conference on Automated Software Engineering*, Nov. 2005.

[5] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 353–362, New York, NY, USA, 2006. ACM Press.

[6] V. Cortellessa, P. Pierini, and D. Rossi. Integrating software models and platform models for performance analysis. *IEEE Trans. Software Eng.*, 33(6):385–401, 2007.

[7] S. D. Eppinger. Model-based approaches to managing concurrent engineering. 2(4):283–290, 1991.

[8] D. Paulish. *Architecture-Centric Software Project Management*. Addison-Wesley, Boston, MA, 2002.

[9] R. Sangwan, Bass, N. M., Mullick, D. Paulish, and J. Kazmeier. *Global Software Development Handbook*. Auerbach Publications, Boca Raton, FL, 2007.

[10] D. V. Steward. The design structure system: A method for managing the design of complex systems. *IEEE Transactions on Engineering Management*, 28(3):71–84, 1981.

[11] K. Sullivan, Y. Cai, B. Hallen, and W. G. Griswold. The structure and value of modularity in software design. *SIGSOFT Software Engineering Notes*, 26(5):99–108, Sept. 2001.