Coordination implications of software architecture in a global software development project

Alberto Avritzer\textsuperscript{a,*}, Daniel Paulish\textsuperscript{a}, Yuanfang Cai\textsuperscript{b}, Kanwarpreet Sethi\textsuperscript{b}

\textsuperscript{a} Siemens Corporate Research, 755 College Road East, Princeton, NJ 08540, USA
\textsuperscript{b} Dept. of Computer Science, Drexel University, Philadelphia, PA 19104, USA

\section{Abstract}

In this paper, we report on our experience of using design structure matrices (DSMs), derived from architecture models developed at early stages of the project, to reveal the coordination needs among globally distributed development teams. Our approach is to automatically transform the box-and-line style software architecture model into an augmented constraint network (ACN), from which a DSM can be automatically generated. After that, we represent the coordination structure among the team members as communication matrices (COMs). We then assess the consistency between the DSM and COMs. Analysis of data gathered during the Global Studio Project Version 3.0 revealed that the architectural DSM model, representing the software modular structure, is highly consistent with the COMs that represent the actual coordination structure, showing that an architectural DSM has the potential to help guide the task assignments in global software development projects.

\section{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 spread 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 (Sangwan et al., 2007). This paper reports on the GSP experience during the third year of the project, referred to herein as GSP V3.0.

One of the key research questions the GSP projects aim to study is the relationship between software architecture and the communication needed between remote teams. For many years, software project organizations have often been structured in accordance with the system architecture design of the product being developed (Paulish, 2002). 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.

Studies have been made on the relationship between dependency structure and communication. Cataldo et al. (2006) empirically studied the congruence between coordination structure and the dependency structure caused by modification requests. Bass et al. (2007) pointed out that considering only dependencies caused by invocation is not sufficient. Instead, more general architectural decisions should be taken into account. To assess and align task assignment with software architecture, we need a general model that can express software architecture as a set of interdependent design decisions, and can explicitly predict the needs of communication among architects and developers at early development stages. In this paper, we focus on high-level architectural decisions such as the major components, their interfaces and the major standards employed.

A problem is that prevailing software architecture models, such as UML component diagrams and architecture description languages (ADLs) are not designed to represent both software architecture structure and its associated organizational structure, that is, the communication structure of its developers. In this paper, we report on our experience of using an emerging software model, the design structure matrix (DSM) (Baldwin and Clark, 2000; Eppinger, 1991; Steward, 1981), to represent an architectural structure as interdependent design decisions, and to assess...
We use a DSM to represent software architecture for several reasons. First, a DSM can uniformly represent both explicit components and implicit decisions, such as naming conventions and standards in use. Second, unlike prevailing architectural models, a DSM can express the asymmetric dominance relations among decisions. Baldwin and Clark (2000) define dominating decisions, such as interfaces between components, as design rule decisions. Finally, a DSM reveals the modular structure of an architecture, which is usually not equivalent to the boxes in traditional box-and-line style architecture models. These features allow us to directly map decisions to decision-makers, both developers and key architectures (design rule decision-makers), map architecture dependencies to both inter- and inner-team communications, and assess the consistency between them.

In this paper, we also present a method of formally transforming box-and-line style software architecture models into a logical model called the augmented constraint network (ACN) (Cai, 2006), from which a DSM with rigorous semantics can be automatically derived (Cai, 2006; Cai and Sullivan, 2005). The architecture diagram of GSP V3.0 is already available. We developed a tool set to automate this transformation and DSM generation. In the transformation procedure, each component is formalized as a set of variables modeling design decisions, and their interfaces are formalized as design rules. We also automate the procedure of deriving the communication structure of GSP v3.0 and modeling it as a matrix model, which we call a communication matrix (COM).

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 project. We have derived COM models from the SNA questionnaires using data related to structure, importance and frequency of the communications.

The DSM and COMs revealed that the modular structure of the architecture is highly consistent with the communication structure. It is clear, by comparing DSMs and COMs, 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 remote teams. The conclusions from this experience report are that DSM modeling, transformed from high-level architecture diagrams developed at early development stages, has the potential to help guide task assignments and team coordination. Furthermore, the coordination process of GSP V3.0 is efficient, as indicated by the DSM and COM models, 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 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 review the concepts of the models used in our experiments. Section 4 presents the formalization and transformation from a box-and-line style architecture diagram to a DSM. Section 5 presents the research questions we aim to study. In Section 6, we describe the experimental procedure. In Section 7, we discuss the results of our case study. In Section 8, we discuss our results. In Section 9, we present our review of related work, and in Section 10, we present our conclusions and topics for future research.

2. Global Studio Project V3.0

In this section, we briefly introduce the development processes and organizational structure used in the Global Studio Projects, as well as the case study of GSP 3.0.

2.1. Development process

Different development processes were used during different years of the Global Studio Project (Avritzer et al., 2007).

The "extended workbench model" development process, used during the first two years of the GSP, had the following characteristics (Avritzer et al., 2007):

- 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.
- 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 to 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 software systems up to 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 2 years. For GSP V3.0, 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 requirements 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 requirements specifications.

The GSP V3.0 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, the test cases/use cases can be developed by inspecting the existing systems’ user interface.

2.2. **State of the practice**

Today’s software project managers have a large number of possible ways to structure their GSD project 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:** The architecture decomposes the system into components and the components are allocated as work packages to the different sites.

• **Process steps:** 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.

• **Release:** 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.

• **Platform:** 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.

• **Competence center:** 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.

• **Open source:** 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 2 years of the GSP, a product structure approach was used to structure the organization and an extended workbench model development process was used (Sangwan et al., 2007). 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.

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

• **Performance Model Generator:** L’Aquila University 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, Italy (Cortellessa et al., 2007) 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 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 as compared to 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. 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.

3. **Background**

In this section, we briefly introduce the key models and theory that form the basis of our assessment approach: the prevailing architecture models, the design structure matrix (DSM) model, and the augmented constraint network (ACN).

3.1. **Box-and-line style architecture models**

The unified modeling language (UML) and informal box-and-line diagrams are widely used in practice to model software architecture. Architectural description languages (ADLs) are also widely
The design structure matrix (DSM) was initially conceived by Steward (1981) and further developed by Eppinger (1991) 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 for which decisions have to be made. A marked cell models the fact that the decision made on the row depends on the decision made on the column. Baldwin and Clark proposed a modularity theory (Baldwin and Clark, 2000) in which they represent the modular structure among design decisions using DSMs.

For example, Fig. 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, such as a sorting algorithm operating on an array data structure. C can be seen as a client that uses the algorithm, which is specific to the array-based implementation. The boxes along the diagonal represent the modules clustered with a set of design decisions. Different from prevailing box-and-line style architecture models, a DSM has the following features:

First, a DSM uniformly represents both concrete components and abstract decisions, as well as environmental conditions (Sullivan et al., 2001). For example, in a DSM model, each component in Fig. 1 will be represented as a design dimension.

Second, a DSM can represent the fact that some key decisions, design rules (Baldwin and Clark, 2000), dominate other subordinating decisions. In other words, design rules influence subordinating decisions, but are not influenced by them. As a result, the marks between design rules and other decisions are asymmetric. The introduction of design rules is a means of decoupling otherwise coupled design decisions into modules.

Fig. 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. For example, I can be a list data type that is independent of the underlying data structure and algorithm in use. As a result, the module with A and B will depend on I because this module will have to implement I. Consequently, C only needs to depend on I, but not the concrete implementation of I, that is, the module of A and B. 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. The distinction between design rules and subordinating modules makes it easy for us to map the architecture structure to the hierarchical structure within an organization. Key design rules usually are decisions made.

### Table 1

<table>
<thead>
<tr>
<th>Member/arch role/team role</th>
<th>Site</th>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA/arch/lead</td>
<td>SCR/US</td>
<td>Overall project</td>
</tr>
<tr>
<td>AB/arch</td>
<td>SCR/US</td>
<td>Performance</td>
</tr>
<tr>
<td>BH/arch</td>
<td>SCR/US</td>
<td>UML Modeler</td>
</tr>
<tr>
<td>FD/arch/lead/lead</td>
<td>L’Aquila/Italy</td>
<td>Performance Model</td>
</tr>
<tr>
<td>GC/arch/lead</td>
<td>Chemtech-Rio/Brazil</td>
<td>Tangram web service</td>
</tr>
<tr>
<td>RC/lead</td>
<td>Chemtech-Rio/Brazil</td>
<td>Tangram web service</td>
</tr>
<tr>
<td>RL/lead</td>
<td>COPE/EFRJ-Rio/Brazil</td>
<td>Tangram</td>
</tr>
<tr>
<td>ES</td>
<td>COPE/EFRJ-Rio/Brazil</td>
<td>Tangram</td>
</tr>
<tr>
<td>PF/arch/lead/lead</td>
<td>PUC-RS/Brazil</td>
<td>PEPS</td>
</tr>
<tr>
<td>EN/lead</td>
<td>PUC-RS/Brazil</td>
<td>PEPS</td>
</tr>
<tr>
<td>MC</td>
<td>PUC-RS/Brazil</td>
<td>PEPS</td>
</tr>
<tr>
<td>EM</td>
<td>PUC-RS/Brazil</td>
<td>PEPS</td>
</tr>
<tr>
<td>JM</td>
<td>Zaragoza/Spain</td>
<td>Performance</td>
</tr>
<tr>
<td>RU/arch/lead</td>
<td>PUC-RS/Brazil</td>
<td>Testing</td>
</tr>
<tr>
<td>LM</td>
<td>PUC-RS/Brazil</td>
<td>Testing</td>
</tr>
</tbody>
</table>

![Sample DSM](image1)

![Sample DSM](image2)

**Fig. 1.** Sample UML component diagram.

**Fig. 2.** Splitting operation: (A) sample DSM and (B) splitting.
by architects leading subordinating teams. Once design rules are agreed upon, ideally, the teams working on independent modules should be able to work in parallel.

3.3. Augmented constraint network

It is possible to transform any box-and-line style diagram into a matrix format directly, making boxes as vertexes and lines as edges. There are several problems with DSMs generated this way. First, certain kinds of “lines” in an architecture diagram, such as connectors in an ADL, embody important architectural decisions that deserve to be considered as design dimensions, shown as columns and rows in the DSM. Second, although components and their interfaces and ports are usually shown together in an architecture diagram, their roles in an architecture are dramatically different: interfaces are decisions visible to other components and usually dominate decisions at component level. Finally, the semantics of pair-wise dependency are not clear: if A depends on B, while B depends on C or D, then is there a dependency between A and C or D?

Cai (2006); Cai and Sullivan (2005, 2006) have developed a logic-based design modeling technique, called the augmented constraint network (ACN) that addresses these problems and formalizes the DSM modeling and design rule theory. An ACN consists of a constraint network (CN) that models the logical relations among design decisions, a dominance relation (DR) that formalizes the notion of design rule and the asymmetric relation among design decisions, and a cluster set (CS) in which each cluster models one way a design can be aggregated into modules, similar to the different ways the rows and columns of a DSM can be reordered. A constraint network consists of a set of design variables (Var), their domains (Dom), and a set of constraints (Con). Formally: \( ACN = (CN, DR, CS) \) and \( CN = (Var, Dom, Con) \).

Similar to DSM modeling, ACN modeling entails paradigm-agnostic means to compute modularity properties of a design, and provides precise definition for the pair-wise dependence relation (PWDR): if a design variable \( y \) depends on variable \( x \), that is, \( (x, y) \in PWDR \), then there must exist a consistent state of the ACN. Changing the value of \( x \) will violate some constraints and make the constraint network inconsistent, and the value of \( y \) has to be changed in one of the ways to restore the consistency. A DSM can be thus automatically generated: the computed PWDR is used to populate the matrix and a selected cluster is used to order the columns and rows of the DSM. In the next section, we introduce the automatic transformation from architectural diagrams to ACNs.

4. Transforming architectural diagrams into decision models

In this section, we introduce our approach of transforming prevailing box-and-line style architecture diagrams into augmented constraint networks (ACNs), which can then be used to generate design structure matrices (DSMs) with rigorous semantics. Our idea is to transform boxes or lines that embody design decisions into design variables, transform the assumption relations among these decisions into logical expressions, and derive the dominance relation among these decisions according to their semantics in the original diagram.

In this section, we use the transformation from a UML component diagram to an ACN model to illustrate our approach, which can be applied to other box-and-line style architecture models, such as a component-and-connector diagram. We choose UML component diagrams because they are standardized, supported by widely accepted tools, such as Rational Rose, and the semantics of the diagram are well-understood. Our automatic transformation tool takes XML files as input. For other types of architectural diagrams, as long as the diagrams can be expressed using the same format, their DSM models can be automatically generated.

In a UML component diagram, we view each component as having two types of design dimensions, its provided interfaces that are visible to other parts and the decisions hidden within the component. We thus model a component with a provided interface using two variables, a component and an interface variable. For example, the Component A in Fig. 1 is formalized as two variables: ComponentA, and ComponentA_Interface. A component can have multiple provided interfaces, and thus can be modeled using multiple variables. To model the fact that any decision in any dimension is possible to change in the future, we assign each variable at least two values, orig and other, to model the current decision and a possible change. Component A is thus modeled as a variable with a domain having two values: ComponentA: \{orig, other\}.

We do not model the required interfaces of a component. Instead, we model the required relation as a logical constraint. For example, if Component A requires ComponentB_Interface, it means that in order to implement A, one has to make assumptions about how B’s interface is designed. We use the following constraint to model this assumption relation:

\[ \text{ComponentA} = \text{orig} \Rightarrow \text{ComponentB\_Interface} = \text{orig}. \]

This logical expression dictates that changes in B’s interface will definitely influence the design of Component A, but not vice versa.

A logical expression is symmetric in that changing the value of either variable can influence the value of other variables in the same expression. However, software decisions are not symmetric. For example, if the interface of a component has been used by other components designed by other designers, it could be too expensive to change the interface. That is, an interface can dominate a component. According to Baldwin and Clark, these dominating interfaces are design rules.

In a UML component diagram, the dominance relation can be derived from its semantics. Every interface that is provided or required by a component, dominates that particular component. This means that a change in the interface may propagate to the component, but a change in the component can not propagate to the interface. The following sample pairs are the members of the dominance relation of the automatically generated ACN of the sample UML component diagram:

\[ \text{ComponentA\_ComponentB\_Interface}; \]
\[ \text{ComponentA\_ComponentA\_Interface}; \]
\[ \text{ComponentB\_ComponentA\_Interface}; \]
\[ \text{ComponentB\_ComponentB\_Interface}. \]

According to the above formalization, we built a utility tool to automate the transformation from a UML component diagram to an ACN. Given a component diagram represented using Rational Rose, we export the model into an XML file. After that, we translate the XML file into an ACN that can be accepted and processed by our tool called Simon (Cai and Sullivan, 2005), which will automatically generate a DSM. The DSM can be clustered so that all the design rules are aggregated into the first block and thus reveal the resulting modules.

5. Research questions

The purpose of our experiment is to assess how well a DSM model, generated from an architecture model at early development
stages, can predict the communications needed. Concretely, we aim to study the following research questions:

RQ1: Is the architecture structure consistent with the coordination structure that actually happened? If the architectural DSM can sufficiently predict the communications needed, the two types of DSM should appear to be consistent in that if there are communications recorded between groups, there should be dependencies between corresponding architectural modules in the architectural DSM. On the other hand, if there are no dependencies between two architectural modules, there should be no communications recorded between the developers in charge of these corresponding modules.

RQ2: Do the design rule decision-makers have intensive and cross-team communications? Design rules in an architecture are key decisions to decompose the system into modules, and thus influence multiple modules. Accordingly, we expect to observe higher levels of communications centered around the architects who determine the design rules, especially at early development stages, and that the communications to and from design rule decision-makers crosscut multiple modules.

RQ3: Do the developers of the same module only coordinate with each other, and need no communications with non-architect members from other teams? If an architectural DSM shows 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, from the communication structure, we expect to observe high consistence with the architectural DSM. That is: intensive communications should be within each team responsible for an architecture module in the DSM, but not between these teams. If a team member is not a design-maker, he/she should not need to communicate with other non-architect members of other teams.

RQ4: How stable is the coordination structure over time? The communication pattern may change at various stages of the project. How the relations will change between the architectural DSM and the communications happened?

RQ5: Can the architectural DSMs reflect the importance level and frequency of the communication between teams? For example, is it true that more important and more frequent communications happen to-and-from design rule decision-makers?

6. Case study

In Section 2 we have briefly introduced the subject of our experiment, GSP V3. In the following subsections, we describe the approach we used to collect communication data, the architecture model, and the experimental procedure.

6.1. The architecture of GSP v3.0

Fig. 4(A) shows the component-and-connector diagram of the GSP 3 architecture. The boxes represent components and the lines represent connectors. It shows that the UML Modeler connects to the Performance Model Generator using the XML representation of UML. It also shows that the Performance 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 Fig. 4(A) represent client/server run-time interactions. The interaction is initiated by the component upstream (closer to the customer) to the component downstream (closer to the performance modeling solver tool).

In other words, the architecture diagram says that the user should first model the system as standard UML diagrams using the first component (UML_Modeler). The output of the first component, in the format of XML, then serves as the input of the second component so that a performance model can be automatically generated. The Tool-I and Tool-II are both performance model solvers that were developed based on different platforms for different purposes, but are supposed to be used interchangeably. Both the output of the performance model generate and inputs of the two adapters conform to the PMIF 2.0 UML standard.

This diagram is not represented using a standard architectural description language supported by an ADL tool. Instead, the architecture is represented in a straightforward way easily understandable by all the five globally distributed teams. Not only boxes are design dimensions, all the lines embody important decisions. For example, the decision of using PMIF 2.0 XML is a key architectural decision that determines how the two adapters can work with the Performance Model Generator (Mosquito). Another important decision, not even shown in the diagram, is that all the components should be wrapped as web services to improve interoperability.

The rationale behind these decisions, made by the chief architects of the project, 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 development.

The adoption of XML 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 XML representation of XML and the PMIF2.0 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.

6.2. DSM modeling for GSP V3.0 architecture

As introduced in Section 4, we have developed a tool that can transform a box-and-line style architecture diagram into ACN and DSM models. Since the architecture of the GSP V3.0 was in an informal manually-drawn diagram, we formalize it as follows, recorded in an XML format acceptable by our tool. First, we model both components and connectors as design variables modeling design dimensions where decisions are needed. For example, a PMIF connector is modeled as a variable: \( \text{dr_PMIF}(\text{XMLV20}, \text{other}) \), showing that the current decision is to use XML 2.0, and this dimension has other unspecified possibilities. To model the fact that the Pep Adapter has to conform to the PMIF standard, we create the following constraint:

\[
\text{com_pep_adapter} = \text{orig} \implies \text{dr_PMIF} = \text{orig}.
\]
All these connectors are important design rules respected by more than one component. Another important design rule is the decision to wrap all the components into web services. We thus model all these design rules using variables starting with “dr_”. The dominance relation of the ACN thus have 13 pairs. For example, the following pair (com_pep_adapter, dr_PMIF) in the dominance relation shows that the decision about how the Pep adapter should be implemented should not influence the decision about which PMIF version to use. The constraint network part of the derived ACN is shown in Fig. 3.

We derive the architectural DSM of GSP 3.0 as shown in Fig. 4(B). The components are modeled with variables starting with “com_”. The first block of the DSM models design rules, and the following blocks along the diagonal model independent component modules. 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 follow 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. This

DesignSpace arch_dsm{
  dr_XML: {orig, other};
  dr_PMIF: {XMLV20, other};
  dr_WebService: {orig, other};
  com_UMLModeler: {orig, other};
  com_PerformanceModelGenerator: {orig, other};
  com_pep_adapter: {orig, other};
  com_pep: {orig, other};
  com_tangram_adapter: {orig, other};
  com_tangram: {orig, other};
  com_PerformanceModelGenerator = orig => dr_XML = orig;
  com_UMLModeler = orig => dr_XML = orig;
  com_PerformanceModelGenerator = orig => dr_PMIF = XMLV20;
  com_pep_adapter = orig => dr_PMIF = XMLV20;
  com_tangram_adapter = orig => dr_PMIF = XMLV20;
  com_pep_adapter = orig => com_pep = orig;
  com_tangram_adapter = orig => com_tangram = orig;
  com_UMLModeler = orig => dr_WebService = orig;
  com_PerformanceModelGenerator = orig => dr_WebService = orig;
  com_pep_adapter = orig => dr_WebService = orig;
  com_tangram_adapter = orig => dr_WebService = orig;
}

Fig. 3. GSP V3 ACN model.

Fig. 4. GSP V3.0 architecture: (A) component and connector view and (B) DSM view.
implied intensive communications among the architects during the architectural design stages. During the initial development stage, developers communicate with the architects to understand the design rules, implying communications between developers and the architects. When the design rules become stable and the system is ideally modularized, there is less communication to-and-from the lead architects.

6.3. Communication data collection

A questionnaire was developed to collect data about the social interactions among the 5 globally distributed development teams involved 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 2 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 2-week GSP V3.0 project phases described below. We used pre-canned SQL queries to analyze the data in two-week periods. We present in Fig. 5 the communication patterns between team members for the 2-week periods ending on May 04 and May 18 as SNA diagrams. An arrow in Fig. 5 from team member A to team member B represents that A remembers having communicated with B over the past 2 weeks. The time periods analyzed included:

- March 01, 2007–April 14, 2007: In this time frame, 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–May 4, 2007: In this time frame, 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 was planned for this time frame but was not achieved on schedule.
- May 5, 2007–May 18, 2007: In this time frame, the team continued to prepare for the automation of the integration testing activities. Activities from the previous iteration continued to take place: TDE adapter, Tangram adapter, 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–June 8, 2007: In this time frame, the team started integration testing activities. Activities from the previous iteration continued to take place: TDE adapter, Tangram adapter, PEPS adapter development, web service development, and deployment for Tangram.
- June 9, 2007–June 28, 2007: In this time frame, the team finalized the execution of integration testing.

6.4. 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 questionnaire. In such a matrix, we label the rows and columns with the team 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). Fig. 6(A) shows the COM during the initial development stage, when the architecture was designed. Fig. 6(B) shows the COM during the second development stage. Fig. 7(A) shows the COM during the third development stage; i.e., at the start of the integration testing phase. Fig. 7(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 are the chief architects, overseeing the project. The next blocks are the UML Modeling Team, Performance Model Generator Team, Tangram Team, and PEPS Team. The last block is the testing team. The COMs are ordered in the same way the corresponding modules are order in the architectural DSM. 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.

We show examples of two additional types of coordination matrices among team members used for modeling the importance and frequency of communications. The information is derived from the social network questionnaire using the following questions:

- Communication importance: How important was the communication in allowing you to get your work done?
- Communication frequency: During the last 2 weeks, how often have you communicated with each person concerning the Global Studio Project?

Figs. 8 and 9 show the importance level and frequency of each communication as indicated by the team members in May and June, respectively. The options available in the questionnaire for response to these questions, by the team member, and the associated numbers shown in Figs. 8 and 9 are listed below:

- Communication importance: How important was the communication in allowing you to get your work done?
  1. Not Important- blank
  2. Slightly Important- 1.0
  3. Moderately Important- 2.0
  4. Important- 3.0
  5. Very Important- 4.0

- Communication frequency: During the last 2 weeks, how often have you communicated with each person concerning the Global Studio Project?
  1. Never- blank
  2. Almost Never (once in 2 weeks)- 1.0
  3. Sometimes (at most once a week)- 2.0
  4. Often (more than once a week)- 3.0
  5. Very often (at least once a day)- 4.0

We can derive the following conclusions from Figs. 8 and 9. First, we notice that the importance and frequency of communication increases from the first 2-week questionnaire, Fig. 8(A) and (B), to the second 2-week questionnaire, Fig. 8(C) and (D), and decreases significantly as shown in Fig. 9. Therefore, the team members perception of frequency and importance of communication reaches its peak during architecture and development phases and decreases during integration testing phases. We attribute this behavior to the fact that integration testing is driven by the integration testing team and generally receives less overall project attention than architecture and development efforts.

Second, by comparing the matrices from Figs. 8 and 9 with the COMs matrices of Fig. 7, we found that the importance/frequency matrices shown in Fig. 9 are sparser than the coordination matrices shown in Fig. 7. As we reflected further on the data we found the following plausible explanations:
Developers and architects may not consider as Important the communication events related to integration testing phases as compared to the communication events related to their own work.

Some team members may not have checked answers for all questions.

The options Not Important and Never communicate are displayed as blank.

Developers and architects are not highly motivated to answer surveys. Therefore, as the project progresses, the accuracy of answers to surveys may vary.

Therefore, we conclude that automated approaches for social network analysis should be used to track the evolution of the communication patterns in long-running projects. We have found that using questionnaires to analyze communication in the social network was extremely insightful; however, we recommend that importance and frequency of communication matrices should be used to help validate the results obtained from the COM matrices.

6.5. 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 Fig. 6 and 7 with the architectural DSM shown in Fig. 4(B) as follows.

The light grey parts of the COMs model the communications through AA and PF. Since they do not belong to any particular team but are the chief architects, 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 upper part of each COM with the architectural DSM. Note that the architectural DSM in Fig. 4(B) is asymmetric: its upper-right part is blank, modeling that the components do not influence 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. 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 by non-architects.

7. Experimental results

In this section, we report our results by answering the research questions proposed in Section 5.

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 through the answers to the following three research questions.

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 leading role in the project; (2) with only one exception, all the off-team communications happened through architects (through the rows/columns labeled with bold font.)

RQ3: Do the developers of the same module only coordinate with each other, and need no communications 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 the dark background and white mark in Figs. 6 and 7. These exceptions are caused by the fact that the Tangram team has more experience with web services and has an experienced architect, FD. The Peps Team has less experience, and has to learn from the Tangram Team by talking to FD and GF. FD is an architect, but EN is physically located closer to GF than to FD (both GF and EN were located in Brazil).

This leads to an interesting observation. The key feature of the GSP V3.0 project process is that the architects were distributed from different remote sites and competence centers, each having different expertise and responsible for a different component. This is different from previous GSP V1.0–2.0 projects where all the architects were located at a central site. However, we notice that there is no architect (design rule decision-maker) in the Peps Team. As a result, the lead of this team (EN) had to communicate with the architect or members from another team. The observation verified our intuition that having a focal architect within each remote site and component team is essential to minimize the communications between teams. We 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 he/she needs to accomplish the task within the local team, and does not need to send inquiries to members of another team.

RQ4: How stable is the coordination structure over time?

RA4: The coordination structure turned out to be very stable in terms of inner- 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 architects, indicating that design rules became stable over time.

RQ5: Can the architectural DSMs reflect the importance level and frequency of the communication between teams?
Central site resources tend to be more trusted by upper management and remote sites in the following ways:

- Central site resources are usually assigned requirements and architectural (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.

Therefore, central site resources tend to become bottleneck resources in very large software development efforts, and generally cost more than remote site resources. In GSP V3.0, a “system of systems” process for global software development which employs domain experts located at remote sites was used. This process for global software development attempts to solve some of the coordination problems that arise when remote teams are composed of less experienced personnel 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 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 the remote sites.

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

8.3. Management

The factors for success from the project management point of view are usually team, communications, and culture. We now describe how these factors were addressed in GSP V3.0 to help ensure project success. The objective of our team building activities was to build a high-performing team with a feeling of partnership. The project was initiated by having a face-to-face kickoff meeting where the key team members were present. After 3 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 remote 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 1 month. The project used a shared document and code repository sharing information and weekly teleconferences to address issues. All project members were invited to attend the weekly teleconference and most teleconferences were well attended.

We believe the high level of commitment observed throughout the project was due to the strong personal relationships that were created by the team building activities. We observed an evolution of the communication patterns among team members depending on the phase the project was in. Initially most of the communication was among colocated 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 in the remote teams.

As the project gathered momentum, we saw a shift in the pattern of communications from site-based to work-related communications depending on need. Finally, as the project reached the critical delivery milestone we saw even greater structure in the communications based on need and less communications based on site. The project had several micro-cultures of a few similar cultures, so culture shock was not 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 communications.

8.4. Strategies

We now describe some of the strategies used in GSP V3.0 to overcome some of the obstacles that are usually found in global software development projects (Sangwan et al., 2007). The central site was located on the East Coast of the US and the remote sites were located in Brazil and Italy. The time difference between Brazil and the US...
varied from 1 to 3 h as the US and Brazil changed from Standard Time to Daylight Savings Time. The time difference between Italy and the US was 6 h. 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 resulting 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 to 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 enough to overcome the several company, country, and university cultures. The project manager traveled at the start of the project to establish personal connections to all team members. In addition, one of the remote site team members (RU) was assigned to work at the central site for a short time. One key member of the central site (FD) had been a member of one of the remote sites. We observed that both team members (RU) and (FD) had very key roles throughout the project.

8.5. Infrastructure support

Good collaboration tools were used (Gforge, SVN, Web-ex, wiki). SVN was used for source code control and Gforge was used as a document repository and issue tracking tool. We attempted to use Gforge as an e-mail repository for team communication; however, most team members preferred to use their e-mail tool of choice. The wiki was used to store team information, process guidelines, and meeting minutes. Teleconferences were short and had a very structured agenda. A sense of urgency was created by defining a short 4-month project with a well defined and achievable goal that was understood by all team members and reinforced at the weekly phone teleconferences. Therefore, the most important project-wide communications tools were 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.

The social network analysis questionnaire was an important tool that was used to track team member’s communications. This tool was custom-developed using php scripts and a back-end sql database. We have found that it was very difficult to motivate developers and architects to accurately and periodically respond to a web-based questionnaire. The project manager had to invest significant amounts of time to obtain the survey data. Therefore, we suggest that automated approaches for data collection of project communication be employed; e.g., e-mail based data mining. However, it was also very difficult to get team member to comply with the e-mail standards that would be required for automated e-mail analysis.

In our questionnaire design, we attempted to keep the survey short and we used only 10 questions. We have learned that even 10 questions was perceived by our team members as a very long questionnaire. We recommend that if social network analysis questionnaires are used, they should be limited in length to as little as three questions: who communicates with whom, importance of communication, and frequency of communication. In addition, the custom designed tool for data collection should flag entries that were not accurately completed to better support the data analysis of the survey.

9. Related work

In Herbsleb et al. (2005), results from experiments with GSP V1.0 were presented. The authors concluded that early face-to-face meetings between central site members and remote site members was a very important factor in establishing trust between central site and remote site team members. In Avritzer et al. (2006), an experience developing software using a global workforce was presented. The authors observed that the process and communication structure used in this specific project were correlated with significant delays incurred in addressing critical problems. In Avritzer et al. (2007), the global software development process analyzed in this paper was introduced.

Using DSMs as project management models has been widely applied in other engineering disciplines (Steward, 1981; Eppinger, 1991), but not in software engineering practice. One reason is that software designers are not used to model software architecture using DSM models. Instead, software designers are usually trained to use box-and-line style diagrams. Our experience shows the possibility of deriving DSMs from prevailing models, and provides preliminary data revealing the potential of using DSM models to predict coordination needs. Cataldo et al. (2006) empirically studied the possibility of identifying coordination requirements from modification requests. By contrast, this paper reveals the potential to identify coordination needs from software architecture itself.

Factors such as cost concerns, company mergers, and acquisitions have made multi-site software development more commonplace (Herbsleb, 2007). It is not uncommon for an open source project to have developers from multiple continents. However, global software development often suffers from a myriad of communication and coordination issues. In this section, we survey studies on socio-technical congruence and some of the problems caused by communications and coordination in global software development.

The importance of organizational communication patterns in software development has long been recognized. Conway in Conway (1968) suggested (in what became known as Conway’s Law) that the structure of a system mirrors the structure of the organization that designed it. Using Conway’s Law, several researchers (e.g., (Bowman and Holt, 1999, 1998)) have reverse engineered the architecture of software based on developer information. Bowman and Holt found this ownership architecture to be quite useful, including its identification of experts for components and discovery of non-functional dependencies.

MacCormack et al. (2008) found empirical evidence in support of Conway’s Law in their comparison of open source and commercial projects. For each commercial project they examined, they compared it against a similar open source project (e.g., Linux against XNU). They report that since developers in the commercial projects were collocated, they were able to build more tightly coupled software components. On the other hand, the open source projects that had less coupled components also had developers that are highly distributed. Further support for Conway’s Law, comes from investigation into one of the open source projects, Gnumeric, that had much higher coupling than any of the other open source projects. Upon further examination, they found that two developers had been contributing more than half of the work to Gnumeric. Hence, since Gnumeric was not as distributed as the other open source projects, its component coupling was significantly higher.

Although the significance of communication and organizational structure have been long recognized, the term socio-technical congruence is fairly new. First introduced by Cataldo et al. (2006), socio-technical congruence measures the alignment of task dependencies to coordination activities (e.g., developer communications). In comparing design and organizational structures, Cataldo et al. (2006) proposed the concept of coordination...
requirements. A coordination requirement matrix is a developer-by-developer matrix that represents the extent to which pairs of developers need to coordinate their work. They use logical dependencies, based on source code files that are changed together in development tasks, to identify communication requirements. Recent studies (e.g., Cataldo et al. 2006, 2008), though, show that when socio-technical congruence is high, development tasks are accomplished more quickly.

Although empirical studies show a strong relation between design structure and organizational structure, they generally only assess coordination mismatches retrospectively. Software project managers lack formal and effective means to reason about how development tasks can be partitioned and assigned to maximize parallelization of work based on dependency relations. We lack a theory and model to quantitatively predict coordination needs based on both design and organizational structures, in order to assist architects and managers in preventing mismatches from occurring. In this paper, we first reported the experience of using matrix models to uniformly make both design and organizational structure explicit, showing the potential to use these models to manipulate both structures.

10. Conclusions

In this paper, we have presented an experience report of using design structure matrixes to explicitly reveal the communication needs of a global software development project, as well as the possibility of deriving DSMs from prevailing box-and-line style architecture models, making it possible to assess communication implications at early development stages. We first conclude that the comparison between DSMs and COMs revealed the fact that GSP V3.0 employed an architecture that enabled an efficient communication structure. Second, 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 tasks, assigning each module to a remote team. Third, we conclude that automated approaches for social network analysis should be used to track the evolution of communication patterns in long-running projects. Importance and frequency of communication matrices should be used to help validate the results obtained from the COM matrices. We have also reported on our experience using a novel process for global software development, which employed domain experts located at remote sites to enable collaboration among remote teams. The approach consisted of the integration of large existing components to build a domain-specific tool for performance analysis of industrial systems. This “system of systems” process for global software development attempted to solve project management problems that arise in the coordination of global software development projects when remote teams are composed of less experienced staff than the central sites staff.

We observe 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 so overwhelmed by remote site requests. This additional scalability is achieved because our process enables the distribution of important tasks to the remote sites. Our results are still 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 project communication requirements in large-scale globally distributed projects.

References


Alberto Avritzer received a PhD in Computer Science from the University of California, Los Angeles, an MSc in Computer Science for the Federal University of Minas Gerais, Brazil, and the BSc in Computer Engineering from the Technion, Israel Institute of Technology. He is currently a Senior Member of the Technical Staff in the Software Engineering Department at Siemens Corporate Research, Princeton, New Jersey. Before moving to Siemens Corporate Research, he spent 13 years at AT&T Bell Laboratories, where he developed tools and techniques for performance testing and analysis. He spent the summer of 1987 at IBM Research, at Yorktown Heights. His research interests are in software engineering, particularly software testing, monitoring and rejuvenation of smoothly degrading systems, and metrics to assess software architecture, and he has published over 50 papers in journals and refereed conference proceedings in those areas. He is a member of ACM SIGSOFT, and IEEE.
Daniel J. Paulish is a Distinguished Member of Technical Staff at Siemens Corporate Research in Princeton, NJ, responsible for the Siemens Software Initiative in the Americas. He is a co-author of Software & Systems Requirements Engineering in Practice, Software Metrics: A Practitioner’s Guide to Improved Product Development, the author of Architecture-Centric Software Project Management: A Practical Guide, and a co-author of Global Software Development Handbook. He is formerly an industrial resident affiliate at the Software Engineering Institute (SEI), and he has done research on software measurement at Siemens Corporate Technology in Europe. He holds a Ph.D. in Electrical Engineering from the Polytechnic Institute of New York.

Yuanfang Cai is an assistant professor at Drexel University. She received her MS and PhD degrees in 2002 and 2006 respectively from the University of Virginia, advised by Kevin Sullivan. Her research interest is to develop design representations and automated, quantitative analysis techniques to reason about design structure and related outcomes early in the development process. Her research on the synergy between software design and organizational structure is funded by the National Science Foundation (CCF-0916891).

Kanwarpreet Sethi currently works as a technology consultant at Deloitte Consulting LLP. His research interests include measuring modularity and stability of software architecture and design, and automated modeling of design and architecture. He received his BS and MS in Computer Science from Drexel University in 2007 and 2009 respectively.