On Coordination Mechanisms in Global Software Development

Marcelo Cataldo[1] Matthew Bass[1] James D. Herbsleb[1] Len Bass[2]
[1] *Institute for Software Research*
[2] *Software Engineering Institute*
Carnegie Mellon University
Pittsburgh, PA 15213
mcataldo@cs.cmu.edu mbass@cs.cmu.edu jdh@cs.cmu.edu ljb@sei.cmu.edu

Abstract

The ability of an organization to successfully carry out its tasks depends on the appropriate combination of organizational structure, processes, and communication and coordination mechanisms. In this paper, we present four case studies that exemplify coordination breakdown problems in global software development. Our analysis showed those problems took place even in the presence of a collection of processes, organizational mechanisms and communication tools established to increases the ability of the teams to perform their tasks. Finally, we discuss possible solutions to overcome the identified problems.

1. Introduction

In the system design literature, it has long been speculated that the structure of a product inevitably resembles the structure of the organization that designs it [5]. Conway [5] reasoned that coordinating product design decisions requires communication among the engineers making those decisions. In order to minimize the communication overhead, products must be split into components with limited technical dependencies among them, and each component assigned to a single team. Conway [5] proposed that the component structure and organizational structure stand in a homomorphic relation, in that more than one component can be assigned to a team, but a component must be assigned to a single team.

A similar argument has been proposed in the strategic management literature. Baldwin and Clark [1, page 90] argued that modularization makes complexity manageable, enables parallel work and tolerates uncertainty. The design decisions are hidden within the modules which communicate through standard interfaces. Modularization adds value by allowing independent experimentation of modules and substitution [1]. Moreover, Baldwin and Clark [1, page 89] argued that a modular design structure leads to an equivalent modular task structure. In the context of software engineering, that approach was first articulated by Parnas [20] as modular software design. Parnas [20] argued that modules ought to be considered work items instead of just a collection of subprograms. Development work can continue independently and in parallel in the different modules. Parnas' [20] views coincide with theoretical arguments from product design [5] and strategy literatures [1].

The theoretical arguments proposed by Parnas [20], Conway [5] and Baldwin and Clark [1] rely on two interrelated assumptions. The authors assume a simple and obvious relationship between product modularization and task modularization. Hence, by reducing the technical interdependencies among modules, the modularization theories argue, task interdependencies are reduced. This, in turn, reduces the need for communication among work groups. The modularization approach is a very useful tool for dividing the development of a complex software system into manageable units. However, some technical dependencies will remain, creating task dependencies that could be difficult to identify and manage. Development teams that coordinate their tasks successfully exhibit better performance [4, 14, 15]. Moreover, geographically distributed development teams are at a disadvantage because of the negative impact of distance on the engineers' ability to communicate and coordinate their work [13]. Hence, we need to complement modularization with appropriate mechanisms to identify relevant work dependencies and, consequently, maintain suitable levels of communication and coordination among teams developing interdependent modules.

One approach is to examine the relationship between organizational structure, processes and coordination mechanisms. In fact, the organizational theory literature suggests that the ability of an organization to successfully carry out its tasks depends on the appropriate combination of organizational structure, processes, and communication and coordination mechanisms [3]. Designing successful global software development organizations requires achieving a better understanding of the intricate relationship among those three elements. Moreover, in the specific context of geographically distributed work, communication tools play a fundamental role in information exchange, coordination, and communication in general. Research has shown that numerous factors, such as task characteristics, urgency of the communication and the social environment, affect the selection of particular communication means [6, 16]. Therefore an additional important question is how and for what developers use the host of tools at their disposal

and what set of processes can be used to ensure that communication tools provide the maximum benefit to the developers and the organization as a whole.

The Global Studio Project [17], sponsored by Siemens Corporate Research, provided a rare opportunity to examine the questions raised in the previous paragraphs. In this paper, we first present four case studies of problems in software development projects. We examined the sources of those problems in the context of a global software development effort. Our analysis showed that although numerous coordination mechanisms were in place to increase the ability of the teams to successfully perform development tasks, coordination breakdowns still occurred. In the second part of the paper, we focus on the role of communication between remote teams and the patterns of usage of various communication means to better understand how information is disseminated in the development organization and how existing processes affect the usage of those tools. Our analysis suggests that in global software development organizations there is a need for an integrative set of processes and mechanisms that promote communication and ensure that appropriate technical information is stored and disseminated accordingly. Finally, we discuss potential solutions and future research paths that would be beneficial in terms of design and management of global software development organizations.

2. The Design of a GSD Organization

The Global Studio Project (GSP) [17] was established by Siemens Corporate Research (SCR) as a test bed to gain better understanding of the issues associated with global software development. The project simulated a real world GSD project by using student teams. The students participated in the GSP as part of their academic curriculum and they operated in universities in Ireland, Brazil, Germany, India and the United States. These students were pursuing graduate degrees in software engineering or related fields. The remote members had no previous experience of working together nor did they meet in person during the project.

The system developed in the GSP, called MSLite system, was a unified management station for building automation systems such as heating ventilation and air conditioning (HVAC), access control, and lighting that will allow a facility manager to operate such systems. The development effort was divided in 6 iterations of 8 weeks each. Remote teams had deliverables that included specifications, functioning code and unit tests at the end of each iteration and in some cases at the half point through the iteration.

A significant amount of effort was devoted to all aspects of the design of the infrastructure, the design of the organization and planning of the project. Best practices from the software engineering literature were carefully considered. The following paragraphs describe the best practices and processes used in GSD and the rational behind those decisions.

The GSP was organized in a two-level hierarchical structure. A central team located at SCR was responsible for requirement specification, overall architecture, project management, integration testing, process definition and management and infrastructure. The remote teams were responsible for detailed design, specification of unit tests, coding, unit testing, and maintaining related artifacts. The central team had a Supplier Manager (SM) for each remote team, who managed the central team interactions with the remote team's corresponding local Supplier Manager. This hierarchical structure has the benefit of centralizing critical decisions and establishing clear paths of communication [22].

The central team used the architectural documentation to identify dependencies among components early in the development process and represented these dependencies as a design structure matrix (DSM). Following an analysis similar to Sullivan et al's [24], the DSM was used to identify the set of tasks to be assigned to each remote team that would minimize the dependencies and consequently, minimize the need for coordination between remote teams.

The central team established processes for communications and meetings, design and development, configuration and change management and integration of the code. The central team also provided access to numerous project planning documents such as work item assignments, per-iteration basis project plan and plan calendar. Finally, the central team emphasized the development and usage of documentation at the architectural and the design-level. All these processes and decision followed recommendations from traditional software engineering literature [2, 21].

Following best practices from the open source community [12], the GSP provided a wiki web portal which gave the central and remote team members access to all the processes and documents described in the previous paragraphs as well as information about the teams involved in the project. The web portal also provided access to several tools such as version control system, discussion forums, defect tracking system and a daily build system.

Modifications to the software code were managed by a central version control system. All teams were encouraged to (a) submit their changes regularly to the central repository and to (b) maintain the teams' local copy of the source code tree synchronized with the central repository. These processes increase the awareness of the current state of the system and the changes made to the source code tree as suggested by the studies on open source projects [11, 12].

As in successful open source projects [12], a daily build system ensured that the code compiled correctly and that unit tests ran successfully. If a problem was encountered, the build system would send email to the central team and to the team that made the last submission to the version control system, who became responsible for resolving the problem. If the issue persisted overnight, the central team would revert the changes the following morning.

Finally, the developers in the GSP had at their disposal several communication mediums. Weekly conference calls between the central team and each remote team were primarily for gathering status of the tasks in progress and define the next set of tasks. Email was the recommended means of communication between central and remote teams. Discussion forums were the suggested environment for exchanging technical information about the project.

Table 1: Coordination Mechanisms Used in the GSP

Mechanism	Purpose
Centralized Structure	Centralize critical decisions and establish clear paths of communication.
Early Identification of Dependencies	Reduce dependencies amongst tasks assigned to remote teams.
Documentation	Reduce the need to communicate amongst remote teams by having access to detailed design decisions.
Change, Configuration and Integration Management Processes	Identify relationships, manage, control, audit and report on the changes made to the software.
Periodic Commits	Increase awareness by making ongoing changes to the system available to all the remote teams.
Daily Builds	Reduce the potential for integration problems by identifying them early.
Communication Tools	Allow for exchange of information amongst teams when other coordination mechanisms are not sufficient.
Periodic Meetings	Status and definition of tasks. Relay information from remote teams to others.

In sum, the software engineering literature would suggest that the best practices chosen (Table 1) in GSP represent a collection of coordination mechanisms aimed at laying the ground for a successful project. In the following section, we discuss several case studies where those mechanisms failed and coordination breakdown occurred.

3. Examples of Coordination Breakdowns

In this section, we present four case studies that exemplify situations in software development projects that tend to have larger implications in global software development. The initial data collection was done using an approach similar to the critical incidents technique [8]. We met with the members of the central team and we asked them to identify events during the life of the project that were representative of important problems in coordination. The central teams members were also asked to provide background information regarding the events they recalled. We then compiled data associated with the incidents from numerous sources such as technical documentation, version control system, project plan, meetings minutes, weekly status reports of the developers, discussion forum, defects database, email archive and social network survey. The following sections describe these events in detail.

3.1 Event I: Change in a design specification.

Problem: The team in Ireland was responsible for task A in iteration 2. The team in India was responsible for task B in iteration 3. Task A consisted of designing several object classes and specifying the properties and methods of those classes. Task B implemented a property editor that used the object classes defined in Task A. The developers involved in both tasks participated in three different discussion forums focused on the technical details on the implementation for task A. All the technical details of task A were not captured correctly in the design specification document. This mistake led to a serious mismatch between the contents of the documentation and the actual implementation, a perennial problem in software development. The Indian team worked on task B guided primarily by outdated design specification which led to integration problems when they submitted their changes into the version control system, delaying significantly the completion of task B.

Analysis: Our analysis suggests that some members of the Indian team were reluctant to make full use to the version control system. The time zone difference between India and the US EST (the location of the central team), meant that around the time the Indian team was leaving for the night, the central team would be starting their workday. The daily builds were done first thing in the morning of the central team time zone. Hence, if any submission to the code repository resulted in a broken build, the Indian team would not have the opportunity to fix the problem before the central team would revert the changes. Hence, the Indian team tended to rely a lot more on documentation and also tended to make less frequent but much larger submissions to the repository.

This incident highlights several interesting issues. First, despite the availability of numerous communication tools (e.g. email, discussion forums, and defect reports), our analysis indicated that very little lateral communication between the remote teams took place. Exchange of information between the remote teams was limited to discussion forums early in the life of task A. Secondly, this case suggests that in this project there was some reliance on documentation as a coordination mechanism and the important role of documentation might have not been clearly communicated to all the remote teams. In addition, the reliance on documentation might also stem from the problems with the daily build processes as discussed next. Finally, the case exemplifies the role that processes play in shaping the be-

havior of certain developers and, ultimately, the coordination of activities. Although nightly builds are considered an effective practice, the consequences that resulted from breaking the build diminished the value of this process.

Take-away Points: There were two coordination mechanisms, documentation and daily builds, in place that should have eliminated the occurrence of this type of problem. If documentation is to take such an important role in coordinating amongst remote teams, there is a need to ensure that the divergence between the source code and the documentation is minimal or ideally non-existent. One approach could be to review the documentation as part of the code review done when a work item was finished or at the end of the iteration.

The data suggested a reason why some teams did not follow the processes limiting the effectiveness of the daily build process could have been the wide range of time zones separating the remote teams. Then, it is important to understand the implication of the processes in the local context of each remote team as processed are designed and defined.

3.2 Event II: Modification of a major interface.

Problem: A team in a US university was responsible for implementing a data access interface that all other components of the system depended on. One of the requirements of the data access module was to uniquely identify instantiated objects. The developer responsible for the task evaluated the original design specification done by the central team and determined that the interface needed to be modified in order to satisfy the requirement to generate unique object identifiers. As required by the design and development processes, the developer sent a proposal for the design change to the central team. However, that took place two days before the deliverables were due. Since there was no reply from the central team, the developer submitted the modifications proposed in his design to the version control system two days later. These actions resulted in some major modifications in various parts of the system causing delays and frustration on the other remote teams. The following email trace shows all the information exchanged between the developer team (named Team X) and the central team regarding this issue:

To: ALL TEAMS
Subject: Access Control Modifications
All,
As you may already have noticed the Access Control component underwent some changes to incorporate … In order to integrate the changes, some teams' code may have required slight modifications which were carried out by the team X. These changes were authorized by the central team and were essential for successful server-side integration. Please review your code to ensure all changes were satisfactory.
Thanks and best regards,
Central Team SM

-----Original Message-----
From: Team X
To: Central team SM
Subject: AccessControlResultSet design
All,
Attached is the detailed design of the AccessControlResultSet and the updated detailed design of the AccessControl component.
Please review and have comments to me before Mondays teleconference meeting. best regards, Developer
-----Original Message-----
From: Central Team SM
To: Team X
Subject: FSS .NET Remoting Failing test
Thanks. Can you update me on your progress?

Analysis: This incident is a good example of several related problems. First, we have a change to an interface, a syntactic dependency, that becomes a major problem because the interface is used many times in all the components of the system. Furthermore, the semantic of the functionality also changed (generate and return a unique identifier), augmenting the scope of the change. Syntactic dependencies tend to be misleading because they are typically considered simple issues. This example shows that certain types of dependencies (e.g. numerous modules depend on the same interface that returns a critical data type) require a lot more attention than others, particularly during the design phase. Early identification of such types of dependencies can also help focus the efforts of management in the most critical aspects of the project. Another interesting issue highlighted by this incident is the impact of lack of contextual information and conflict. After the central team announced that major changes would take place in the code, several teams expressed frustration against team X because the changes to be made would delay their current work. However, none of the remote teams had a complete understanding of why the changes were necessary.

Take-away Points: A growing body of work suggests that the identification of dependencies is a challenge in software development organizations, particularly in those that are geographically distributed [7, 10, 13] and this case provides an example of the implications of such a problem. There are proactive steps that managers or other stakeholder could followed. For instance, one approach would be to identify interfaces that would affect teams at different sites and dedicate more effort to validate those interfaces. However, there is still a need for developing mechanisms to aid the process of identification of dependencies, particularly in early stages of the project.

3.3 Event III: Circular dependency between components.

Problem: One US university team was responsible for task A due at the midpoint of iteration 3. Another team from a different US university was responsible for task B due at the end of iteration 3 of the project. The information

in discussion forums and email indicated that the teams had extensive exchange of technical information associated with interfaces developed as part of task A and that the component developed as part of task B would depend on. These interfaces represented a case of syntactic dependencies between two components (A → B). Upon completion of task A, the central team and the development team did a detailed code review and the modifications were approved.

Analysis: Unfortunately, the architectural documentation also revealed a more complex semantic dependency through a publisher/subscriber mechanism that would need to be resolved as part of task B (B → A). This dependency went undetected during the code review and this oversight resulted in major modifications to the component developed in task A during the execution of task B. Code reviews are well known practices in software engineering. Although code reviews are commonplace, they do not typically involve an analysis of dependencies to determine appropriate set of developers to participate in the review [26]. This incident shows how important it is to identify all the dependencies among components in order to have the code reviewed by the relevant developers. Distance and the central management approach tend to augment the impact of this type of mistake because impromptu and lateral communication paths are limited.

Take-away Points: Traditional software engineering practices might need adjustments in order to successfully apply them in the context of global software development. For instance, decentralizing code reviews by allowing member of other remote teams to participate could potentially help in the identification of problems with the existing implementation as well as dependencies with other modules that are responsibility of other remote teams. Remote team members bring diverse sources of knowledge about the system and diversity in knowledge tends to improve the performance of group-level tasks [27].

3.4 Event IV: Significant delays in the implementation of a complex module.

Problem: The development of a low priority but complex component was originally scheduled for iteration 3 to be done by the Irish team. Modules developed in iteration 5 were dependent on this component. The Irish team was not able to finish the design of the component so the design and development was re-scheduled for iteration 4 to be done by the group in Brazil. Shortly after the Brazilian team did the preliminary analysis and estimated the effort to complete the component, the Supplier Manager for the Brazilian team got sick and communications with the central team dropped almost to a halt. Ultimately, the task had to be rescheduled again for iteration 5 and this time one of the US universities teams would be responsible.

Analysis: The delay in the implementation of this particular component changed the dynamics of the coordination required between the teams. A fairly loose coupling (Team A implements the module, then during a subsequent iteration, Team B uses it) became a very tight coupling (implementation and use are concurrent). The development teams involved are located one in the US and one in Germany. The need for fluid exchange of information and tight coordination are critical as the design decisions made by one team could have implications in the other team's development efforts. Our qualitative analysis revealed that little lateral communication took place between the remote teams. Moreover, numerous modifications to the code had to be reverted because the changes broke the build. This situation escalated to a clear point of frustration as the following message in a modification to the code indicates:

> r1901 | Central team SM
> Changed paths:
> M /MSLite/MSLite.Rules/ConditionEvaluation/Evaluator.cs
> ….
> D /MSLite/MSLite.Tests.Rules/RuleEngineTests.cs
> Reverting repository revisions 1898 and 1897 by "Developer A" in attempt to successfully build integration server – AGAIN

Take-away Points: As changes in the schedule occur, it is important to adapt the communication and coordination patterns to the new needs. The transformation from a loosely-coupled relation between the tasks into a tightly-coupled situation increased the interdependency between the two remote teams. Hence, coordination mechanisms such as lateral communication ought to be promoted to handle the higher levels of interdependency [9].

3.5 Conclusion

The collection of events presented in the previous paragraphs suggests that the coordination problems encountered in global software development project depend on an intricate relationship of several factors. First, elements that influence how closely-coupled the work is, such as complexity and uncertainty of the interfaces, as well as whether the work is carried out sequentially or concurrently. Secondly, factors that influence the ability to communicate and coordinate, such as geographic separation, whether communication is direct or through an intermediary, and the quality of documentation play a important role. Finally, organizational factors such as processes, structure and goal alignment are significant mediators as well.

4. Understanding Interaction Patterns in GSD

In the case studies presented in section 3, one issue repeatedly came up in the analyses, the need to communicate amongst remote teams. Communication and coordination amongst workgroups that develop pieces of a system that are interdependent is crucial for a successful outcome of the development effort [4, 13, 14]. In the organizational

literature, particularly in organizational design, the idea of division of labor into interdependent units is a well developed idea and mechanisms for coping with the varying degree of interdependency have been proposed. For instance, Thompson [25] argued that highly interdependent tasks can be coordinated by mutual adjustment which "involves the process of transmission of new information during the process of actions" [25, page 56]. Galbraith [9] argued that as the level of interdependency increases additional mechanisms are required such as slack resources and lateral communication. In sum, the need to communicate amongst remote teams should not be neglected, in fact, it should be promoted whenever is appropriate.

As we indicated in section 2, in the GSP, developers and members of the central team had several means of communication to share technical information, discuss design issues, discuss technical problems, and coordinate their efforts such as a discussion forum tool, email, and instant messages. The central team defined guidelines regarding the usage of the various communication means. Those guidelines were available on the wiki web portal and they are summarized in table 2.

Table 2: Communication Means and Rules of Usage

Communication Means	Recommended Rules of Usage
Weekly Conference Calls	"… are either proposed by the central team or the remote team(s) to talk about the status of the project and clarify questions, or they take place at dates specified in the project plan, usually to discuss deliverables …"
Discussion Forum	"… All discussions on any aspect of the GSP or MSLite system must take place on the dedicated discussion forum …"
Email	"… 1) all communications with the Supply Manager … 2) all communications over email within the remote teams … 3) all cross-team communications will use a specific mailing list …"
Instant Messenger (IM)	"…can be used as a last resort at critical stages of the MSLite project … All IM conversations are to be logged …"

As the case studies suggested, the availability of the communication tools and the usage guidelines did not directly translate into remote teams using those tools to communicate amongst themselves in all the instances that it was required. Motivated by those findings and previous research that has shown that multiple factors affect individuals' choice of using particular communication means [6, 16], we examined how and for what purposes developers used the various communication mediums at their disposal. In order to perform this analysis, we developed a codification scheme to categorize the content of the interactions or discussions across the various communications means. In the following sections we describe the framework in detail and we present the results of our analysis.

4.1 Framework for Content Analysis

4.1.1 Information sharing and acquisition: This category refers to interactions that intend to provide or seek technical information related to the system design and implementation and development environment and infrastructure. We separate the interactions in three groups and we provide a representative example for each group:

- **Information seek**: "…How should I write sanity tests for methods? …"
- **Information post**: "…The CreateAlarm receives now one IToken and one IAlarmRule as arguments …"
- **Information seek-reply** refers to information seek interactions that received one or more replies: "… Developer A: How do we check our code for remote access? …. Developer B: how to do this ... Copy the instance variables, first and last fixtures …"

4.1.2 Task-related interaction: This category refers to an interaction that intends to define aspects of a particular development task or seek actions or decisions on a particular task. We separate the interactions in four groups:

- **Task negotiate** refers to an exchange of information and options to collaboratively define a particular task: "… Developer A: As I understand it the UL iteration 3 deliverables do not include the LnRRuleEngine or the AlarmRuleEngine as these are iteration 4 deliverables … My question is this, does IIITB need this for the current iteration presentation system rule edits or are they using a mock-up? … CT member: The interfaces listed on <artifact> will be implemented by your team in this iteration. … Developer A: That makes sense to me …"
- **Task seek decision** refers to a request to clarify and/or confirm the definition of a task to perform**:** "…The acceptance tests for our current component (rules object model) are not that thorough. Is it desired to have coverage beyond the specified by the acceptance tests? …"
- **Task notify decision** refers to a reporting action that a particular task definition has taken place**:** "…Regarding the scope of your component you just need to develop the rules processor for L&R and Alarm rules. The alarm processor deals with alarm objects … and is not part of this iteration …"
- **Task seek action** refers to a request to perform a particular task**:** "… The function GetAlarmRules(token) in MSLite.Interfaces.Data.DataAccess does a validateAcess … hope u do understand the requirement and reflect the changes as soon as possible …"

- **Task notify status** refers to a report of progress in a particular development task: "… Team A has finished the implementation of the Rule Editor …"

4.1.3 Design-related interaction: In this category, we group interactions where the content relates to design decisions about the system. We defined three groups:

- **Design negotiate** refers to an exchange of information and options to collaboratively make a particular design decision: "… Developer A: This means that ALL value updates after this initial setup are done through the Pub-Sub. Therefore, this method should NOT enforce property access rights since it is not called by the user… Developer B: Adding a method to allow the VirtualFSS to call at init time is probably the correct way to go … Developer A: In fact there's no need at all to have the new method I proposed. Only a rename of the actual method is fine … Developer B: Ok, if you feel that everything is fine, go ahead and commit …"
- **Design seek decision** refers to requests to clarify and/or accept a design decision**:** "…We think this is good approach and we like to know the central team opinion …"
- **Design notify decision** refers to a message that informs that a particular design decision has been made: "…We pretty much knocked the idea of one manager for two caches anyway. So we'll have a new class diagram on its way to you soon …"

4.1.4 Defect resolution interaction: This category focuses on interactions where the content refers to exchanges of information and discussion about problems encountered in the product and how to resolve them. We grouped the interactions in two different sub-categories:

- **Problem-solution exchange** refers to a discussion where options of the potential sources of the problem are presented but no resolution or outcome is reached: "… Developer A: Instead of using Client Code We could use a configuration file and the code looks like this … Developer B: I did the experiment to show that Config file does not work for more than 1 client instance …"
- **Problem-solution negotiate** is an exchange of information and options that leads to a resolution of the reported problem: "… Developer A: the interface IRuleId does not contain any attribute as of now … Developer B: I was not aware of this. I think this interface should be changed to derive from the IObjectID interface … CT member: I think it should be a matter of adding the required attribute to the IRuleID interface … Developer B: A solution has been committed today…"

4.2 Results of Content Analysis

In order to understand how and for what purpose remote developers used the various communication means, we examined the content of discussion forums, meeting minutes and emails. The data indicated that the usage of instant messaging (IM) was very limited. There were 7 IM discussions over a period of 2 days and they only involved 3 developers. These 7 interactions had logs. We also identify 9 additional IM interactions where the content of the discussion was not logged. These 9 IM interactions involved 2 developers from different teams and they corresponded to a single work item from iteration 2 of the project. We think that the IM interactions are not representative of patterns that could be generalized to the whole project. Hence, we did not consider them in our analysis.

We focused our analysis in a period of 5 months from December 2005 until early May 2006 for several reasons. Three out of the six development iterations took place during those months, representing a significant part of the development effort. During the period of time we studied, all six remote teams were participants in the project. Prior to December only three teams were part of the project for an average of 1.5 months and only one development iteration took place. After May, only two teams remained until the end of the project working on the last two iterations.

Phone-based Meetings: The central team had phone-based meetings with each remote team almost on a weekly basis. The central team defined the main purpose of those meeting as to get status on the progress of the development tasks assigned for a particular week and determine the action items for the next week. There were 112 phone-based meetings between December 2005 and May 2006. Each remote team met with the central team an average of 19 times (min: 15 times and max: 22 times). Using the meetings minutes, we examined the activities carried out in those meeting to understand the role they play in the development process. Table 3 summarizes the results of the codification process.

As it was originally intended by the central team, the phone-based meetings were primarily used for gathering status of the tasks and defining the set of tasks to perform on a weekly basis. The data also showed that the central team used these meetings to relay status information from remote teams. Table 3 indicates phone-based meetings were also used to seek and share information (28% of the meetings) between the remote and the central teams as well as to discussed design decisions (17% of the meetings).

Email Interactions: In the December-May period of time, there were 5486 emails that corresponded to 2443 interaction threads. We eliminated 481 email threads because they were not related to the development activities of the project. The discarded emails contained information such as welcoming messages and action items related to the research activities of the project (e.g. request to fill up surveys and interactions of central team members with researchers). Then, we examined 1962 email threads. As table 3 shows the majority of the emails iterations were related to information post and information seeking and sharing activities. It is important to highlight that the information post pattern was so pervasive because 678 out of the 709 emails threads were status emails issued automatically by the defect tracking system.

Table 3: Patterns of Usage of Communication Mediums

	Teleconference		Email		Discussion forum	
	Occurrence		Occurrence		Occurrence	
Topic Category	#	%	#	%	#	%
Information seek	0	0	0	0	7	7
Information post	1	1	709	39.6	22	21
Information seek-reply	31	28	299	16.7	41	39
Task negotiate	112	100	146	8.2	8	8
Task seek action	10	9	209	11.7	6	6
Task seek decision	0	0	0	0	3	3
Task notify status	112	100	278	15.5	0	0
Task notify decision	0	0	14	0.8	1	1
Problem-solution exchange	5	4	137	7.7	11	10
Problem-solution negotiate	5	4	7	0.4	2	2
Design negotiate	19	17	38	2.1	10	9
Design notify decision	3	3	30	1.7	6	6
Design seek decision	1	1	0	0	9	8

The central team recommended that all technical issues related to the GSP project were brought up in the discussion forum (see table 2), which would allow the information to be available to all team members. However, the data suggests developers relied also heavily on email for information acquisition (299 email threads), limiting the possibility of that information to be available to all the teams. The results from table 3 also show that 633 emails threads (35.4%) were for task-related activities suggesting emails were an integral part of maintaining status on the tasks in progress and to request specific actions of tasks (e.g. "… work on defect 45 …"). Moreover, only 7.7% of the email interactions involved problem-solving activities. The central team established a mailing list specifically for inter-remote team communication. The data showed that only 32 (1.6%) email threads were cross-remote-teams interactions.

Discussion Forum: In the December-May time frame, we identified 113 topics created in the discussion forum tool. We discarded 7 topics because their contents were not related to development activities (e.g. welcoming messages or notifications maintenance activities in the tool). Then, we considered 106 topics in our analysis. Table 3 summarizes the results of applying our content analysis scheme to the discussions.

More than one-third of the topics created in the discussion forum were related to information seeking activity and developers responding to those requests. Topics that were just disseminating information accounted for 21% of the total number of topics created between December and May. Design-related interactions, seeking a confirmation on design decisions or negotiating those design decision represented 17% of the topics. Finally, our analysis show that developers tended not to use discussion forums for "problem-solution negotiate" type of interactions. That is, exchanges information about solutions to problems found in the system that reached a resolution were not common in the discussion forums.

The data also showed that overall there were only 27 topics (25.5%) that involved more than one remote team. The central team was involved in the discussions on 60 out of 84 topics of non-information-post type and of the other 24 topics only 7 involved more than one remote team. These results suggest that lateral communication between remote teams was limited in this communication means.

We also classified the contents of the discussion forums based on the responsibilities that the central team and remote teams had as described in section 2 (see table 4). We found that about half of the discussions were related to design issues and 27% of the discussions had architecture-related content. Moreover, discussions that contained architectural aspects of the system tended to evolve from an initial discussion regarding detailed design. Architectural or detailed design activities usually require high-bandwidth means of communication because of the extensive usage of graphical representations to convey key pieces of information and the need to deal with conflict arising from competing solutions [18]. In distributed development teams, the requirements for rich communication could be addressed by collocating architects and engineers in the early design phases of the project. However, our findings show a significant part of the detailed design and architectural definition exchanges took place well into the development stage. Then, our findings raise questions regarding the adequacy of the communication tools used in the project given the information exchange requirements of design activities.

4.3 Conclusion

The availability of several communication means and the guidelines suggested by the central team intended to promote information and knowledge sharing among teams and, through the usage of a discussion forum tool, provide a globally accessible repository of additional technical information for the project. Our analysis suggests the guidelines were not particularly followed and there was evidence of limited communication amongst the remote teams. These findings argue in favor of an integrative set of processes and mechanisms that to promote communication, contributions and ensure that appropriate technical information is stored and disseminated accordingly.

Table 4: Responsibility-related Discussions

Task	Occurrence	
	#	%
Requirement Specification	6	5
Architecture	30	27
Design	58	51
Coding	14	12
Maintenance	6	5
Specification of Tests	9	8
Unit testing	4	4
Integration testing	1	1
Project Management	6	5
Process	14	12
Infrastructure	11	10

5. General Discussion

In this paper, we have presented a qualitative analysis of events that are common to software development projects but we see their impact is much higher in geographically distributed software development. Moreover, providing a set of communications tools to the developers might not be sufficient to achieve the necessary coordination and exchange of information that interdependent development tasks might require. Our analysis showed that in several instances processes were not followed. Although, a traditional argument in software engineering would say "you must follow processes", we face the crude reality that in real organizations processes might be partially followed or not followed at all. Then, the important question as researchers is what we can do to overcome those limitations. In the following paragraphs, we discuss managerial prescriptions as well as issues to consider in future research.

***Promote lateral communication*:** the organizational theory literature suggests that lateral communication is the appropriate coordination mechanism as level of interdependency increases [9]. In our analysis, we found cases where lateral communication could have been beneficial even in cases where low levels of interdependency existed between remote teams. Then, technical leads and managers ought to promote lateral communication. Site visits and face-to-face meetings have been found to be successful mechanisms for developing an enduring relationship that facilitate exchange of information and knowledge and collaboration [19]. In other words, the goal should be to promote the development of an informal network of ties that can overcome the constraints that formal communication paths established by the organizational structure might impose on the development organization.

Identification of dependencies: Early could increase the likelihood of success of a global software development project by allowing managers and other stakeholders to design the appropriate organizational structure to carry out the tasks and identify the necessary set of mechanisms to facilitate coordination and flow of information. Unfortunately, identifying dependencies in an early stage of the project is not a trivial process. Currently, it is a manual process that requires deep domain knowledge. Identification of dependencies is further complicated by their dynamic nature. The coordination needs amongst the developers change over time as modifications are made to the software system [4]. Therefore, global software development organizations would benefit from mechanisms that allow the identification of the changes in dependencies such that manager and developers are notified of those changes and can react accordingly.

***The documentation-source code gap*:** Agile methods argue that documentation should not play an important role in the development process because it is always outdated [23]. However, documentation has the potential to be a useful coordination mechanism in geographically distributed software development organizations. The main challenge, then, is devising mechanisms that could ensure that the gap between the contents of the documentation and the actual source code implementation is closed. Some steps towards that goal could be implemented as part of code reviews where verification of the documentation is an integral part of the code review process.

***Architectural/high-level design activities*:** Change is inevitable in a software development project, consequently, architectural definition or high-level design activities could take place in various points in time in the lifecycle of the project. Our analysis showed a significant amount of the communication in discussion forums had to do with design and architecture. Those findings highlight the need for communication tools that provide higher levels of richness in the information exchanges. More importantly, our findings suggest the need for mechanisms to identify communications that involve architectural definitions or high-level redesign and facilitate organizational awareness of such situations in order to appropriately address the implications and impact of such changes.

6. Acknowledgements

The authors gratefully acknowledge support by NSF grant IIS-0534656, the Software Industry Center and its

sponsors, particularly the Alfred P. Sloan Foundation, the Software Engineering Institute, the US Department of Defense, NSF IGERT program 9972762 and the CASOS center at Carnegie Mellon University.

7. References

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

[2] L. Bass, P. Clements and R. Kazman. *Software Architecture in Practice, 2nd Edition*. Addison Wesley Publishing, 2000.

[3] R.M. Burton and B. Obel. *Strategic Organizational Diagnosis and Design*. Kluwer Academic Publishers, 1998.

[4] M. Cataldo, P. Wagstrom, J.D. Herbsleb and K.M. Carley. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. In *Proceedings of the Conference on Computer Supported Cooperative Work* (CSCW'06), Banff, Alberta, November 2006.

[5] M.E. Conway. How do committees invent? *Datamation*, 14, 5 (1968), 28-31.

[6] R.L. Daft and L.K. Trevino. The Relationship Among Message Equivocality, Media Selection, and Manager Performance. *Research in Organizational Behavior*, 6, 1987.

[7] C.R.B. De Souza. *On the Relationship between Software Dependencies and Coordination: Field Studies and Tool Support*. Ph.D. dissertation, School of Information and Computer Sciences, University of California, Irvine, 2005.

[8] J.C. Flanagan. The Critical Incident Technique. *Psychological Bulletin*, 51, 4 (1954).

[9] J.R. Galbraith. *Designing Complex Organizations*. Addison-Wesley Publishing, 1973.

[10] R.E. Grinter, J.D. Herbsleb and D.E. Perry. The Geography of Coordination Dealing with Distance in R&D Work. In *Proceedings of the Conference on Supporting Group Work (GROUP'99)*, Phoenix, 1999.

[11] C. Gutwin, R. Penner and K. Schneider. Group Awareness in Distributed Software Development. In *Proceedings of the Conference in Computer Supported Collaborative Work (CSCW '04)*, Chicago, 2004.

[12] T. Halloran and W. Scherlis. High quality and open source practices. In *Proceedings of the 2nd Workshop on Open Source Software Engineering*, Orlando, 2002.

[13] J.D. Herbsleb and A. Mockus. An Empirical Study of Speed and Communication in Globally Distributed Software Development. *IEEE Transactions on Software Engineering,* 29, 6 (2003), 481-494.

[14] R.E. Kraut and L.A. Streeter. Coordination in Software Development. *Communications ACM*, 38, 3 (1995), 69-81.

[15] T.W. Malone and K. Crowston. The interdisciplinary study of coordination. *Computer Surveys*, 26, 1 (1994), 87-119

[16] M.L. Markus. Towards a Critical Mass Theory of Interactive Media Universal Access, Interdependence and Diffusion. *Communication Research*, 14, 5 (1987).

[17] N. Mullick et al. Siemens Global Studio Project: Experiences Adopting a GSD Infrastructure. In *Proceedings of the International Conference on Global Software Engineering*, Florianopolis, Brazil, 2006.

[18] D.E. Perry, N.A. Staudenmayer, and L.G. Votta. People, Organizations, and Process Improvement. *IEEE Software*, 11, 4 (1994), 36-45.

[19] C. O'Dell and C.J. Grayson. *If Only We Knew What We Know: The Transfer of Internal Knowledge and Best Practices*, Free Press, 1998.

[20] D.L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications ACM,* 15, 12 (1972), 1053-1058.

[21] R.S. Pressman. *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 2004.

[22] W.R. Scott. *Organizations: Rational, Natural and Open Systems, 4th Edition*, Prentice-Hall, 1998.

[23] M. Stephens and D. Rosenberg. *Extreme Programming Refactored: The Case Against XP*, Apress, 2003.

[24] K.J. Sullivan, W.G. Griswold, Y. Cai and B. Hallen. The Structure and Value of Modularity in Software Design. In *Proceedings of the International Conference on Foundations of Software Engineering* (FSE '01), Vienna, Austria (2001), 99-108.

[25] J.D. Thompson. *Organizations in Action: Social Science Bases of Administrative Theory*. McGraw-Hill, 1967.

[26] K.E. Wiegers. *Peer Reviews in Software: A Practical Guide*. Addison-Wesley, 2001.

[27] K.Y. Williams and C.A. O'Reilly. Demography and Diversity in Organizations: A Review of 40 Years of Research, *Research in Organizational Behavior*, 20 (1998), 77-140