Siemens Global Studio Project: Experiences Adopting an Integrated GSD Infrastructure[1]

Mullick, N., Bass, M., El Houda, Z., and Paulish, D.J.
Siemens Corporate Research, Inc
Princeton, NJ
Neel.Mullick, Matthew.Bass, Daniel.Paulish @Siemens.com

Cataldo, M. and Herbsleb, J.D.
Institute for Software Research International
Carnegie Mellon University
Pittsburgh, PA
Mcataldo,herbsleb@andrew.cmu.edu

Bass, L.
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA
ljb@sei.cmu.edu

Sangwan, R.
Penn State University
Great Valley Campus
Malvern, PA
rsangwan@psu.edu

Abstract

Environments and processes in typical software development are not fully adapted to the needs of global software development (GSD). In particular, they do not have all of the capabilities necessary for cross-site collaboration. While research literature is rich with examples of individual practices and tools that an be used in this setting, there is a lack of examples illustrating how these tools and processes can be used in combination. We have augmented a set of tools and processes for GSD and applied them to an experimental project called the Global Studio Project (GSP). This paper describes the tools and processes developed, and insights gained from applying them to the GSP.

1. Introduction

As researchers and practitioners gain more experience with GSD, we are beginning to move from individual tools and practices that help with particular aspects of GSD toward integrated solutions that attempt to cover the full range of needs of distributed projects. The fundamental GSD problem, of course, is that co-located projects make very effective use of coordination mechanisms and communication channels that are not available or not as effective for distributed teams (e.g. [4], [3]). The fact that these mechanisms operate so naturally and invisibly has made adjusting to GSD all the harder, since we tend to underestimate the difficulties and to be unsure about what additional capabilities GSD projects require. In addition, differences in language, culture, and physical context make it much more difficult to establish common ground to ensure that communications carry the same meaning for all parties [1][11].

[1] We gratefully acknowledge support by the National Science Foundation under Grants No. IIS-0414698 and IIS-0534656, by the Software Industry Center at Carnegie Mellon University and its sponsors, especially the Alfred P. Sloan Foundation, the DoD and Siemens Corporate Research, Inc.

While it is clearly an oversimplification, much of what has been learned about GSD can be summarized in the following themes:

Decoupling the work. Interdependent tasks require some type of coordination mechanism and, potentially, extensive communication exchanges [10]. Modularity plays an important role in software design, in general, and it is particularly vital for GSD [5]. The drastically attenuated communication across sites makes it difficult to manage dependencies among developers working on the same modules.

The importance of unplanned and informal communication. Informal communication pays a major role in project coordination, particularly where the project is uncertain [9]. Since informal communication is nearly absent across distributed sites, uncertain projects are likely to suffer from lack of communication.

Project management must be both more flexible and more rigid. Different sites are going to operate somewhat differently because of differences in history, culture, expertise, and project management must accommodate such differences. It is much easier, however, to be surprised by delays and technical setbacks in a GSD environment.

The lack of contextual information causes misunderstandings and confusion. Different sites operate within different cultural norms, communication styles, different tempos, and are subject to different issues and priorities [1][8]. The lack of information about the context in which other developers are immersed leads to frequent misinterpretations [2]. This lack of mutual understanding leads to conflict as when unresponsiveness is interpreted as irresponsibility rather than a consequence of time pressure [7].

There are many practices and tools that could be used to address these concerns, and the research literature is rich with descriptions of individual tools that have been developed and used in academic and industrial settings. While these studies are quite valuable in advancing our state of knowledge, looking at tools and practices individually does not tell us much about what happens when we use them in combination on distributed projects. We have much to learn about conflicts and complementarities among them, and to understand what kinds of adjustments it takes to attempt to put an end-to-end solution into place. One of the reasons for the absence of research on total solutions is, of course, that it is very risky to introduce simultaneously many different changes in practices and tools in a real project.

One alternative to introducing wholesale changes into an actual project is to use surrogate projects, such as simulated with student teams. This paper reports initial results from a relatively large-scale student project, in which multiple teams from a number of universities on four continents collaborated on year-long projects in order to create a laboratory in which the effects of multiple GSD practices and strategies could be studied.

2. GSP Overview

The Global Studio Project (GSP) is a project that simulates a real world geographically distributed project by using student teams to develop software. The GSP was initiated by Siemens Corporate Research (SCR) in order to develop a better understanding of the issues associated with developing software using geographically distributed teams.

In the GSP there is a central team located at SCR that is responsible for the requirements, software architecture and some aspects of design, system test, integration, project management and defining the overarching processes. The remote teams are responsible for design, development and unit tests for particular code modules or sub-systems defined by the central team. The interactions between the central and each remote team are managed by an individual on the central team that we call a Supplier Manager. There is a Supplier Manager for each remote team.

Table 1: School Participants of the GSP

	School	Country	Dates Involvement	
			Start	End
Year 1	CMU	USA	09/2004	08/2005
	MU	USA	09/2004	04/2005
	UL	Ireland	11/2004	06/2005
	TUM	Germany	11/2004	06/2005
	IIITB	India	11/2004	06/2005
Year 2	CMU	USA	09/2005	03/2006
	MU	USA	09/2005	04/2006
	PUCRS	Brazil	08/2005	07/2006
	UL	Ireland	11/2005	06/2006
	TUM	Germany	11/2005	06/2006
	IIITB	India	01/2006	06/2006

The GSP is executed in distinct one year increments (corresponding to the academic year) and is currently in its second year. The remote teams are student teams operating in academic environments at universities around the world. These students are pursuing their masters or diplomas (equivalent to masters) in software engineering or associated fields. The programs, universities represented and the approximate time frames of their engagement with the GSP are described in Table 1.

The system under development in the GSP is what we call the MSLite system. MSLite is to be 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 summary of the high level functional requirements of the MSLite system are:

- Manage the field objects (objects in the building automation domain, e.g. HVAC sensors) which are represented in the FSS.
- Issue commands to the field objects in the FSS to change values of their properties.
- Define logical conditions based on property values of field objects that trigger reactions and issue commands to field objects.
- Define alarm conditions akin to the logical conditions that when met, trigger alarms notifying appropriate users.

3. GSP Development Effort

The GSP was split into two distinct years with a particular set of student teams being involved for one year. For the second year the infrastructure and processes were refined to address the issues experienced and lessons learned during the first year.

At the time this report was written the GSP project had completed 2/3 of the second year. In this section we will describe the design of the project in the first year, how it was intended to deal with the perceived issues associated with GSD, the results, the redesign for the second year, and where available further results. These findings have been separated into themes that ended up being central in this project in one way or another.

3.1 Centralized Management

During the planning process for the GSP there was much discussion about how to structure the overall management of the project and how to conduct the early lifecycle activities. In the end our strategy was governed by past experience [4] and the constraints of the environment. The tenets of our approach were as follows:

- High level process defined and enforced centrally, but the detailed development processes were the responsibility of the remote teams
- Items of global concern (e.g. system requirements, architecture, system testing) would be the responsibility of a central team, local concerns (e.g. development, detailed design, unit testing) would be the responsibility of the remote teams
- The system would be built using iterative and incremental development
- Coordination would be managed centrally

How these tenets were realized changed from year 1 to year 2, however. We will discuss the details of the approaches below.

3.1.1 Year 1. The development for the first year was to occur in an iterative and incremental fashion. The central team was to deliver a set of artifacts (called a work packages) at the beginning of each iteration. The idea was that this work package would contain all of the artifacts necessary for the remote teams to complete their assigned tasks in that iteration.

There was a person that we called a supplier manager designated to manage each remote team. Because of the overhead associated with managing a remote team it wasn't possible to have a single person manage all the teams. Each supplier manager was responsible for communicating expected tasks, monitoring progress, being the first point of contact for questions, and facilitating the communication between the remote team and other members of the project as needed. It was planned that the supplier managers would have a kick-off meeting via teleconference with their remote teams at the beginning of each iteration. Additional meetings would be held as needed.

Remote teams were to deliver code, design artifacts, test cases, and associated documentation at the end of each iteration. They were to make the delivery into a central version control system. The central team

would then integrate the deliveries and conduct system tests that corresponded to the features that were to be delivered in the previous iteration. Issues that were discovered were to be assigned to the team that owned the code that was perceived to be responsible for the issue. Responsibility could be reassigned if required by the central team.

Coordination across teams was to be managed by the central team. If a remote team needed to coordinate with another remote team they were first to inform their supplier manager. The supplier manager would then let them know if it was appropriate to contact the remote team and be party to all of the communication.

There were many issues with the central management of the first year. Before teams were able to become fully productive several of them became quite frustrated with the project and became increasingly difficult to interact with. In two instances onsite visits were required and the academic advisors were brought into the discussions. The first year of the project was eventually stopped three months early as it was obvious that the current processes were not adequate and we wanted to use that time to re-plan for the second year. We summarize the related issues below that led to these conditions.

Unclear process communication. The process was communicated to the teams individually either face to face, via teleconference, or via electronic media (email or electronic documents). It took much longer than expected, however, for the remote teams to understand the processes, what was expected of them, how they make deliveries, the meaning of the work packages, and so forth. In most cases confusion and re-explanations of various aspects of the process were required for the first several months of involvement.

Inconsistent communications from the supplier managers. The internal processes and infrastructure was not adequate to ensure that the supplier managers provided consistent communications across teams. The supplier managers would respond to emails and questions as they saw fit. It was not uncommon to later discover that another supplier manger responded to a similar question from another team differently. These inconsistencies resulted in the central team seemingly "changing their mind" and contributing to the perception of disorganization.

It was also the case that the coordination processes ended up being different for the various teams. For example the IIITB team in India rarely participated in teleconferences. They were difficult to schedule based on the time differences, the student team had difficulty securing a location where they could participate in such a teleconference, and it was felt that for either language or cultural reasons the student team had a strong bias towards written communications.

CM and infrastructure problems. There were several issues associated with the CM process and infrastructure. Some of the teams were using the central version control system as a working repository during the iterations. Others had their own local repository with a structure that mirrored the central one. There was no branching of the main trunk for the remote teams. We had several issues with this such as:

- Broken builds when teams checked in interim code artifacts
- Hierarchy of the main repository changing making it difficult to merge the changes of the local team
- Remote Teams reporting bugs in other teams code when in fact the updated code hadn't been checked in yet.

3.1.2 Year 2. During the second year several tactics were adopted to address the issues mentioned above. In this section we will describe these tactics and where available discuss the results.

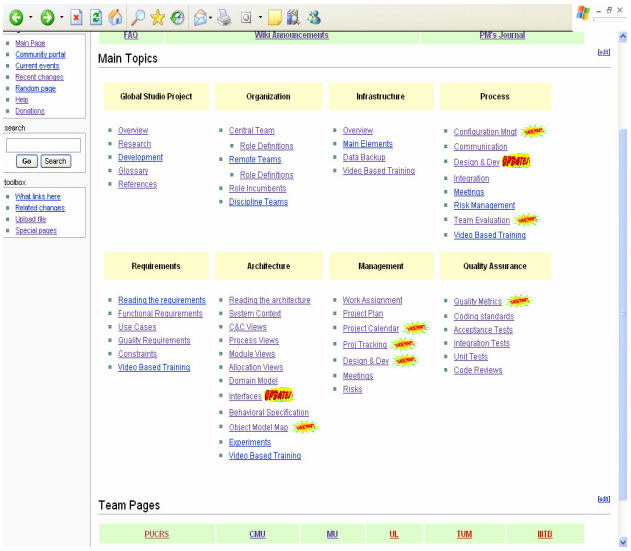

Fig. 2 – The main page of the MSLite wiki

Consistent communication with the remote teams. The central team put processes and infrastructure in place to help ensure that the communications with the individual teams were consistent across teams. A

central infrastructure was created to provide a single window into the project (see figure 2). MediaWiki was selected as the software to support the main entry point to the project. All project related artifacts and process description were made available via this infrastructure. Training videos were developed and made available on the wiki as well.

When a team joined the GSP they first had a kick-off meeting with their supplier manager. The supplier manager followed a detailed script for the meeting. The teams were then required to watch the training videos and conduct a series of exercises to help ensure that they understood the infrastructure and knew how to use it.

All of the teams had weekly status meetings with their supplier managers. These meetings were conducted according to a structured format. All questions and issues posed by the remote teams were required to be submitted to the central team prior to the meeting. This allowed the central team to formulate consistent answers and not reply on the fly.

These changes were largely successful. It was the case that all teams became productive within one month and no team required retraining at any point unless there was a major change to the infrastructure or the process.

Transparency. It was mentioned that during the first year the remote teams were frustrated by the personal impact of design changes and re-planning by the central team. It was felt that much of this frustration came from the remote team not understanding the context leading up to a decision. If the remote team was aware of the issues and party to the decision they would (perhaps) see that the best option for the project overall was to proceed in a way that may burden a limited number of teams. In order to promote this sense of team, the central team worked to make the entire project transparent and accessible to all. Part of this involved the infrastructure mentioned above. In addition a change control board (CCB) and change control process was defined. The CCB included the team leads from the remote teams. This allowed them to understand the rationale for changes and take ownership of the decision to proceed despite potentially adverse impacts to their team.

Mandatory central version control system. Subversion was selected as the version control system to be used by all. A continuous test and build process was defined and supported by CruiseControl (a test and build environment). The infrastructure would automatically build the system and execute test cases when code was submitted. The results of the build and tests were emailed to the central team as well as the team that submitted the code. If the build was broken the team was given a short amount of time (on the order of a ½ hour) to fix the build. If the build was not fixed their submission was backed out and the code base went back to that of the previous successful build. This would allow for all teams to work from a single code base without impeding the progress of any of the teams. If the build was allowed to remain broken for an extended period of time, it would potentially hold up the progress of all teams. Issues from the test cases were reported at the end of an iteration (we will talk more about this in the next section).

3.2 Iterative Development

We anticipated having issues estimating effort, monitoring progress, managing quality, and re-planning. In order to deal with this uncertainty we chose to adopt an iterative and incremental approach. We borrowed some of the practices of SCRUM [12] such as short iteration cycles, the notion of functioning deliveries, and so forth. Below we will look at the way the iterative aspects of the project were realized in the two years and the results.

3.2.1 Year 1. In the first year the development was to be done in a series of short (4 – 6 weeks each) iterations called sprints [12]. At the end of these sprints each team would deliver a portion of their module. These modules were to then be integrated by the central team to realize a predefined portion of the externally visible functionality. The central team would integrate and test the system assigning issues back to the team that owned the module likely responsible for any problems.

The development was to progress incrementally as well. The high level functional requirements were to be defined and prioritized, the initial functional decomposition of the system was to be done, sub-systems defined, and high-level planning done. The requirements and architecture would then be refined as needed to achieve the functionality defined for the upcoming sprint. The detailed tasks would be allocated to the teams, the work executed, deliveries made, and integration and system test would occur in the subsequent sprint.

The requirements and the design were largely done in UML. The tasks were derived from mapping the

features planned for a given sprint onto the architecture. Responsibilities within the impacted modules were defined and assigned to the teams that were responsible for the given module.

The project plan was documented in Microsoft Project and had the features as milestones. The features were broken down into responsibilities that corresponded to those identified in the process described above. The responsibilities were allocated to the appropriate team on the project plan.

All of these artifacts were emailed as a work package to the teams at the beginning of each sprint. In addition there was a kick-off meeting with the supplier manager responsible for the team and the remote team. This was typically held via teleconference. Subsequent meetings were scheduled on an as needed basis. Typically questions were posed via email.

As mentioned in the last section, the planning and execution of the first year of the project did not run smoothly. Below we summarize the issues as they relate to the iterative nature of the project.

Frequent re-planning required. From the beginning of the first iteration unanticipated issues began to surface. Teams frequently required additional information related to a task that hadn't been completed yet. In some cases the dependencies were things like: team A discovered that they required the definition of a particular portion of the object model (that was to be defined in an upcoming iteration by team B) in order to complete their tasks, or team C realized they needed previously unspecified services from another portion of the system outside their control.

When this would occur the team would be unable to move forward. The central team wasn't sure how to respond to these issues. They didn't want to have teams sitting idle, but were also unable to adequately provide answers to the questions. The central team acted in one of two ways; They made some assumptions, gave the team an answer based on those assumptions and told them to continue assuming that was correct, or they reassigned the team to another task. Both of these options often did not work out well. The alternate task was often busy work, and the assumptions were often not totally correct.

Interdependence of tasks not well understood. Related to the first item, the interdependent nature of the assigned tasks wasn't adequately understood. Some of this can be attributed to ambiguities in the architecture. While an architecture is often not totally stable until the system is retired, in this case the work was distributed before the architecture was *stable enough*. Ambiguities resulted in dependencies to be discovered during implementation, and because of the distributed nature of these teams the ambiguities were difficult to rectify and threw the work plan into disarray.

Additionally the temporal dependencies were not adequately investigated. In other words there was not an adequate understanding of what the prerequisite work tasks were in order to complete a given set of functionality during an iteration. Functionality was assigned to an iteration by priority rather than ordering them according the identified temporal dependencies.

Work assignments not easily understood. The teams had trouble understanding what was expected of them. They had trouble making sense of the artifacts (project plan, architecture, and requirements) supplied in the work packages and teleconferences were often not adequate to clarify questions.

Integration tests weren't easy to execute. The original test plan mirrored the project plan. The plan for integration and system tests was derived from the requirements associated with the features planned for a given integration. Because of the ad-hoc re-planning that occurred in the middle of iterations, however, the actual deliveries often didn't match the planned deliveries. The result was that it wasn't clear what the central team needed to integrate and test. The result was often untested and poor quality deliveries.

3.2.2 Year 2. While we continued to have an iterative and incremental approach in year 2, we changed many things as a result of the issues experienced in the first year. While we still believed iterative development would help with many concerns, we felt that more upfront work was required prior to engaging the remote teams and beginning the iterations. In addition more structure and better use of tools was required to support the iterative processes. In this section we will describe these changes.

More upfront effort. While it was still believed that some aspects of the requirements and design could happen iteratively, it was also now believed that a larger percentage of the efforts would be needed prior to engaging the remote teams. In addition, much more

effort was focused on understanding and planning for the interdependent nature of the tasks.

In addition to describing and modeling the requirements, UI mockups were created. These mockups helped to convey the meaning of the requirement to the remote team. They also helped to identify issues with respect to technical feasibility early. In at least one case a remote team raised issues with respect to the chosen technology based on viewing the UI mockup (e.g. AJAX doesn't support the mechanisms needed to provide the views in the way specified in the requirements and made evident by the mockup). This allowed adjustments to be made to the requirements prior to the sprint in which this functionality was to be implemented.

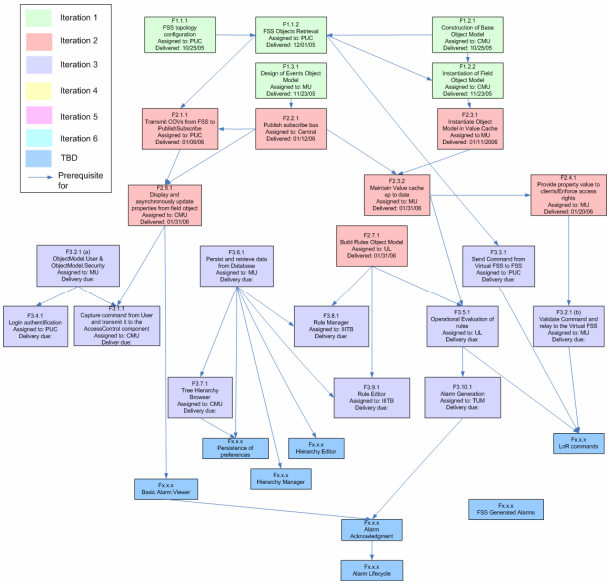

Fig. 4 – Functional dependency diagram

For the second year the central team worked much harder to understand the nature of the interdependencies in the architecture. Particular attention was paid to the temporal dependencies (i.e. which tasks needed to be completed first). In order to understand these dependencies, two views of the system were created. One view highlighted the functional dependencies (see figure 4). This view showed the dependency between the functions of the system. These functions were then allocated to iterations.

In addition to understanding the functional dependencies the central team wanted to understand the dependencies among the modules required to implement these functions. Another view was created to highlight this. Each iteration from the functional dependency diagram was mapped to a modular view of the system. The modules required to fulfill the iteration were color coded according to the iteration. This highlighted things such as dependencies on the data model which wouldn't be instantly obvious from other views such as the sequence diagrams (and caused issues in the first iteration).

Additional views of the system were developed to help with planning such as a technology view which identified the technologies required for any given subsystem. The idea was to limit the number of technologies that each team needed to learn in order to implement their modules. Additionally prototypes were done by the central team to help ensure that the architecture was understood and stable prior to distribution.

Improved tool support. As previously mentioned the infrastructure was changed drastically for the second year. Much of the new infrastructure was modeled on that used by the open source community (e.g. the Apache projects). This infrastructure became integral for communicating the requirements, design, and plan. "Work packages" were done away with and instead all information was posted on the Wiki. In addition to the modeled requirements and associated textual descriptions all of the requirements were listed in a "traceability matrix". This matrix had links to the high level requirement, the associated use cases, any relevant sequence diagrams, the associated test cases, any UI mockups, and the task on the task plan. The numbering system indicated the iteration, the responsible team, and an id for the requirement.

The work plan was generated on the Wiki itself. For each iteration it contained the high level goals, the use cases, the development focus for each team, the impacted modules, components, and interfaces. This allowed for all associated information to be referenced in a single place. After the first iteration no further explanation was required to any of the teams, so this mechanism seemed to be much improved.

3.3 Information and Awareness

One governing principle for this project was that we wanted to minimize the need for cross team communication. It was recognized that it was impossible to eliminate the need entirely, but the central team wanted to manage all cross team communication. The rationale for this was that

decisions made between two teams may have a wider impact and the central team wanted to be aware of these communications and ensure that the overall goals of the project were considered. In this section we will talk about our initial ideas in this area and how they evolved.

3.3.1 Year 1. In the first year we felt that the primary vehicle for coordinating was going to be the requirements, architecture, and the project plan. We distributed what we called "work packages" for each sprint. The work packages contained the updated architecture, the associated requirements and the project plan along with task assignments for the upcoming sprint. It was expected that this along with a kick-off teleconference with the supplier manager would be adequate for the teams to understand what was expected of them. In addition we set up email aliases for the team to use to communicate with the central team. If there were no requests from the remote teams during the sprint they were left alone to complete their tasks.

The issues associated with this approach have been sufficiently covered in the previous sections. Suffice to say that the processes and infrastructure used to augment the awareness of the remote teams in the first year were far from adequate.

3.3.2 Year 2. As already mentioned a central infrastructure was created so that all participants would have a single window into all aspects of the project. Much thought went into the best way to structure the processes and infrastructure to support awareness within the remote teams. Many related aspects have been discussed already (e.g. training, transparency in the project, and the integrated infrastructure). In this section we will talk about additional elements of the processes and infrastructure that have not yet been mentioned.

Ensuring a consistent understanding. In the first year there were problems with the teams adequately understanding the directions and clarifications of the central team. While some of these problems were immediately addressed by new structure of the project, some problems lingered. Early in the execution of the second year the central team realized that they were having a tough time gauging the level of understanding of the remote teams during teleconference meetings. This realization came only after reviewing artifacts delivered by the remote team and conducting end of sprint post-mortems.

To address this issue the central team mandated the posting of meeting minutes on the wiki. In the meeting minutes the remote teams were required to summarize in their own words both the questions asked and the answers given. This allowed the central team to review the contents of the minutes and further gauge the level of understanding.

Consistent communications. In addition to previously mentioned actions to support consistent communications with all the remote teams the central team adopted a practice of answering all emailed questions via a discussion forum. Initially the central team would directly answer all emailed questions via email. The central team, however, found that they were spending significant time answering the same questions from different teams. In order to reduce the overhead on the central team and to ensure consistent answers, the central team set up a discussion forum. In this forum they created topics with threaded discussions. Whenever a new question was posed the central team would post the question and answer in the discussion forum. Repeat questions would be referred to this forum. Eventually remote teams got trained to look there first and to begin to use this as a means for asking questions and communicating across teams.

Universal participation. With some of the remote teams there was obvious participation from only one or two people. At first the central team assumed that this was because these were the people that were most comfortable in English, but later suspected that these were the only people who were actively contributing to the project. To deal with this situation the central team required that each team member fill one of the required roles and that only that person respond to questions regarding topic relevant to their role. This improved the participation of all of the team members. It is difficult to know the result in terms of their contributions (we plan to check the repository logs), but the central subsequently felt more comfortable with the overall participation.

3.4 Formality of the Specifications

Initially the planners of this project had the idea that the more formal the specifications were the less the remote teams would need to communicate with either the central team or the remote teams. In this section we will talk about the formal aspects of the specifications in year one and year two along with our findings.

3.4.1 Year 1. Much of the effort making more "formal" specifications went into the requirements. The requirements were modeled in UML as an acyclic directed graph (i.e. a hierarchy) with the higher level nodes being abstract features and the leaf nodes being the concrete detailed use cases. The idea was that by describing the requirements in this way we reduce the "gap" between the requirements and the design (in other words the process of going from the requirements to the design is much more mechanized) and thus reduce ambiguity. The design (also done in UML) would then offer external interfaces that corresponded directly with the business services detailed in the requirements.

As was mentioned several times already this didn't work out. While specifying the requirements in this way do make them more readable by a machine (it is possible to automate certain activities), they are not intuitive particularly to people who didn't develop the model and are not familiar with the domain. The remote teams were not able to look at the requirements model and easily develop an understanding of what the system was to do and how this related to their activities.

Additionally, this view of the system focused primarily on the external aspects of the system. While we did develop an architecture that described the internal aspects of the system, it was not sufficient to understand all that we needed to know about the interdependent nature of the tasks as well as to convey the aspects that we did understand to the remote teams.

3.4.2 Year 2. In the second year we didn't place as much effort on "formal" specifications. The requirements were primarily text based requirements. While the architecture was much more detailed it was not "formal" rather it tried to be intuitive. Instead much effort was placed on trying to formalize the processes that surrounded the artifacts. In other words we tried to structure and convey how the remote teams would take the artifacts provided and use them to extract the needed information.

This proved to be much more successful. The teams were able to use the artifacts as intended. There were some areas where the specifications were ambiguous and issues continued to arise, however. We didn't for example specify pre and post conditions. This led to conflicting assumptions in at least one case and eventually resulted in rework. Likewise, as previously mentioned, teams initially implemented exactly what was written even when it clearly didn't make sense.

4. Conclusions

With the benefit of hindsight, it seems obvious why the practices employed in the GSP during the first year weren't successful. At the time, however, they seemed reasonable. We had augmented practices that had been successfully employed in collocated projects. We underestimated, however, the extent to which ad-hoc interactions can fill the gaps and resolve the conflicts left by ambiguous specifications, misunderstandings, and poor planning.

In the second year we did a much better job accounting for these issues. While the project is not yet over, the outcome to date has been largely successful. At the time of this writing we have over 15KLOC of tested functioning code realizing much of the planned functionality which is more than three times the output of the entire first year.

The improvement in the second year can largely be attributed to an increased recognition of the areas likely to raise questions. We put mechanisms in place to either eliminate the source of question before it arose (e.g. more stable specifications, increased training, and more visibility) or by putting measures in place to recognize earlier that a question or issue exists (e.g. validating understanding, implementing an inclusive CCB, and assigning roles and requiring participation from all).

This is not the end all and be all, however. While we can work to minimize the number of issues and questions, we cannot possibly eliminate them. We continued to spend more effort and time doing things like trying to track down the responsible party for an identified bug, re-planning in the event that development doesn't progress exactly as planned, and so forth. Things that, in a collocated environment, could be addressed by getting all concerned together in one room take much longer because of delay introduced by time zone changes, lack of awareness of what remote individuals are doing, and delayed recognition of the problem.

It is also unclear how the techniques used in the GSP context would scale for an industrial sized project. The scope and complexity for the MSLite system is clearly a fraction of what it would be for many commercial applications. There will likely be some issues with the infrastructure and processes when

trying to deal with hundreds or thousands of requirements, understanding the needs of an embedded real time system, or complying with the constraints imposed by organizational processes.

6. Next Steps

Over the next year we plan to further this work in a couple of ways. First we want to further analyze the data collected from the GSP. We have been collecting social network data as well as the logs for all of the infrastructural elements and would like to analyze the data to better understand how the remote teams coordinated via the infrastructure to complete their work.

In addition we want to take a look at other projects that have more complex architectural concerns such as real time performance needs, fault tolerant concerns, or memory constraints to better understand the role that these aspects of the architecture play in coordination. In particular, we want to be able to predict, from the architecture documentation, how best to support coordination between teams that design and construct different components.

6. References

[1] Armstrong, D. J., P. Cole. 2002. Managing distances and differences in geographically distributed work groups. P. J. Hinds, S. Kiesler, editors. *Distributed Work*. MIT Press, Cambridge, MA, 2002, pp. 167–186.

[2] Cramton, C.D. The mutual knowledge problem and its consequences for dispersed collaboration. *Organization Science*, 12, 3 (May-June), 2001, pp. 346-371.

[3] Damian, D.E. and Zowghi, D. An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03)*, 2003.

[4] Herbsleb, J.D., Paulish, D.J., and Bass, M. Global software development at Siemens: experience from nine projects. In *Proceedings of the International Conference in Software Engineering (ICSE'05)*, May 15–21, 2005, St. Louis, Missouri, pp. 524-533.

[5] Herbsleb, J.D. and Grinter, R.E. Architectures, Coordination and Distance: Conway's Law and Beyond. IEEE Software, Sep-Oct, 1999, pp. 63-70.

[6] Herbsleb, J.D. and Mockus, A., An Empirical Study of Speed and Communication in Globally-Distributed Software Development. *IEEE Transactions on Software Engineering*, *29,* 3 (2003), pp. 1-14.

[7] Hinds, P.J. and Bailey, D.E. Out of sight, out of sync: understanding conflict in distributed teams. *Organization Science*, 16, 6 (Nov.-Dec.), 2003, pp. 615-632.

[8] Kiesler, S. and Cummings, J.N. What do we know about proximity and distance in work groups? A legacy of research. P. J. Hinds, S. Kiesler, editors. *Distributed Work*. MIT Press, Cambridge, MA, 2002, pp. 57–80.

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

[10] Malone, T.W. and Crowston, K. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26, 1 (March), 1994, pp. 87-119.

[11] Olson, G.M. and Olson, J.S., Distance Matters. *Human-Computer Interaction*, *15,* 2 & 3 (2000), pp. 139-178.

[12] Schwaber, K. and Beedle, M., *Agile Software Development with Scrum, 1st Edition*, Prentice Hall PTR, Upper Saddle River, New Jersey, USA, 2001.