Architectural Misalignment: An Experience Report

Bass, Matthew
Siemens Corporate Research, Inc
Matthew.Bass@Siemens.com

Mikulovic, Vesna
Faculty of Informatics, Vienna University of Technology
Vesna.Mikulovic@Siemens.com

Bass, Len
Software Engineering Institute
Carnegie Mellon University
ljb@sei.cmu.edu

Herbsleb, James; Cataldo, Marcelo
Carnegie Mellon University
jdh@cs.cmu.edu
mcataldo@andrew.cmu.edu

Abstract

It has been well documented that there is a correlation between the structure of an architecture and the organization that produces it. More concretely there is a correlation between task interdependencies and coordination among the people or teams realizing these tasks. The amount of coordination needed among teams is related to the nature of these task interdependencies. As the scale and complexity of organization and systems grow it is not uncommon to have factors such as geographic boundaries, organization boundaries, cultural differences, and so forth impede the ability of certain individuals or teams to coordinate effectively. While there is some understanding of the factors that impede the ability of teams to coordinate, the factors that cause task interdependence in software systems is less well understood. The current view is that it is the interactions across module boundaries (assuming a module is assigned as a task or work item to a single team) that cause task interdependence; we have found that this view is not sufficient. In this paper we present three cases where additional architectural mechanisms created task interdependencies that the organizations were unable to accommodate. We go on to discuss the implications of these findings and suggest future research activities.

1. Introduction

It has been well established that there is a relationship between a system's architecture and the structure of the organization developing it [3][3]. Essentially this means that, to a large extent, dependencies among software engineering tasks will mirror the technical dependencies among components in the architecture.. Factors such as organizational boundaries, geographic distance, and cultural differences reduce the ability of teams to coordinate [7][8]. When these factors are present it is important to ensure that these teams do not have tasks which create a need for them to coordinate beyond their ability to do so. Evidence suggests that the consequences of assigning such tasks to teams that are not able to effectively manage the coordination (we call this architectural misalignment) can be quite severe [5][13].

Currently the view is that the technical mechanisms that cause these task interdependency are invocations across modules (assuming a module is a task assignment to a single team) [3][6][7][13]. This view leads to a relatively narrow focus when architects and managers attempt to align the architecture with the organization. We have found that this narrow view is not sufficient. There are additional architectural mechanisms such as state management, resource utilization, and schedule synchronization which can also require extensive interaction among teams.

In this paper we examine three projects where such mechanisms resulted in architectural misalignment such that the organization was unable to produce the intended design. The contribution of this paper is to identify and present brief case studies of several as-yet unrecognized sources of misalignment. Awareness of potential problems is the first step toward being able to manage them successfully or to choose work assignments that avoid them altogether.

We can summarize our argument in two main points:

1. *The traditional view that it is only the module interactions that create task interdependencies is not adequate.*
2. *Project management and architects together need to recognize the factors that impede coordination, the architectural mechanisms that imply coordination, and the alignment between the two.*

In the next section we give the background on the relationship between architecture and the organization, what is known about the potential impact of having a misaligned architecture, what is currently understood about the aspects of the architecture that create a need to coordinate, and the kinds of connectors recognized by the architectural community. We then give the details for three systems examining the technical decisions that were made, the resulting need for coordination, and the issues that were experienced when trying to coordinate appropriately. In the final sections we discuss our conclusions from these observations and propose future work to address the questions raised.

2. Background

2.1 Organizations and Architectures

That interpersonal coordination is an important aspect of software engineering is not in question [1]. As the scale and complexity of the system grows, the coordination needs grow as well. Melvin Conway recognized this in his now widely known paper "How Do Committees Invent?" [3]. Conway, recognizing that the structure of the architecture dictated the need for coordination amongst technical folks, reasoned that to manage this coordination the system would need to be split into components with limited technical dependencies, and these components would need to be assigned to no more than one team (resulting in a homomorphic relationship between the architecture and the organization).

Since Conway's paper, others have recognized the relationship between the architecture of a system and the structure of the organization that produces it. In his paper on modular design [6], Parnas considered modules to be work units. Limiting the interactions amongst modules would allow them to be developed in parallel by separate teams with minimal interactions among them. Kraut and Streeter indicate that while certain factors increase the need for coordination or impede the ability of an organization to coordinate effectively, the original need for coordination comes from dependencies in the architecture that span tasks [1].

2.2 Impact of Architectural Misalignment

The importance of these ideas has become clearer over time. As the scale and complexity of systems continues to grow, the ability to have unrestricted coordination amongst all team members diminishes. It is not uncommon to have organizational or geographic boundaries exist amongst development teams, impeding the ability to coordinate [7][8]. As Henderson and Clark noted in [5] the consequences for having an architecture misaligned with an organization can be disastrous.

2.3 Current Understanding of Alignment

Given the fact that there needs to be an alignment between the coordination needs imposed by an architecture and the coordination capability of an organization, it makes sense to understand specifically what aspects of an architecture imply the need to coordinate. Starting with Parnas's work on decomposing systems into modules (in this paper we use the term "module" and "component" interchangeably) there was a recognition that invocations from one part of the system to another imply the teams developing these parts of the system will need to coordinate. Thus, by having "loosely coupled" components with well defined invocations amongst them, teams can work relatively independently coordinating only on these narrowly defined interfaces that span components [6][9][9][13]. In other words current wisdom (based on the literature and our experience in practice) believes that it is the invocation mechanisms for connecting components (e.g. component A "calls" component B) that creates the need for organizational coordination.

Our experience is that this belief influences the decisions made by architects and project managers. As we will see in some of the examples given in this paper, architects will explicitly consider work allocation when decomposing a system. In addition extra effort will be made to "loosely couple" or minimize the complexity and number of invocations between components assigned to teams that have a limited ability to coordinate.

2.4 Architectural Connectors

As recognized by the architectural community, however, there are other kinds of connectors that require coordination between architectural entities of the system ([11], [12]). Things like timing schedules, shared resource, and state management also impose the need for coordination between components of a system. These kinds of connectors are not, however, typically considered explicitly when looking at the coordination needs imposed on an organization by an architecture.

The practical result of this line of thinking is that project managers need to understand the aspects of the architecture that are relevant for their concerns (e.g. structuring the developing organization, task allocation, setting up the development and management processes, and monitoring the progress of the development). With the current understanding they would use measures of coupling and a call graph of the architecture as input. For example, if two teams are geographically distributed and less likely to be able to coordinate effectively, they should be given components of the system that have few invocations between them.

Additionally architects typically deal with the most important decisions first and defer less important decisions until later. The importance of decisions is typically determined by some combination of importance to the organization and some measure of risk. If based on the structure of an organization, certain decisions become more or less risky, this has an impact on how important they are and how much attention they should receive from the architects. In order to do this, however, there would need to be some explicit recognition of the ways in which the architecture and the organization relate.

3. Case Studies

3.1 System 1

System Goals: System 1 is a platform designed to support a family of real-time embedded products. These products vary across several dimensions including features supported, available memory, and timing requirements, and within a given product segment the additional functionality is expected to be added regularly over the lifetime of the platform.

The main motivators for the architecture team were:

- Flexibility: As the platform was intended to support a diverse customer base with a variety of needs the platform needed to be able to quickly and easily be instantiated in configurations supporting a variety of hardware devices and software applications.
- Easy to Use: Some customers required the ability to be able to extend the system to support custom Human Machine Interface (HMI), and so the platform had to support this with a minimum of cost and training.
- Cost and Risk Minimization: As the market demanded high quality and reliability, the company developing the platform wanted to minimize the "non-compliance costs" (costs associated with recall of the devices from the field).
- Reduce Time to Market: There was a desire to reduce the time to market for the product instantiations developed with the platform.

Ease of development was explicitly considered as a primary motivation.

In addition the system had a set of performance, reliability and resource utilization requirements. There were specific performance requirements dealing with system startup that indicated the allowable latencies for bringing particular functionality online. There were also memory utilization requirements that could not be exceeded.

System Description: Considering the needs that the system had to fulfill (described above), the architecture team settled on several approaches. These approaches were selected based on their proved success in other projects. The primary architectural concepts chosen were:

Frameworks: A framework was developed to provide the basic building blocks for a family of components. The framework supported common concepts and helped enforce compliance with the defined Meta Architecture. Use of a framework was selected in order to help address the need to allow for reduction in time to market, the ability to allow other developers to be able to quickly and easily extend the system (e.g. by adding a custom HMI), and to mitigate quality risks.

Component Based Solution (CBSD): System 1 was developed as a component based solution using a component integration framework. This framework handled the lifecycle management of the components and allowed for upgrade scenarios both in the field and at development time. The components were "loosely" coupled with well defined interfaces allowing for development by independent teams (at least in theory). The idea was that this would allow for increasingly parallel development, as well as simplify integration issues at various stages of the lifecycle.

Dependencies against interface specifications rather than component implementations: Inline with the desire to maintain "loosely" coupled components that could be swapped out at any point in the system lifecycle; the teams were to ensure that they developed their particular modules against the interface specifications. This meant that they were not to implement them in such a way as to be dependent on the component implementations of another team.

In order to address the performance and resource utilization requirements the architects borrowed heavily on past solutions. These were standard kinds of requirements in this industry and several legacy products existed that had fulfilled similar requirements. The architectural mechanisms include:

Separation of concerns: The system had several types of memory, CPU, Cache, RAM and Flash. The architecture group restricted the use of various kinds of memory for particular purposes, and partitioned the RAM into several logical partitions each for its own part of the system. For each part of the system (OS, dynamic allocation for object management, native portions of the system, java partitions, …) memory utilization was estimated (or in some cases measured). Much of the functionality lived in the same memory partition.

Priority based scheduler: The performance requirements were to be dealt with by using a priority based scheduler. Thread priorities were assigned by using a coarse grained classification scheme allowing for 10 categories of threads. Start-up performance was to be optimized by saving state upon shutdown and restoring this state upon startup.

Organizational Structure: Previously the organization developed their solutions using tightly knit collocated teams. Over the last several years, however, several smaller companies were acquired resulting in development capabilities in several geographic locations. System 1 was intended to be a platform used for the next generation of products across the company including the products developed at these disparate sites. The design and implementation for System 1 was staffed with people from each of the development sites. As this was an effort of strategic importance with high external visibility, the best people from each site were selected to work on system 1.

There were in total four development sites with two in Germany and two in France. There was a central architecture team made up of the most qualified representatives from each site, and each site had responsibility for the design and development of one or more subsystems.

The architecture team would meet regularly (as often as every week) in various locations for design meetings. They were responsible for defining the high level architecture. This included selecting the architectural concepts appropriate to achieve the driving requirements, allocation of responsibilities to the components, interface specification for the primary components, and specification of concerns that spanned the individual subsystems.

The overall project was managed from one of the main sites in Germany with the project manager spending much of his time traveling to the various sites. The total development effort for version one was around 400 staff years implemented in roughly four calendar years. The size of each development site was more or less equal.

Issues Experienced: At the time this report was written, two versions of System 1 had been built. The system had been deployed in limited numbers in the field. While there were many issues, as is normal with a system of this size and complexity (this was an embedded system with over 3 million SLOC), some of the issues were not expected and had a major impact on the business success of the platform. The system had stability problems. It would "hang" periodically and give the perception of a software failure. In addition the system was not able to stay within its memory budget during startup. Performance was also a problem (in addition to the hanging problem) during startup (i.e. the system was unable to bring certain features online within the required time during startup).

Many of the smaller issues were solved, but the memory, performance, and stability issues (or hangs) were not so easily solved. The organization maintained the same structure for some time trying to track down and deal with these issues. When this proved to be unsuccessful they appointed a performance and memory management team to be in charge of this aspect of the system. This team was to manage the resolution of these issues with the architects and the involved design and development teams.

While overall some progress was made, the impact on the overall market's perception of the platform was significant. In order to realize specific product instantiations, additional features needed to be integrated into the platform. For business reasons they attempted to do this development and integration in parallel with the resolution of the performance and memory issues, further compounding the problems. These issues were the topic of board level discussions across companies and ultimately led to key customers switching to competing products representing a significant loss of revenue.

Analysis: In looking at the issues and the events that preceded the issues a couple things became apparent.

Resolving the performance and memory conflicts was a coordination intensive activity: Only informal calculations were used to determine if the prioritization scheme would achieve the performance requirements. This required extensive testing (and subsequent re-prioritization) to validate that these latencies could be met. When they were violated, extensive coordination between the teams familiar with the code competing for CPU time was required. There was no upfront recognition that this was a possibility. The architects assumed that the prioritization rules were adequate to allow the teams to work independently. This was largely due to a similar mechanism working for previous solutions. What was not recognized, however, was that much tweaking was required in the previous solutions as well. This tweaking didn't have the same coordination impact on the previous development team, however, as they were collocated and able to coordinate effectively without unduly disrupting the schedule or sacrificing quality.

The organizational structure inhibited coordination concerning performance and memory issues: While the architecture team was ultimately responsible for the performance and memory characteristics of the system, the implication of their decisions (the scheduling policy, and resources utilization policy) was that many decisions were deferred to the development teams.

The impact of the local decisions was not recognized until late in the development lifecycle. Once it was recognized, it was difficult to coordinate as many of the teams involved were distributed and only vaguely familiar with the activities of the other teams. After the difficulties of version one the organization tried to restructure accordingly and created a performance team. This performance team was responsible for identifying the source of the issues and managing the coordination amongst the involved teams to resolve the issues. This also proved to be problematic. While there was a single group that was responsible for the issues, much coordination was needed to understand exactly what was happening in the system at runtime and how the processes and threads were interacting to use memory and CPU time.

In this project the architecture team was explicitly concerned with the ability of the organization to deliver the system in a timely manner. This concern, however, only resulted in considering the component framework and functional decomposition of the system. There was no understanding of the impact of the performance and memory management decisions on the organization until it was too late. Likewise project management only became aware of the inadequacy of the existing organizational structure for this system once complex issues began to surface. They then tried to reorganize to address these issues, but it was too little too late.

3.2 System 2

System Goals: *System 2* was a PC based application that interacted with networked embedded devices. The primary purpose of the system was to display status of the devices on the system (a non-critical function), but there were other capabilities that could have safety implications under certain situations. As a result, specific aspects of the system had to have high availability. The functionality of the system was not overly complex compared to systems built in the past, but the scale of the application was larger than the developing company was used to for this particular class of system. The development effort was around 300 staff years, about twice the size of the next largest effort.

Unlike system 1, system 2 was a custom application being developed for a specific customer. Again this was a highly visible project (visible both within the company as well as in the general public) that represented the largest such project that this organization had done to date with the potential for a much larger order upon successful completion. There were architecturally significant requirements that were deemed important including:

- Defined startup times after which all functionality must be available
- Failover times
- Availability requirements
- Fault tolerance requirements
- Various failure modes (i.e. a "safe state")
- Minimize development costs

The availability and startup requirements were imposed by the customer and to be validated in various testing phases. As this was a fixed price project (with a bonus/malice clause) the developing organization had an incentive to minimize development costs and schedule.

System Description: The architecture team for System 2 also decided to use a component based solution. They focused on defining a set of components with an explicit set of responsibilities, interfaces, and behavior. System 2 was developed using J2EE technologies. CBSD was selected primarily to reduce development costs.

In order to achieve the availability requirements the architects made several decisions:

- *Decentralized system:* In order to minimize the impact of failures on various aspects of the system, the architects reduced the functionality that was managed centrally. Most of the logic of the system was to be replicated on many machines to handle activities of a specific location. A protocol was devised so that these local machines could continue to operate in the event that they lost communication with the central system (see figure 1).
- *Redundant components:* The central components were developed so they could be deployed redundantly.
- *Complete backup system:* The system was intended to have a complete backup system that mirrored the functionality and state or the primary system.
- *Watchdogs:* In addition there were component level watchdogs to monitor the status of important components and allow the system to attempt restarts at the component level.

Organizational Structure: For system 2 the customer, primary responsibility for system level requirements, and system engineers were in the US. The software architects were located in Western Europe and the software developers were primarily in Eastern Europe.

The groups in the US and Western Europe were in different divisions of the same operating company. Both the system and software side, however, had several outside consultants on the team. This included some key positions on the software architecture team. The group in Eastern Europe was part of a different operating company (but from the same overall organization) as the other groups.

Overall there was little history working together by the individual participants on this project. The two operating companies involved had collaborated on many past projects, but the individuals on this particular project had not worked together in the past. Likewise the personnel from the US and Western Europe had not interacted together in the past.

Frequent trips were made by key personnel in all directions and as the project progressed several of the software architects were located full time in Eastern Europe with the software developers and testers. Software teams were aligned with the various sub-systems, the UI, the database, and testing.

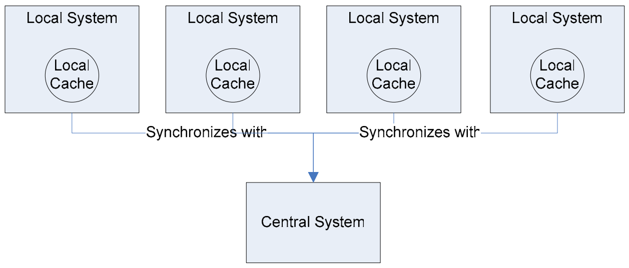

Figure 1

Issues: Ultimately the organization was not successful in implementing this architecture. The project was reorganized and a new architecture was developed. The issues that led to this failure were primarily realized as stability issues. The system was never able to become stable enough to get past system testing. Many of the issues that caused the stability problems had their roots in the complex synchronization mechanisms.

Analysis: Part of the decision to use a decentralized solution included decentralized caching of data. This included configuration data as well as application data. While this improved performance and availability of data it made synchronizing state across the various elements of the system quite complicated. Decisions around state management were largely deferred to the sub-system teams. As each team made local decisions regarding state management this was handled inconsistently across the various sub-systems.

The decentralized caching of data and state management within the sub-systems created many interdependencies across the tasks of the various sub-system teams. The overall correct operation of the system relied on correct and consistent synchronizations mechanisms which involved many aspects of the system and was largely impacted by the decisions about state management and complicated due to the local caching of data.

The organization had significant difficulty recognizing these issues and working together as a team to solve them. There was a lot of finger pointing (e.g. “that is really the responsibility of team A”) amongst teams when close cooperation was needed to resolve the issues.

Mistrust existed amongst the different organizational entities and as problems emerged various managers seemed to work hard to deflect blame. This problem seemed to be exacerbated by the disparate entities involved. With a large number of external consultants, several divisions, and management chains involved the individuals had no history of working as a team and had different motivations and incentives.

Looking at these issues with respect to our initial argument we see that there were several aspects of the architecture that required extensive coordination among disparate teams, namely how to synchronize state across sub-systems.
We also can see that because of the nature of the organization (e.g. geographic and organizational boundaries), the ability of the organization to coordinate was impeded. These issues were not explicitly recognized and planned for, however. The project managers and architects were both diligently performing their duties and even trying to account for the other’s concerns, but were unaware of the impact of these architectural decisions on the organization.

3.3 System 3

System Goals: The third system was also a platform effort. This platform allowed for the integration of a suite of related products to realize functionality needed by the target markets. The products were devices (e.g. sensors, actuators), embedded controllers, and SCADA systems. The primary goal of the platform was to enable a collection of these products to function as a single system from the end user/customers perspective. In addition the domains in which these systems were deployed had diverse needs with respect to performance, reliability, and availability as well. This platform was intended to live for 10 – 15 years and so had to accommodate future customer needs in order to remain viable in the marketplace. One such feature introduced by the market was the need to support versioning of system data.

System description: This system was organized as a loosely coupled collection of sub-systems or distinct applications (depending on your perspective). At runtime these subsystems operated largely independently (e.g. deployed on their own hardware). The interactions among these systems was built on defined (often standards based) protocols. Complex scheduling policies across subsystems was not required as most of the concurrency was within a given system. To accommodate the performance needs budgets were allocated to each subsystem. When estimated execution times were uncertain prototypes were developed to help ensure the budgets were realistic. Data management was decentralized. Each subsystem managed and stored its own data (see figure 2).

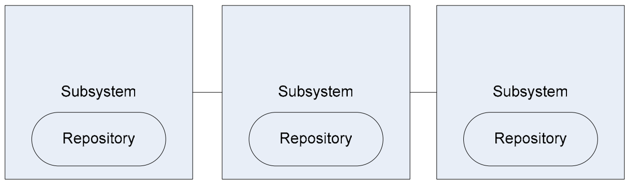

Figure 2

Organizational Structure: In this project there was an overall platform architecture team located in Europe. This team was responsible for the definition of any aspect of the architecture that required coordination of more than one of the subsystems. Each subsystem group was organized as an independent project with its own project manager, architects, developers, testers, and so forth. The integration of these subsystems, as needed to handle the total system features, was set up as a project as well with project managers and testers. Competing schedules were managed by the project manager of this project.

Each of these teams spanned organizational boundaries. Geographic boundaries existed both within a particular project and across projects. There were regular meetings of management to coordinate priorities across projects and technical had frequent interactions across projects as well. Several technical teams existed with representatives of all of the projects participating (if not all of the sites).

Issues: In many ways this was a very successful effort. These subsystems were sold individually and with great success. Likewise they were integrated and sold as a total solution. The issues arose, however, during the evolution of the system. There was a request from marketing to introduce versioning of the data into the complete system. All of the individual subsystems had versioning capabilities, but they were inconsistent and did not result in a single solution from the customer's perspective. A technical solution was devised for how to resolve the divergent versioning strategies. The organization was unable to realize this solution, however. The reasons reported were the complexities in coordinating the different groups involved. The result of this difficulty was that the initial solution was discarded and an alternative was designed with a centralized datastore (see figure 3).

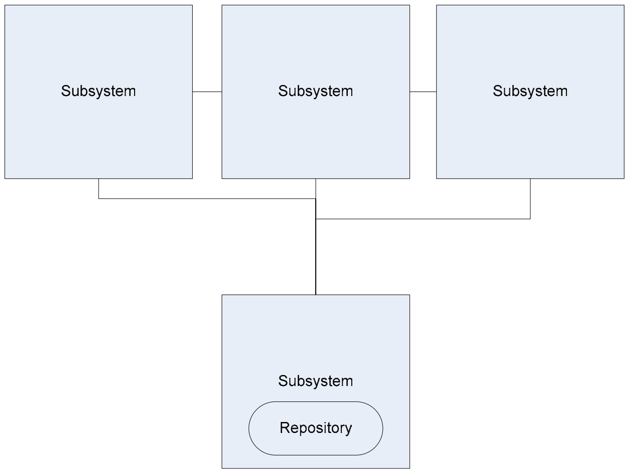

Figure 3

Analysis: In looking at the details of this situation it was in some ways different from the previous two. In this case (as in the others) there was a technical decision that required coordination amongst teams in order to realize. It was also the case in system 3 (as in the others) that the ability for the teams to coordinate was impeded. In this case, however, it was the organizational boundaries that were the primary contributors to the coordination impedance. As the subsystem groups were organizationally distinct, they had their own management structure, their own schedules, and their own priorities. The decision to decentralize the datastores meant that for the addition of a unified versioning mechanism coordination across groups was required. It became too difficult to synchronize the release schedule in such a way that business objectives could be met.

The second design, however, with the centralized datastore was well aligned with the organization. It did, however, end up requiring additional coordination in different areas. In order to meet particular performance requirements, for example, now involved the coordination of many more disparate teams.

4.0 Conclusions

In each of these three systems we see examples of architectural decisions other than invocations across modules that create significant dependencies across tasks. In system 1 it was the scheduling strategy and the strategy for managing resource consumption, in system 2 it was trying to manage state synchronization, and in system 3 it was trying to synchronize the release schedules to realize a particular customer feature. Furthermore the task interdependencies created by these architectural decisions resulted in coordination needs that could not be met by the organization. It was explicitly recognized in all three cases that invocations across modules assigned to different teams implies coordination among those teams. This recognition was not adequate, however, to ensure that the architecture was buildable by the organization.

We recognize that this need for coordination would not be an issue in all cases. Some organizational structures could accommodate these decisions without problems. If for example, there were no organizational boundaries across the subsystems in organization 3, it likely would have been simpler to manage the release schedules. There would not have been different product management teams imposing conflicting priorities on the various teams. Organization one had in fact previously been successful at implementing a similar architecture in their legacy system. It was only once they had distributed teams that coordination issues arose. In short it is not the architectural mechanisms in isolation that cause a problem; it is the combination of these mechanisms with particular factors that cause the misalignment.

It is this fact which motivates our second point. It is neither the organizational structure nor the architectural decisions in isolation that cause a problem, but rather the lack of alignment across these areas. This speaks for the need to consider both the organizational situation and the architectural mechanisms together. This is currently an issue, however, as there isn't typically one person that has insights into both areas, and there isn't a common understanding of what the relationship between these two areas of concern is. The architect(s) are concerned with the technical decisions, the project manager is concerned with the organizational factors, and there is inadequate recognition of how these two groups need to interface. All too often this results in misalignment as we have seen in these three cases often having a significant detrimental effect on the schedule, quality, or even the likelihood of project completion. Thus we feel it is important to more precisely define these areas of connection in order to allow for the architectural alignment to be explicitly considered early enough in the project that corrective action can be taken without undue expense or delay.

5.0 Future Work

In order to understand what it will take to align organizations and architectures we first need to:

- Understand precisely what architectural mechanisms imply the need for coordination
- Understand more about the organizational characteristics that impede or support coordination
- Understand what the relationship is between the particular architectural mechanisms that imply coordination and different organizational characteristics

While these three points describe a formidable, long-term research agenda, we will start by collecting several kinds of data from executing projects. First we need to have data regarding the kinds of technical dependencies that exist in the architecture. We need to know about the allocation of tasks to teams, we need to have an understanding of the kind of coordination that occurred across teams, and we need to have an understanding of what the results were (e.g. quality issues, successful realization of the solution, delays, and so forth). We can then analyze this data to determine the precise nature of the coordination required by varioius architectural mechanisms, and which organization factors facilitated or inhibited coordination.

In the long run the vision is to have some means to be able to analyze the coordination capability of a particular organization, evaluate the coordination needs of a particular design, determine the relative "fit" between the two, and have tactics that can be made to the design, the organization, or both in the event of significant mismatch.

6.0 Acknowledgements

The authors gratefully acknowledge support by NSF grant IIS-0534656, as well as support from the Software Industry Center and its sponsors, particularly the Alfred P. Sloan Foundation. The authors would also like to thank the Software Engineering Institute, the US Department of Defense, and Siemens Corporate Research for their sponsorship and support.

12. References

[1] Kraut, R. & Streeter, L. (1995). Coordination in large scale software development. *Communications of the ACM*, *38*(3), 69-81.

[2] B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," Comm. ACM, vol. 31, no. 11, pp. 1268-1287, 1988.

[3] M.E. Conway, "How Do Committees Invent?" *Datamation,* Vol. 14, No. 4, 1968, pp. 28-31.

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

[5] Henderson, R.M. and Clark, K.B. Architectural innovation: the reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quar-terly*, 35, 1 (Mar. 1990), 9-30.

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

[7] Herbsleb, J.D. and Grinter, R.E. Splitting the organization and integrating the code: Conway's law revisited. *In Proceedings of the International Conference in Software Engineering (ICSE '99)*, Los Angeles, 2004, 85-95.

[8] Clements, P and Northrop, L, Software Product Lines: Practice and Patterns. *Addison-Wesley,* 2001

[9] De Souza, C.R.B., Redmiles, D., Cheng, L.T., Millen, D., and Patterson, J. Sometimes You Need to See Through Walls – A Field Study of Application Programming Interfaces. *Computer Supported Cooperative Work,* 2004, Chicago, Illinois

[10] Grinter, R.E., Herbsleb, J.D., Perry, D.E. The Geography of Coordination: Dealing with Distance in R&D Work *Group 99,* 1999 Phoenix, AZ

[11] Shaw, M and Garlan, D., *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996

[12] Mehta, N.R., Medvidovic, N., and Phadke, S.. "Towards a Taxonomy of Software Connectors." In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, pages 178-187, Limerick, Ireland, June 4-11, 2000

[13] Ovaska, P., Rossi, M., and Marttiin, P.,"Architecture as a coordination tool in multisite software development" In *Software process improvement and practice* pages 243-247, John Wiley & Sons Ltd, Oct/Dec 2003