

**DEPENDENCIES IN GEOGRAPHICALLY DISTRIBUTED
SOFTWARE DEVELOPMENT:
OVERCOMING THE LIMITS OF MODULARITY¹**

Marcelo Cataldo

--- Dissertation Summary ---

School of Computer Science
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee

Kathleen M. Carley, Co-Chair

James D. Herbsleb, Co-Chair

Len J. Bass

David Redmiles

Copyright © 2007 Marcelo Cataldo

¹ This dissertation was funded in part by National Science Foundation under Grants No. IIS-0414698, IIS-0534656 and IGERT 9972762.

INTRODUCTION

Over the past couple of decades, geographically distributed work has become pervasive and software development organizations are no exception. Factors such as access to talent, acquisitions and the need to reduce the time-to-market of new products are the driving forces for the increasing number of geographically distributed software development (GDSD) projects (Herbsleb & Moitra, 2001; Karolak, 1998). Unfortunately, this new trend has its costs. Distance leads to numerous problems in communication and coordination, and ultimately, impacts the performance of software development teams (Herbsleb et al, 2000; Herbsleb & Mockus, 2003). The failure to identify work dependencies among developers or development teams results in coordination problems. A growing body of work on coordination in software development suggests that the identification and the management of dependencies is a fundamental challenge in software development organizations, particularly in those that are geographically distributed (some examples are: Cataldo et al, 2007; de Souza, 2005; Grinter et al, 1999; Herbsleb et al, 2000; Herbsleb & Mockus, 2003). The modular product design and organizational theory research streams could inform the design of software development organizations so they are better able to identify and manage work dependencies. However, we first need to understand the assumptions of the different theoretical views and how those assumptions relate to the characteristics of software development tasks.

The Nature of Software Development and Modular Design

The idea of dividing a complex task into smaller manageable units is consistent with the reductionist view (Simon, 1962; von Hippel, 1990) which is well developed in the product development literature (Eppinger et al, 1994). Projects, typically, have a general description of the system's components and their relationships or a more detailed report such as architectural or high-level design document. Managers use the information in those documents to divide the development effort into work items that are assigned to specific development teams minimizing the interdependencies among those teams (Conway, 1968; Eppinger et al, 1994; Sullivan et al, 2001). The modularization theoretical perspective has received attention in three streams of research: the system and

product design literature (e.g. Conway, 1968; Eppinger et al, 1994), the strategic management literature (e.g. Baldwin & Clark, 2000) and the software engineering literature (e.g. Parnas, 1972; Sullivan et al, 2001). All three theoretical perspectives argue that reducing the technical interdependencies among modules, task interdependencies are reduced, which consequently, reduces the need for communication among work groups. Unfortunately, there are several problems with these assumptions in the context of software development. First, existing software modularization approaches only use a subset of the technical dependencies, typically syntactic relationships, of a software system (Garcia et al, 2007). Then, potentially relevant work dependencies might be ignored. Secondly, recent empirical evidence indicates that the relationship between product structure and task structure is not as simple as previously assumed. Moreover, the theorized similarity between product and task structures diminishes over time (Cataldo et al, 2006). Thirdly, promoting minimal communication between teams responsible for interdependent modules is problematic. Recent studies suggest that minimal communication between teams, collocated or distributed, is detrimental to the success of projects (de Souza et al, 2004; Grinter et al, 1999; Yassine et al, 2003). Yet another problem associated with the assumptions of modular design is the nature and stability of the interfaces between software modules. De Souza (2005) encountered that interfaces tended to change often and their design details tended to be incomplete, leading to serious integration problems. Moreover, Cataldo and colleagues (2007) presented case studies where even simple interfaces between modules developed by remote teams create coordination breakdown and integration problems. These findings argue that the interfaces between software modules might differ in complexity and, often, it is not possible to specify those interfaces at the necessary level of detail, increasing the likelihood of future changes to them. This lack of stability represents a constant challenge for software development organizations.

In sum, the modularization approach is a very useful tool for dividing the development of a complex software system into manageable units. However, modularization is not a sufficient representation of work dependencies in software development activities. The relationship between the task dependency structure and the product structure is not as simple as theorized. Appropriate mechanisms are then required

to identify relevant work dependencies and, consequently, maintain suitable levels of communication and coordination among teams developing interdependent modules, particularly, in the case of geographically distributed software development.

The Nature of Software Development and Interdependency Theories

The organizational theory literature has proposed numerous mechanisms for handling interdependencies among tasks (e.g. Crowston, 1991; Galbraith, 1973; March and Simon, 1958; Thompson, 1967). However, those perspectives are appropriate for stable and enduring task structures because they rely on the assumptions of determinism and stability (Staudenmayer, 1997). Unfortunately, there are several characteristics of software development activities that limit the applicability of traditional organizational theories as well as the more recent computational and mathematical organizational theory work. First, it is widely accepted among software engineering researchers and practitioners that the requirements of the system become known over time or those requirements change as time progresses (Leffingwell & Widrig, 2003). In some cases the changes in the requirements result in minor alterations of specific development tasks. In other cases, new features have to be added or features under development are eliminated. These events introduce a certain level dynamism in software development that challenges the determinism and stability assumptions of the information processing views of interdependency.

Secondly, the dynamic nature of finer-grain dependencies that arise as part of the development of a piece of code is not well suited for traditional organizational theories of coordination. The act of developing a software system consists of a collection of design decisions, either at the architectural level or at the implementation level. Those design decisions introduce constraints that might establish new dependencies among the various parts of the system, modify existing ones or even eliminate dependencies. The changes in dependencies can generate new coordination requirements that are quite difficult to identify a priori, particularly when they are not obvious, or as a project matures over time (Henderson & Clark, 1990; Sosa et al, 2004). Failure to discover the changes in coordination needs might have a profound impact on the quality of the product (Curtis et al, 1988) and on productivity (Herbsleb & Mockus, 2003). In addition, little is known

about the specific impact of the various types of dependencies that arise among parts of a software system such as explicit versus implicit dependencies or syntactic versus logical dependencies. Then, the use of the computational and mathematical organizational theory approaches is limited because of the lack of theoretical framework that guides the modeling of the relationships between the organizational tasks, their dependencies and the need to communicate and coordination.

In sum, software development tasks are embedded in an evolving network of coordination requirements that need to be satisfied. The coarse-grain and idealized approaches suggested by the organization theory literature are not appropriate to identify and manage such a dynamic web of interdependencies. A finer-grain view of coordination would provide a better framework in dynamic knowledge-intensive tasks such as software development.

Research Questions

The limitations of the current mechanisms for identifying and managing dependencies in geographically distributed software development organizations discussed in the previous sections led to the following general research questions addressed by this dissertation:

RQ 1: How relevant task dependencies can be identified from technical dependencies?

RQ 2: What is the impact of those task dependencies on traditional outcome variables such as productivity and quality?

A FRAMEWORK FOR IDENTIFICATION OF WORK DEPENDENCIES

The Concept of Socio-Technical Congruence

The concept of socio-technical congruence proposed in this dissertation builds on the idea of “fit” from the organizational theory literature (Burton & Obel, 1998; Carley & Ren, 2001; Levchuck et al, 2004) and from a mathematical stand point builds on the

meta-matrix model from the dynamic network analysis literature (Carley, 2002; Krackhardt & Carley, 1998). Combining those two lines of research allows for two important contributions to the literature. First, the socio-technical congruence framework presented here provides a fine-grain level of analysis. Secondly, the measure facilitates assessing the role of coordination activities in multiple and complementary ways as well as examining the impact of several types of dependencies.

Formally, socio-technical congruence is defined as the match between the coordination requirements established by the dependencies among tasks and the actual coordination activities carried out by the workers. In other words, the concept of congruence has two components, coordination needs and coordination activities, and the following sections discuss the mathematical framework to measure them.

Identification of Coordination Requirements

In order to identify which set of individuals should be coordinating their activities, we need to represent two sets of relationships. One set is represented by which individuals are working on which tasks. The relationships or dependencies among tasks represent the second element. Assignments of individuals to particular work items is represented by a people X task matrix where a one in cell ij indicates that worker i is assigned to task j . I will refer to this matrix as *Task Assignments* (T_A). Following the same approach, the set of dependencies among tasks can be represented as a square matrix where a cell ij (or cell ji) indicates that task i and task j are interdependent. I will refer to this matrix as *Task Dependencies* (T_D). Now, if the *Task Assignment* and *Task Dependencies* matrices are multiplied, a people by task matrix is obtained that represents the set of tasks a particular worker should be aware of, given the work items the person is responsible for and the dependencies of those work items with other tasks. Finally, a representation of the coordination requirements among the different workers is obtained by multiplying the product of the *Task Assignment* and *Task Dependencies* matrices by the transpose of the *Task Assignment* matrix. This product results in a people by people matrix where a cell ij (or cell ji) indicates the extent to which person i works on tasks that share dependencies with the tasks worked on by person j . In other words, the resulting matrix represents the *Coordination Requirements* or the extent to which each pair of

people needs to coordinate their work. Formally, the *Coordination Requirements* matrix is determined by the following product:

$$\mathbf{C}_R = \mathbf{T}_A * \mathbf{T}_D * \mathbf{T}_A^T \quad \text{(Equation 1)}$$

where, \mathbf{T}_A is the Task Assignments matrix, \mathbf{T}_D is the Task Dependencies matrix and \mathbf{T}_A^T is the transpose of the Task Assignments matrix.

Measuring Socio-Technical Congruence

Given a particular *Coordination Requirements* matrix constructed from relating product dependencies to work dependencies, we can compare it to an *Actual Coordination* (\mathbf{C}_A) matrix that represents the interactions workers engaged in through different means of coordination. I refer to the match between those to matrices as socio-technical congruence. Then, given a particular set of dependencies among tasks, congruence is the proportion of coordination activities that actually occurred (given by the *Actual Coordination* matrix) relative to the total number of coordination activities that should have taken place (given by the *Coordination Requirements* matrix). For example, if the *Coordination Requirements* matrix shows that 10 pairs should coordinate, and of these, 5 show *Actual Coordination* interactions, then the congruence is 0.5. Formally, we define congruence as follows:

$$\begin{aligned} \text{Diff}(\mathbf{C}_R, \mathbf{C}_A) &= \text{card} \{ \text{diff}_{ij} \mid cr_{ij} > 0 \ \& \ ca_{ij} > 0 \} \\ |\mathbf{C}_R| &= \text{card} \{ cr_{ij} > 0 \} \end{aligned}$$

We have,

$$\text{Congruence}(\mathbf{C}_R, \mathbf{C}_A) = \text{Diff}(\mathbf{C}_R, \mathbf{C}_A) / |\mathbf{C}_R| \quad \text{(Equation 2)}$$

The value of congruence belongs to the [0,1] interval that represents the proportion of coordination requirements that were satisfied through some type of coordination activity or mechanism. The measure of socio-technical congruence proposed here provides a new way of thinking about coordination, particularly, by providing a fine-

grain level of analysis of different types of product dependencies and allowing us to examine how coordination needs are impacted by them.

EMPIRICAL STUDIES

Study I: Congruence and Development Productivity

Identifying work dependencies and determining the appropriate coordination mechanisms to address the dependencies is not a trivial problem in geographically distributed software development. Numerous types of work, for instance non-routine knowledge-intensive activities such as software development, are potentially full of fine-grain dependencies that might change on a daily or hourly basis. Conventional coordination mechanisms like standard operating procedures or routines would have very limited applicability in these dynamic contexts. Failure to identify the new needs for coordination and information exchange might hinder the organization's ability to adapt to changes in their competitive environment (Henderson & Clark, 1990). This study represents the first step in the examination of how the gaps between coordination needs and actual coordination activity impact productivity in the context of geographically distributed software development activities. The analysis presented in this study also explores the dynamic evolution of coordination requirements.

Research Setting

Data from a large geographically distributed development effort were used to examine the stability of coordination requirements over time and the relationship between congruence and resolution time of modification requests. A total of 2375 multi-team modification requests were identified. Those modification requests belonged to the first four releases of the product in which 114 developers worked fulltime.

Key Results

This study evaluated a measure coordination that extends traditional conceptualizations of coordination by taking a fine-grain level of analysis to better examine the mismatches between dependencies and coordination activities. Those gaps could have major implications for the productivity and the quality of the output of product development organizations (Curtis et al, 1988; Espinosa, 2002; Herbsleb & Mockus, 2003; Sosa et al, 2004). The results suggested that the congruence framework provides a useful mechanism to examine how coordination needs that are not satisfied impact software development productivity. When the developers coordinate their task with the relevant set of workers, development productivity increases. I also addressed the dynamic nature of dependencies that exist in complex tasks such as software development. Individuals have difficulties identifying task interdependencies that are not obvious or explicit (Sosa et al, 2004) and the developers' ability to recognize dependencies diminish as coordination requirements change over time (Henderson & Clark, 1990). For these reasons, volatility in the coordination requirements represents a major hurdle for work groups and, particularly, for those that are geographically distributed. Collaborative tools could play an important role in reducing the gap between recognized and actual interdependencies. It would be highly desirable for future tools to be able to assess the characteristics of the task and assist the users in identifying and dealing with dependencies unknown a priori or that emerged as a consequence of the evolving characteristics of tasks. The congruence measure provides a framework for those future tools.

The results showed the product structure-task structure relationship is not as simple as theorized. Modularization techniques in software development only consider one type of technical dependencies, syntactic relationships (Garcia et al, 2007). That limitation manifested clearly in the results. The empirical evaluation of the congruence framework showed the importance of understanding the dynamic nature of software development which is not necessarily captured by the traditional way of thinking about technical dependencies in software: syntactic relationships. The analysis indicated that logical dependencies provide a more accurate representation of the most relevant technical dependencies in software development projects.

Study II: The Structure of Dependencies, Congruence and Product Quality

Customer reported software faults are, arguably, caused by violation of dependencies that are not recognized by the developers implementing a software system. Those dependencies could stem from various sources such as technical properties of the system under development and how the development work is organized. The software engineering literature suggests several types of technical dependencies. One form of software dependencies are syntactic relationships among modules of a system that are reflected in the code by the definition and use of functions, methods, variables and other programming language constructs. This line of work found that higher levels of coupling are related to higher levels of failure proneness of a software system. However, syntactic dependencies are only one approach for representing the structure of a software system. In more recent work in the software evolution literature, Gall and colleagues (1998) examined the evolution of changes to modules to identify logical dependencies. The approach attempts to uncover dependencies among modules that are not explicitly identified by traditional syntactic approaches. However, no prior study has examined the combined impact of logical and other types of technical dependencies on the failure proneness of a software system. Hence, further study of this relationship is required.

Human and organizational factors may also affect the quality of a software system. The level of interdependency between tasks tends to drive communication and coordination among workers (Galbraith, 1973; von Hippel, 1990). However, recent studies of coordination in software development suggest that the identification and management of technical dependencies is a challenge in software development organizations, particularly, when those dependencies are semantic rather than syntactic (e.g. Cataldo et al, 2007; de Souza, 2005; Grinter et al, 1999). Then, appropriate levels of communication and coordination may not occur, potentially decreasing the quality of a system (Curtis et al, 1988; Herbsleb et al, 2006). Consequently, it is important to understand how work dependencies and the coordination behavior of developers impact the failure proneness of a system.

This goal of this study is three-fold. First, I examine how syntactic and logical dependencies relate to a software system's failure proneness. Secondly, I incorporate in the analysis of failure proneness the role of work development. Thirdly, the developers' ability to coordinate their work congruently with regards to the coordination needs is considered. In sum, I examine how work-related factors affect the quality of software system above and beyond the technical dependencies among the various parts of that software system.

Research Setting

Data from two distinct large development projects from two unrelated companies were used in this study to examine the relationship between dependencies, congruence and failure proneness. I build a dataset containing 8,257 modification requests covering the first four releases of the product from the project used in study I. The second project was a large embedded system for a telecommunications device. In this case, the dataset contained 7000 modification requests associated with the last six releases of the product.

Key Results

The study reported in this chapter has several important contributions to the software engineering literature. First, the study examined the impact that syntactic, logical and work dependencies have, simultaneously, on the failure proneness of a software system. All three types of dependencies are relevant and their effect is complementary suggesting their independent and important role in the development process. Consistent with past results, the analysis showed that source code files with higher number of syntactic dependencies were more prone to failure. More importantly, the results also showed that source code files with higher number of logical dependencies are more likely to exhibit field defects. In addition, this study is the first analysis that highlights the importance of the structure of the logical relationships. The results showed that software modules with logical dependencies to other highly interconnected files were less likely to exhibit customer-reported defects. Then, this finding suggests a new view of product dependencies with significant implications regarding how we think about modularizing the system and how development work is organized. The effect of the

structure of the network of product dependencies elevates the idea of modularity in a system to the level of “clusters” of source code files. Then, those highly inter-related sets of files become the relevant unit to consider when development tasks and responsibilities are assigned to organizational groups.

A second significant contribution of the study is the recognition and the assessment of the impact the engineers’ social network has on the software development process. The results showed that individuals that exhibited a higher number of workflow dependencies and coordination requirements were more likely to introduce defects in the files they worked on. These findings suggest the potentially detrimental effect of the additional effort on the part of a developer that needs to receive work from or coordinate with multiple people and manage those relationships accordingly in order to perform the tasks.

Finally, the study has two additional important contributions. The empirical analysis was replicated across two distinct projects from two unrelated companies obtaining consistent results. Then, this study exhibits strong external validity, a factor typically neglected in the software engineering literature. In addition, the statistical models proposed in this chapter showed significantly higher level of predictive power than recent proposed models of failure proneness (Nagappan & Ball, 2007) that focused on the role of traditional factors such as syntactic dependencies and churn metrics.

Study III: The Evolution of Coordination Behavior

The limitations of the modular design approach discussed in the introduction could be overcome to some extent by promoting and fostering appropriate communication and coordination among the appropriate set of formal teams. Organizational and geographic barriers to communication could be overcome by individuals in key roles who facilitate and promote the interaction between teams (Allen, 1977; Ancona & Caldwell, 1992; Hauschildt & Schewe, 2000). Several definitions of those key positions have been proposed in the product development literature (Hauschildt & Schewe, 2000). Examples are “alliance champion”, “external liaison”, “gatekeeper”,

and “process promoter”. Although those definitions differ slightly from each other in their theoretical underpinnings, the overarching theme is that those individuals perform a different type of activity than the rest of the members of a R&D group and their task is critical for the success of a project. Those key people have access to different sources of information and they are capable of synthesizing the information in a way useful for the various groups so they can to better perform their development activities (Hauschildt & Schewe, 2000).

The use of “liaison” or “gatekeepers” to manage the dependencies between teams has also been proposed as a mechanism for facilitating coordination in geographically distributed software development (Sangwan et al, 2006). As engineers perform their development tasks, critical information and knowledge about the parts of the system involved in the tasks at hand is exchanged. As software development tasks change over time, developers get the opportunity to gain access to new information and knowledge about the technical properties of different parts of the system. This system of social relationships, which I will refer to as a *Coordination Network*, is an evolving entity. If gatekeepers are strategically embedded in the coordination networks, they can acquire the necessary knowledge to discover the relevant technical and task dependencies.

In this study, I present a longitudinal examination of how coordination networks evolve using data collected from a large geographically distributed software development project. In addition, data a second geographically distributed project was used to replicate a portion of the statistical analysis that explored the relationship between coordination patterns and individual-level productivity.

Research Setting

The coordination-related data associated the dataset from study I were used to examine the evolution of coordination patterns among developers. Given the particular patterns of communication tools usage in the project, I focused on analyzing data from the two primary means of communication and coordination: the online-chat system (IRC) and the MR tracking system.

Key Results

This study presented a longitudinal analysis of coordination activities in a geographically distributed software development project. The results showed that developers that are positioned centrally in the social system of information exchanges and coordination perform a critical bridging activity across formal teams and geographical locations. These findings are consistent with past research highlighting the critical role “liaisons” individuals play in the performance of teams and projects (Ancona & Caldwell, 1992; Hauschildt & Schewe, 2000). However, the analysis also revealed those same individuals contributed the most to the development effort. More interestingly, in the research setting, the “liaisons” emerged over time from each development group, contrary to view typically discussed in the literature where these key roles are formally established (Ancona & Caldwell, 1992; Hauschildt & Schewe, 2000, Sangwan et al, 2006).

The analysis also showed that the patterns of coordination were relatively stable. In fact, the stability of the coordination patterns increased over time on both means of communication, MR and IRC, used by the development organization. These findings are consistent with past research indicating once established patterns of communication and coordination resist change (Henderson & Clark, 1990). Moreover, Henderson and Clark (1990) argued that the stability of the communication paths could be detrimental to the development organization because those communication conduits might not be the appropriate ones when the product structure changes. This line of research highlights the potential detrimental effects of the stability of patterns of communication and coordination. On the other hand, stable patterns of communication and coordination with particular structure might have benefits in relation to changes in the product structure. For instance, the core-periphery structure found in our research setting involved members of all development teams. One could argue that such structure could act as a “council” where relevant pieces of information from all parts of the system are shared and understood. Then, the changes in dependencies introduced by modifications in the product could be more easily identified. Recognizing the changes in dependencies might be significantly more challenging in other types of structures such as hierarchies.

CONTRIBUTIONS AND LIMITATIONS

Contributions

This dissertation has important theoretical and empirical contributions to the software engineering, CSCW and organizational literatures. In terms of theoretical contributions, this dissertation presented a fine-grain view of coordination that addresses the limitations of traditional approaches from the organizational theory literature. The proposed framework for measuring socio-technical congruence provides the necessary machinery to examine the consequences of coordination requirements that are not satisfied. In addition, the congruence framework provides the sufficient flexibility to consider multiple types of product dependencies and their implications on the work dependencies encountered by product development organizations.

This dissertation also has significant empirical contributions. First, the empirical evaluation of the congruence framework showed the importance of understanding the dynamic nature of software development. Identifying the “right” set of product dependencies that determine the relevant work dependencies and coordinating accordingly has significant impact on reducing the resolution time of modification requests. The analyses showed traditional software dependencies, such as syntactic relationships, tend to capture a relatively stable view of product dependencies that is not representative of the dynamism in product dependencies that emerges as software systems are implemented. On the other hand, logical dependencies provide a more accurate representation of the most relevant product dependencies in software development projects. The statistical analyses showed that when developers’ coordination patterns are congruent with their coordination needs, the resolution time of modification requests was, on average, reduced by 32% when considering the collective effect of all four measures of congruence. Generalizing, the empirical examination of the congruence framework and coordination patterns showed the tight relationship between team design, coordination and performance providing an important contribution to the organizational literature.

Secondly, this dissertation moves forward our understanding of the relationship between product and work dependencies and software quality. Study II showed that logical dependencies among software modules and work dependencies are two of the most relevant factors affecting the failure proneness of software modules. For instance, the statistical analyses indicated that a unit increase in logical dependencies increased twice as much the likely of failure relative to the impact of syntactic dependencies. In addition, the proposed statistical models that included the different types of technical and work dependencies exhibit significantly better predictive power than recent models (e.g. Nagappan & Ball, 2007) that consider traditional factors such syntactic dependencies and churn metrics.

Finally, study III presented a longitudinal analysis of coordination activities in a geographically distributed software development project. The results showed that approximately 20% of the developers were positioned centrally in the social system of information exchanges and coordination activities performing a critical bridging function across formal teams and geographical locations. In addition, those same individuals contributed the between 50% and 65% to the development effort in terms of implementing the software system in each released covered by the data. The analysis also revealed that the patterns of coordination become stable over time, and those patterns were only partially driven by the coordination requirements of the development tasks.

Limitations

It is also important to highlight some of the limitations of the work reported in this dissertation. First, the measures proposed as part of the congruence framework are contingent on assumptions about the software development processes used in the development organization as well as usage patterns of tools that assist the development effort such as defect tracking and version control systems. One key assumption is the possibility to identify (1) the set of source code files that were changed as part of a modification request and (2) the developers that made those changes. For instance, a policy of source code file ownership by particular developers could potentially bias the congruence measures. Developers that own a particular source code might appear as participants in the development effort associated with a modification request, however,

that might not be the case. In other cases, such as open source projects, the nature of the work in certain project is such that the information about which files changed together as part of a modification request is almost impossible to reconstruct in a reliable way.

The alternative approach of computing coordination requirements based on syntactic relationships also has its limitations. The method relies on tools that can reliably extract the dependency information among software modules for a specific programming language. More importantly, projects that use multiple programming languages will represent a challenge, particularly, in terms of determining syntactic dependencies that involve modules written in different programming languages.

Another limitation of the work presented in this dissertation is a potential concern for external validity of some of the empirical analyses. For instance, study I examined only one system with particular properties that might be conducive to support the results found by the analysis. However, the processes and tools used by the development organization are commonplace in the software industry. Moreover, the general technical characteristics of the system are similar to other types of distributed systems developed into products in the software industry. Hence, I think the results are generalizable, particularly, in the context of development organizations responsible for delivering complex software systems.

REFERENCES

- Allen, T.J. (1977). *Managing the Flow of Technology*. MIT Press.
- Ancona, D.J. and Calwell, D.F. (1992). Bridging the boundary: external activity and performance in organizational teams. *Administrative Science Quarterly*, Vol. 37.
- Baldwin, C.Y. and Clark, K.B. (2000). *Design Rules: The Power of Modularity*. MIT Press.
- Burton, R.M. and Obel, B. *Strategic Organizational Diagnosis and Design*. Kluwer Academic Publishers, Norwell, MA, 1998.
- Carley, K.M. (2002). Smart Agents and Organizations of the Future. In *Handbook of New Media*. Edited by Lievrouw, L. and Livingstone, S., Sage, Thousand Oaks, CA.
- Carley, K.M and Ren, Y. Tradeoffs between Performance and Adaptability for C³I Architectures. In *Proceedings of the 6th International Command and Control Research and Technology Symposium*, Annapolis, Maryland, 2001.
- Cataldo, M., Wagstrom, P, Herbsleb, J.D. and Carley, K.M (2006). 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, Canada.
- Cataldo, M., Bass, M, Herbsleb, J.D. and Bass, L (2007). On Coordination Mechanism in Global Software Development. In *Proceedings of the International Conference on Global Software Engineering*, Munich, Germany.
- Conway, M.E. (1968). How do committees invent? *Datamation*, Vol. 14, No. 5, 28-31.
- Crowston, K.C. (1991). *Toward a Coordination Cookbook: Recipes for Multi-Agent Action*. Ph.D. Dissertation, Sloan School of Management, MIT.
- Curtis, B., Kransner, H. and Iscoe, N. (1988). A field study of software design process for large systems. *Communications of ACM*, Vol. 31, No. 11, pp. 1268-1287.
- de Souza, C.R.B. (2005). *On the Relationship between Software Dependencies and Coordination: Field Studies and Tool Support*. Ph.D. dissertation, Donald Bren School of Information and Computer Sciences, University of California, Irvine.
- de Souza, C.R.B., Redmiles, D., Cheng, L., Millen, D. and Patterson, J. (2004). How a Good Software Practice Thwarts Collaboration – The multiple roles of APIs in Software Development. In *Proceedings of the 12th Conference on Foundations of Software Engineering (FSE '04)*, Newport Beach, CA, 221-230.
- Eppinger, S.D., Whitney, D.E., Smith, R.P. and Gebala, D.A. (1994). A Model-Based Method for Organizing Tasks in Product Development. *Research in Engineering Design*, Vol. 6, pp. 1-13.
- Espinosa, J.A. (2002). *Shared Mental Models and Coordination in Large-Scale, Distributed Software Development*. Unpublished Ph.D. Dissertation, Graduate School of Industrial Administration, Carnegie Mellon University.
- Galbraith, J.R. (1973) *Designing Complex Organizations*. Addison-Wesley Publishing.
- Garcia, A., et al. (2007). Assessment of Contemporary Modularization Techniques, ACOM'07 Workshop Report. *ACM SIGSOFT Software Engineering Notes*, Vol. 35, No. 5, pp. 31-37.
- Grinter, R.E., Herbsleb, J.D. and Perry, D.E. (1999). The Geography of Coordination Dealing with Distance in R&D Work. In *Proceedings of the Conference on Supporting Group Work (GROUP'99)*, Phoenix, Arizona.
- Hauschildt, J. and Schewe, G. (2000). Gatekeeper and process promoter: key persons in agile and innovative organizations. *International Journal of Agile Management Systems*, Vol. 2, pp. 96-103
- Henderson, R.M. and Clarck, K.B. (1990). Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms. *Administrative Science Quarterly*, Vol. 35, pp. 9-30.
- Herbsleb, J.D. and Mockus, A. (2003). An Empirical Study of Speed and Communication in Globally Distributed Software Development. *IEEE Transactions on Software Engineering*, Vol. 29, No. 6, pp.
- Herbsleb, J.D. and Moitra, D. (2001). Global Software Development. *IEEE Software*, March/April, pp. 16-20.
- Herbsleb, J.D., Mockus, A. and Roberts, J.A. (2006). Collaboration in Software Engineering Projects: A Theory of Coordination. In *Proceedings of the International Conference on Information Systems (ICIS '06)*, Milwaukee, Wisconsin.

- Herbsleb, J.D., Mockus, A., Finholt, T.A., and Grinter, R.E. (2000). Distance, Dependencies, and Delay in a Global Collaboration. *In Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'00)*, Philadelphia, Pennsylvania.
- Karolak, D.W. (1998). *Global Software Development: Managing Virtual Teams and Environments*, IEEE Computer Society.
- Krackhardt, D. and Carley, K.M. (1998). A PCANS Model of Structure in Organization. *In Proceedings of the 1998 International Symposium on Command and Control Research and Technology*, pp.113-119.
- Leffingwell, D. and Widrig, D. (2003). *Managing Software Requirements: A Use Case Approach, 2nd Edition*. Addison-Wesley.
- Levchuk, G.M. et al. (2004). Normative Design of Project-Based Organizations – Part III: Modeling Congruent, Robust and Adaptive Organizations. *IEEE Trans. on Systems, Man & Cybernetics*, Vol. 34, No. 3, pp. 337-350.
- March, J.G and Simon, H.A. (1958). *Organizations*. Wiley, New York, NY.
- Nagappan, N and Ball, T (2007). Explaining Failures Using Software Dependencies and Churn Metrics. *In Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement*, Madrid, Spain.
- Parnas, D.L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of ACM*, Vol. 15, No. 12, 1053-1058.
- Sangwan, R. et al. (2006). *Global Software Development Handbook*, Auerbach Publishers.
- Simon, H.A. (1962). The Architecture of Complexity. *In Proceedings of the American Philosophical Society*, Vol. 106, No. 6, pp. 467-482.
- Sosa, M.E., Eppinger, S.D., and Rowles, C.M. (2004). The Misalignment of Product Architecture and Organizational Structure in Complex Product Development. *Management Science*, Vol. 50, No. 12, pp. 1674-1689
- Staudenmayer, N. (1997). *Managing Multiple interdependencies in Large Scale Software Development Projects*. Unpublished Ph.D. Dissertation, Sloan School of Management, Massachusetts Institute of Technology,
- Sullivan, K.J., Griswold, W.G., Cai, Y, and Hallen, B. (2001). The Structure and Value of Modularity in Software Design. *In Proceedings of the International Conference on Foundations of Software Engineering (FSE '01)*, Vienna, Austria, 99-108.
- Thompson, J.D. (1967). *Organizations in Action: Social Science Bases of Administrative Theory*. McGraw-Hill, New York, NY.
- Von Hippel, E. (1990). Task Partitioning: An Innovation Process Variable. *Research Policy*, Vol. 19, pp. 407-418.
- Yassine, A., Joglekar, N., Braha, D., Eppinger, S. And Whitney, D. (2003). Information Hiding in Product Development: The Design Churn Effect. *Research in Engineering Design*, Vol. 14, pp. 145-161.