Association for Information Systems
AIS Electronic Library (AISeL)

ICIS 2007 Proceedings

International Conference on Information Systems (ICIS)

December 2007

Design Architecture, Developer Networks and Performance of Open Source Software Projects

Xiang Liu
Boston University

Bala Iyer
Babson College

DESIGN ARCHITECTURE, DEVELOPER NETWORKS, AND PERFORMANCE OF OPEN SOURCE SOFTWARE PROJECTS

Xiang (Michelle) Liu
Boston University School of Management
595 Commonwealth Ave., #610
Boston, MA 02215
xiangl@bu.edu

Bala Iyer
Babson College TOIM Division
Babson Hall, Room 211A
Babson Park, MA 02457
bala.r.iyer@gmail.com

Abstract

In this study we seek to understand the factors differentiating successful from unsuccessful software projects. This article develops and tests a model measuring the impact on software project performance of (1) software products' design architectures and (2) developers' positions within collaborative networks. Two indicators of project success are used: product quality and project velocity. Two dimensions of design architecture – degree of decomposition and coupling – and one characteristic of developer network structures – degree centrality– are investigated for their impact on project performance. Using data gathered from SourceForge.net and its monthly dumps, we empirically test hypotheses on the top 100 projects according to project rankings. These rankings are generated from the traffic, communication, and development statistics collected for each project hosted on SourceForge.net. Besides the top 100 projects, we also randomly choose another 100 projects to form the data sample. The main findings are that (1) the degree of decomposition has an inverted U-shaped relationship with project performance, (2) when tested on the sample of top 100 projects, average degree centrality of a project team has a positive and significant effect on project performance and (3) the effects of network metrics on some of project performances are positive in the top 100 projects and some of them are negative in the random 100 projects. We employ some control variables such as developer work experiences and sponsorship, to provide insight into the direction of the effect of degree centrality on project success. Theoretical and practical implications are discussed, and several future research directions are outlined.

Keywords: Architecture, software design, modularity, developer networks, centrality, success

Introduction

The last several decades have witnessed a steady trend toward the globalization of software-intensive high-technology businesses. Distributed software development has emerged as a fact of IS departments in organizations. One conspicuous difference between distributed development and traditional one is that the locus is no longer within the boundaries of a single organization, but at the nexus of relationships between a variety of parties. This change is profoundly impacting not only marketing and distribution but also the way products are conceived, designed, constructed, tested, and delivered to customers (O'Hara-Devereaux and Johansen 1994). Open Source Software (OSS) projects, usually characterized by distributed teams of developers and free source codes available and modifiable by end users, are one typical instance of such kind of software development.

However, on the other hand, highly distributed software development poses more challenges to both researchers and software developers. First, communication issues in large software engineering projects—especially in distributed development environment across heterogeneous sites, cultures, or time zones—can substantially slow development speed (Brooks 1975). Second, in the case of software development, coordination among project developers such as design reviews or code changes through mailing lists requires the capability of effective knowledge sharing and integration, which were identified as key factors hindering project outcomes (Walz et al. 1993). Therefore, one of the goals of this study is to better understand and explain the factors that enable organizations to achieve successful software project performance.

Through two theoretical lenses—design theory and network theory—the study focuses on understanding how both software design architecture and the software developer's position in developer networks affect project success. Within software engineering and computer science research, researchers and engineers have been long concerned with developing a fundamental understanding of product architecture and its characteristics (Chidamber and Kemerer 1994;Darcy et al. 2005;Subramanyam and Krishnan 2003). However, due to insufficient empirical evidence, prior research studies on software development have offered little potential to predict the phenomena with which the design constructs are associated. Thus, another goal for this research is to characterize the structure of design architectures and to present empirical evidence in support of this study's propositions. In addition, we propose that the developer's position emerges as an important element, playing a pivotal role in project performance due to communication and coordination problems that distributed development teams will face. Therefore, another goal of this study is to examine both team structure and software architecture and their impacts on project success. Using the open source software development context, our work shows the importance of access to knowledge and the modularity of the source code in project success. This has implications for allocations of resources, design and software quality.

One of the motivations of this study is to fill the gap existing in the field of research on product architecture and organizational structure in complex product development, resulting from a lack of empirical evidence to support theoretical propositions. The limitations of prior studies are reflected by their limited foci: their analyses either on single or small numbers of products and their design structure (e.g., Baldwin and Clark 2000) or were restricted to a single product-development organization (Henderson and Clark 1990). Furthermore, robust conclusions have been limited by their lack of appropriate metrics, as well as difficulty in collecting and analyzing data for characterizing key attributes of a product's architecture. Therefore, this study aims at filling this gap through rich empirical data, by characterizing the differences in design structure between complex software products, measuring these specific characteristics and comparing the impact of these constructs on software project success.

Another motivation for this study stems from the shortage of measurement of *modularity* in this stream of research. One exception is a survey-based study involving software firms and exploring the role of initial technology strategy (Nambisan 2002). Although this study yielded a measurement of the degree of modularity, it is based on the subjective data source that required the respondents to make complex and difficult judgments. In addition, the measure of modularity actually reflected only one dimension of modularity: the degree of decomposition of software product. Our study proposes to measure *modularity* as two constructs based on the software product's source code itself available from the SourceForge.net. Thus, by definition it is feasible to capture the essence of *modularity* by "live" and "objective" data source.

The rest of this paper is organized as follows: we first develop a theoretical research framework for understanding the relationship between software product architectures as well as developer networks and product (project) success. Next we describe details of the research design including the elaboration of data source, data collection, and

construct operationalization. The Negative Binomial Regression approach is applied for the model estimation. Main effect hypotheses are tested and results are summarized. We conclude by discussing findings and future research directions.

Theoretical Groundings and Development

This section develops a research framework. It begins by reviewing and synthesizing previous research on the architecture of complex systems and one of the important design solutions for complexity: modularity. Two dimensions of modularity—degree of decomposition and degree of coupling—are identified. We argue that, to obtain optimal level of modularity, both attributes must be considered while designing software products. Based on theoretical development, we substantiate how the extent of modularity affects project success. We then turn to a theoretical argument that relates one important aspect of a project's ego network—average degree centrality of its developers—to project success. Our theoretical rationale is that, since software development involves knowledge work, its most important resource is the specialized skills and knowledge that a developer brings to the project's task (Faraj and Sproull 2000). Linkage between the developers and the resultant collaboration networks are key vehicles through which project teams obtain access to knowledge. Examining the relationship between network position and software project output can provide an empirical indicator of the effectiveness of knowledge flows through such networks. A number of hypotheses will be generated to build the conceptual model of this thesis. Figure 1 summarizes the core thesis and depicts the major theoretical framework.

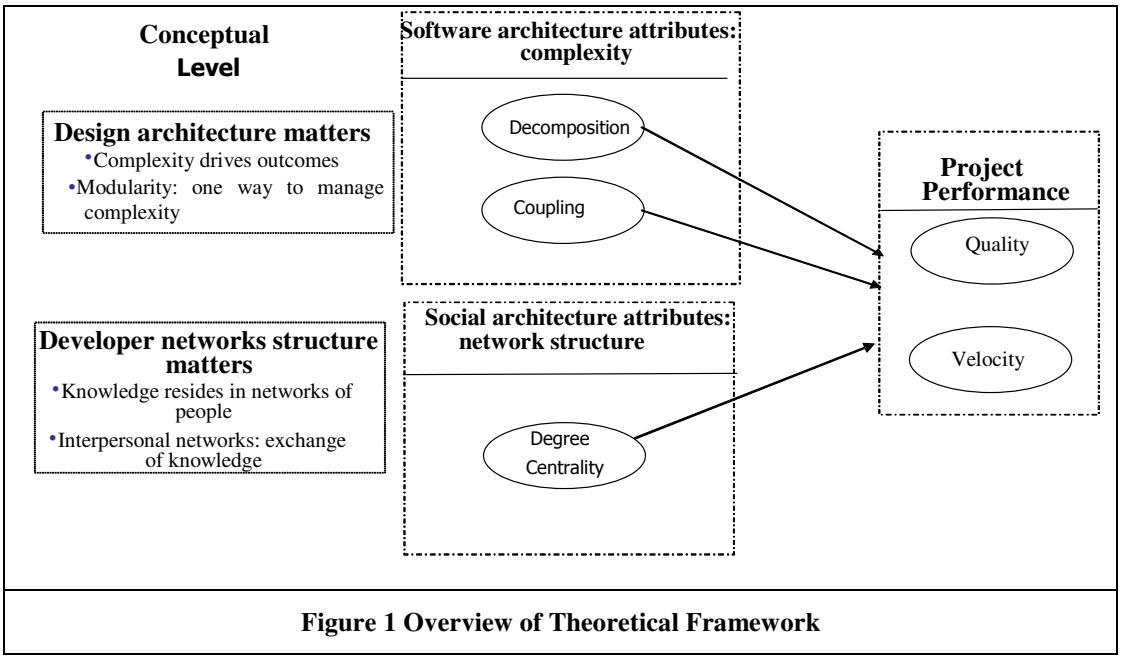


Figure 1 Overview of Theoretical Framework

Project Success: Multidimensional Construct

Since software development is a form of system development, it is reasonable to discuss project success based on information systems (IS) literature (Crowston et al. 2003). However, this study is not attempting to provide an exhaustive list of success variables, but rather to identify several constructs relevant to software projects.

In this study, software project success will be discussed from two dimensions: product quality or code quality, and project velocity. Generally speaking, product quality refers to the technical achievement of a project. This criteria for project success is consistent with the literature in information systems on software success (Grewal et al. 2006). To some extent, another dimension of project performance—project velocity—reflects development speed of a project.

Product quality. Generally speaking, software product quality has received increased attention due to its potential role in influencing customer satisfaction and the overall negative economic implications of shipping defective software product (Jones 1997). The most commonly cited model for IS success also suggest *system quality* as one of

important aspects of IS success (DeLone and McLean 1992). Product quality, also referred to as code quality here, has been studied extensively in software engineering (Slaughter et al. 1998). However, the software community has long been faced with severe difficulties in delivering and supporting quality software products on time (Blackburn and Van Wassenhove 2002). Despite efforts such as applying quality management and related principles to the development and maintenance of software, the software industry continues to be plagued by an inability to develop quality software products (Demarco 1995). It has been observed in the literature that allocating more resources in the early stage may improve the quality of the software considerably. The rationale behind this argument stems from the importance of requirement analysis and design (Krishnan et al. 2000). Design complexity was also reported as a significant variable impacting quality in the delivered software products (Harter et al. 2000). However, empirical evidence supporting the effect of software design architectures on quality is rare. Thus, it is important to identify the drivers of quality from the design perspective. In addition, code quality would be particularly applicable for studies of OSS, since the code is publicly available.

Project velocity. In DeLone and McLean's (1992) IS success model, systems development is implicitly treated as a one-off event. However, for OSS projects development is instead an ongoing activity, as the project continues to release updated versions (Raymond 1999). "An OSS project is characterized by a continuing process of developers fixing bugs, adding features, and releasing software" (Crowston et al. 2003). In order to take into account this characteristic of OSS development, we employ the construct *project velocity* as one indicator of project success. Project velocity, defined as the speed that a project releases software products into production, is taken as an explorative variable in this study. In OSS development, there is a strong community norm to "release early and release often" (Raymond 1999), which implies that an active release cycle is a sign of a healthy development process and project (given all else remains equal). For example, FreshMeat provides a "vitality score" that assesses how recently a project has made an announcement of progress on the FreshMeat site (Crowston et al. 2003). In addition, one consequence of cross-site problems in distributed environment was delay, which means the additional time it most likely takes to resolve an issue when more than one site is involved (Herbsleb and Mockus 2003). Therefore, what becomes a crucial research question and critical pragmatic issue is this: how design architecture and developer networks structure affect the speed with which an OSS project is accomplished.

Performance Effects of Design Structure Factors

Product design has received increased attention in the academic and business communities over the past decade (Nussbaum 1995;Ulrich and Pearson 1998). This attention resonates with the statement that "…designs are an important source of economic value, consumer welfare, and competitive advantage for individuals, companies, and countries" (Baldwin and Clark 2005). For the design activity to be managerially important, this study assumes that there must be significant variation in design performance among competing design efforts (Ulrich and Pearson 1998). Furthermore, for the design performance variation to be significant it should contribute to the competitively important differences in the underlying design structures of the product. However, without a way to measure the key attributes of a design with empirical data, past research cannot reach the level of specificity needed to draw robust managerial prescriptions (MacCormack et al. 2006).

The work on system architecture forms the foundation for viewing products or organizations as complex systems and analyzing the properties of complex systems according to the general design principles (Alexander 1964;Simon 1962). Based on the work of Simon (1962), complex systems can be characterized as consisting of a large number of elements that interact in a "non-simple" way. Thus, the characterizations of complexity have at least two underlying implications. First, the great number of elements creates difficulty in comprehending the structure that binds them. Second, understanding the interactions between the elements and the effects on system behavior and performance is non-trivial. Complexity increases as the system gets larger because designers need first to discover which elements interact with which others and then to identify the nature of the interaction relationship. There has been one kind of characterization of organizational forms focusing on the "segmentation" or "departmentalization" of activity and the allocation of specific functions to particular organizational subunits (Marengo et al. 2000;Rivkin and Siggelkow 2003). Therefore, the design of complex systems invokes two important principles: reductionism, or breaking up a complex whole into simpler units and the division of labor, grouping tasks or units based on similarity in function (Ethiraj and Levinthal 2004a). Actually, this idea has persisted in and remained central to the work of more recent organization theorists, such as those studying nearly decomposable systems (Simon, 1962); loosely coupled systems (Weick 1976); or pooled, sequential, and reciprocal interdependence (Thompson 1967).

One can find, in fact, that the above two principles resonate with the crux of one critical concept in design: modularity. Modularity is a general set of design principles for managing the complexity of large-scale interdependent systems. It also involves breaking up the system into discrete parts, responding to the first principle; and tracing how those parts communicate with each other through standardized interfaces or rules and specifications, paralleling the second one (Langlois 2002). In software development, modularization of the software code refers to the creation of high degree of independence, or "loose coupling", between components by standardized component interface specifications. Through modularization, a project's transparency may be increased, while barriers for developers to contribute become lower, allowing for specialization by enabling efficient use of knowledge.

We analyze modularity at two dimensions: (1) degree of decomposition, or the extent to which a software product is partitioned into modules; and (2) degree of coupling, or the number of "interactions" or "interdependencies" that the software product exhibits. This is not to suggest that are not other important facets one might want to consider. For example, interesting aspects of modularity could also include the extent to which components can be reused, or some notions of value and costs. However, as Simon (1962) has suggested, complexity is a matter both of the sheer number of distinct parts the system comprises and of the nature of the interconnections or interdependencies among those parts—the two dimensions of modularity constructs which this study uses reflect these two characterizations of complexity respectively. According to Simon, complexity, to some extent, drives outcomes. Furthermore, modularity has been regarded as one important solution to reduce complexity (Baldwin and Clark 2000). Therefore, we relate modularity to project performance.

The argument for including degree of decomposition as a critical factor in the theoretical model of software product quality relates to the ability of software developers to comprehend and control the various phases and roles when developing a complex software system. A fundamental approach to improving software development and managing complexity has been to modularize the design by splitting the implementation of the solution into parts (Parnas 1972). Because the process of designing and developing a software product challenges the developer to understand software through normal cognitive processes, a higher degree of decomposition—i.e., breaking up a complex whole into simpler units—could facilitate better comprehension of a system's functionality, economizing the cognitive demands placed on the designer (Miller 1956). However, excessive decomposition may have negative implications. The cognitive challenge in designing a highly modular architecture of autonomous modules with minimal interdependencies is certainly not trivial (Ethiraj and Levinthal 2004b). A simulation model developed by the just-cited authors also highlights the trade-off of the design variable modularity. Fine-grained decomposition of decision problems is likely to increase the uncertainty as well as the search cost. In turn, it results in difficulty for developers to comprehend functionality of relative classes, which more likely will lead to greater defects in software product. When past a certain point of decomposition, the product quality will decrease. The product quality will first increase with partition, but beyond a certain point, additional partition will decrease product quality.

> ***Hypothesis 1 (H1):*** *Ceteris paribus (Given all else remains equal), degree of decomposition of a product is curvilinearly (taking an inverted U-shape) related to its quality.*

The greater value of coupling implies the higher sensitivity of a system to change in other classes of the design and more difficulties for developers localizing quality problems. Consequently, tightly-coupled systems are more vulnerable to changes in their environment than systems composed of loosely coupled (i.e., more modular) elements. For example, take object-oriented software design: in the event that one kind of dependency arises from the simple sharing of services across the classes, increased class coupling could increase difficulties for designers and developers trying to maintain and manage the multiple interfaces required for sharing across classes. Another kind of dependency between classes arises from the inheritance hierarchy of classes in the design. Since an attribute or operation declared in the parent class will be inherited across all child classes, whenever there is a change to the parent class, changes would result in all child classes that must invoke methods or inherit attributes from parent classes. Therefore, the higher the number of inherited methods and variables, the greater the difficulty for developers in comprehending and understanding the functionality of the inheriting class. Hence, classes with higher degree of coupling value will be associated with a higher number of defects (bugs).

> ***Hypothesis 2 (H2):*** *Ceteris paribus, degree of coupling of a product is negatively related to product quality.*

Project velocity is used by software developers as a construct capturing the speed that a project releases software products into production. Following the same logic presented above when discussing the relationship between product quality and degree of decomposition, we argue that neither excess nor scarcity of decomposition would speed product releases. Low degree of decomposition may hide potentially important interactions between decision

choices from developers (Nambisan 2002), and thus increase the product development effort and duration. On the other hand, greater degree of decomposition of design tasks enhances the number of possible configurations achievable from a given set of inputs, greatly increasing the flexibility of a system (Schilling 2000). However, the speed and efficiency gains from flexibility will be offset by the increased time spent in the testing and integration phase (Ethiraj and Levinthal 2004b). Furthermore, excessive decomposition can slow down searches within a problem domain by increasing the information processing burden associated with managing specific product functions. In summary, the above discussion suggests that degree of partition is curvilinearly related to subsequent project velocity.

> ***Hypothesis 3 (H3):*** *Ceteris paribus, the degree of decomposition of a product is curvilinearly (taking an inverted U-shape) related to project velocity.*

In an integrated design, processes for developing tightly coupled components also become tightly coupled, because changing the design of one module generally precipitates a chain reaction of compensating transformations in the designs of other modules (Sanchez 1999). Thus, changes made by one developer alter the parameters within which others have to work. Each developer must constantly adapt to constrains imposed by other developers' actions (Levinthal and Warglien 1999). Unlike a tightly integrated product, "loosely coupled" (modular) products are composed of elements, or "modules," that independently perform distinct functions. Because of this nature of "separations of concern," each module or subsystem (clusters of modules with more dependencies within more than between) maintains a consistent functional focus. Developers may acquire cumulative experience with certain kinds of problems faster, enabling them to search for and evaluate alternative solutions more quickly (Pil and Cohen 2006). Recent literature also suggests that modular design can reduce design and development time by allowing parallelism in design and testing (Baldwin and Clark 2000).

> ***Hypothesis 4 (H4):*** *Ceteris paribus, degree of coupling is negatively related to project velocity.*

Performance Effects of Developer Network Structure

We analyze the structural position of software developers by modeling their work relationships and tasks as a collaborative network, in which the developers are vertices (nodes) of a network, and joint membership on an open source project is a collaborative link (tie) between the developers. For example, Figure 2 presents a fragmentation of developer collaborative networks at SourceForge.net, in which many developers may participate in one project, while at the same time one developer may contribute to multiple projects. There are five projects shown in the figure as five clusters: MCF, Virtual DubMod, eDoncky, Azureus, and snowball. There are another three projects which only have two participants for each: GaussSoft Development Kit, MIF Turning Machining, and WikiAnalysis. For instance, since the developer *amc1* participates in two projects at the same time, (s)he has links to every other developer also joining the project Azureus. At the same time there is a tie between *amc1* and *gwd1*, meaning that *amc1* and *gwd1* have the same joint membership to another project GaussSoft Development Kit. The label beside each node represents a team member's username at SourceForge.net. From this illustration, one can see, for example, that developer "belgabor" participates in three projects—MCF, Virtual DubMod, and Azureus— at the same time; while "gwd1" only works on project GaussSoft Development Kit.

In past studies researchers identified two kinds of network benefits associated with collaborative linkages (Ahuja 2000). First, collaborative linkages can provide the benefit of resource sharing, allowing developers to combine knowledge, skills, and expertise. Second, they can provide access to knowledge spillovers, serving as information conduits through which new insights into problems or failed approaches travel from one project member to another. Based on the work of Ahuja (2000), we consider one aspect of a developer's network structure that is likely to be relevant to the above benefits: degree centrality, or the extent to which the individual is linked to others in and across groups.

An individual in a central position is able to exchange messages with a large number of other group members. If an individual exchanges many messages, this will not only change his or her own position in the structure, but also others' relative positions as well, altering the entire structure (Carley 1991). Therefore, individuals who have a high degree centrality can exert more influence by virtue of being linked with a large number of people in the network. An individual's degree centrality, measured as the number of direct ties maintained by him or her, potentially provide both resource-sharing and knowledge-spillover benefits (Ahuja 2000). Recently, in their article on how the nature of the relationship among projects and developers impacts on the success of open source projects, Grewal et al (2006) posit that "network embeddedness"—capturing the architecture of network ties— relates to project

success. The major reasoning for this stems from the argument by organizational sociologists that organizational routines, processes, and structures are embedded in the broader social context (Smelser and Swedberg 1994).

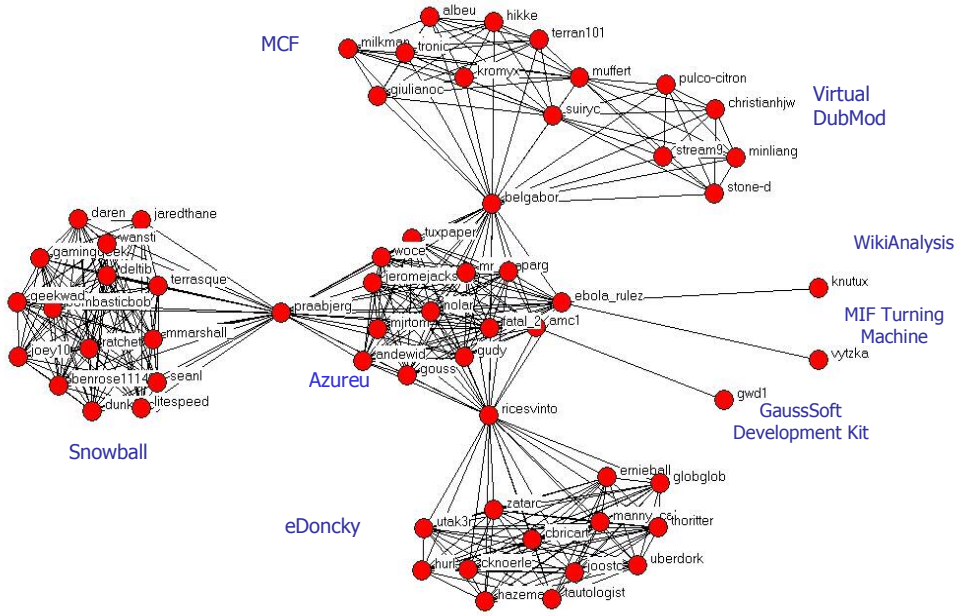


Figure 2 Fragmentation of developer networks at SourceForge

As mentioned previously, degree centrality is associated with certain benefits such as enabling resource sharing and knowledge spillover. Specifically, central developers have access to greater resources, since direct ties enable knowledge sharing (Berg et al. 1982) and better information quality for direct ties are perceived as a signal of quality (Sorenson and Stuart 2001). Development processes, including tasks such as code development, debugging and document writing etc., can be handled more effectively with greater resources. In addition, access to better quality information should also increase product quality, as this information tends to be more relevant, accurate and reliable (O'Reilly 1982). Research in diverse contexts shows high-quality information results in better outcomes than does low-quality information (Maltz and Kohli 1996;Raghunathan 1999). Thus, based on the above discussion, we propose that:

> ***Hypothesis 5 (H5):*** *Ceteris paribus, average degree centrality of developers in a project team is positively associated with product quality.*

Degree centrality reflects the extent of developers' involvement. For distributed and collaborative software design and development activities, centrality leads to greater involvements into the project, suggesting that more developers contribute to one project. Collaboration among these developers brings together complementary skills, thereby enabling developers to improve their performance by enhancing their own knowledge base (Ibarra 1993). Scale economies arise when larger projects generate significantly more knowledge than smaller projects. The greater the number of collaborative linkages formed by a project team, the higher the number of project output (software releases) it could obtain. All else being equal, a high extent of involvement may potentially increase the pace of the development processes.

> ***Hypothesis 6 (H6):*** *Ceteris paribus, average degree centrality of developers in a project team is positively associated with project velocity.*

Methodology

Data Source

SourceForge.net, owned and operated by OSTG, Inc., is used as the data source of this research. With over 100,000 projects and over 1 million registered users, SourceForge.net is one of the largest and most recognized OSS hosting web sites, offering features like bug tracking, project management, forum service, mailing list distribution, Concurrent Versions System (CVS), etc. An open source software development project hosted on SourceForge is typically initiated by an individual or a small group with an idea they want to develop for their own intellectual, personal, or business reasons. This individual or group generally develops a first, rough version of the code that outlines functionality. The project founder, in the most cases called "administrator" on SourceForge, sets up a project mailing lists for those interested in using or further developing the code, which they can use to seek help, provide information, or offer new open source code for others to discuss and test. For each project, SourceForge records a brief description and the registration date. It also includes information about project details such as developers' user names, development status (planning, alpha, beta, production, mature, etc.), project rankings, license, operating system, programming language, and topic. In addition, the system tracks and reports project activity as a public record: bug reports, support request, or forum posts, etc. The SourceForge.net is database driven and the supporting database (provided by University of Notre Dame) includes historic and status statistics on projects and developers' activities. By studying the web site SourceForge.net and accessing the relative monthly dumps provided by the University of Notre Dame, we could effectively explore developer networks structure as well as analyze design architectures of software products for each project.

Data Collection

We first selected project categories from the Software Map to draw the sample: Enterprise and Infrastructure. In other words, to be eligible to be included in the sample, a project had to be listed in one of these categories. Within these categories, we only selected the projects whose programming language is Java. We further differentiated between projects with many beta releases and projects with major milestone releases. All of projects in the sample have version of 1.0 or above, ensuring that the software products collected in the sample possess complete features and stable properties. Our selection resulted in a total population of more than 1,500 projects involving 6,000 developers. Of these, we selected a subset of our data sample: the top 100 projects and random 100 projects based on project rankings[1] on SourceForge. The sampling strategy was based on the prior study (Venkatraman and Prescott 1990). Generally speaking, the top 100 projects on the ranking lists of SourceForge.net could be regarded as successful ones since the rankings are created in terms of traffic ("popularity" of the projects), communication, and development ("velocity" of the projects). Therefore, the top 100 could be used to capture the high "performers". The random 100 was used to see if there was a significant difference in performance. Through the above selection process, the final sample size for analysis was 200 projects.

In addition to employing SourceForge as the reserch site, we also derived the data analysis using the database provided by University of Notre Dame. OSTG, Inc. has shared certain SourceForge.net data with Notre Dame, which includes over 100 relations (tables) in the database. Thus, we obtained data on a monthly basis both from project homepages on SourceForge and the data warehouse comprised of dumps from SourceForge. Although the data warehouse included project data from January 2003 through August 2006, for the current study, we collected data from the dump of April 2006. In this way, we can control for the change in quality or velocity of the software projects caused by tools and development technology. In general, we exercised caution in acquiring, cleaning, and analyzing data. Specifically, we submitted queries to extract desired data from the database. Then we downloaded those data to create the local database and set up relevant associations among tables (relations). Finally, by exerting

[1] Project rankings are generated based on the traffic (to track the raw popularity of a project), communication (to quantitatively determine how effectively a project is communicating), and development statistics (the work a project team is performing while developing their software) which are collected for each project. track.

relational operations such as JOIN on relevant tables in the local database, we obtained data on project name, age, number of developers, release date, number of bugs, etc.

Source code was downloaded for each software product in the format of a JAR file. A module, in this case, is a Java class. (A JAR file is a zip file used to distribute a set of Java classes. It is also used to store compiled Java classes and associated metadata that can constitute a program.). Then we applied a tool named Lattix Dependency Manager (LDM) to each JAR file of every project in the data sample. LDM was used to extract the system architecture (modules and their dependencies) from the source code of software products. (Please refer to Appendix A.). The results extracted from LDM were used for measuring constructs of degree of decomposition and degree of coupling. We obtained data on project releases through detailed archival research on release notes of each version of release on SourceForge. The data collection and coding exercise for the entire data sample involved studying over 3,000 release notes. We excluded those releases under version 1.0 or focused solely on fixing bugs.

To obtain data on developer networks, we created a *developer-project* table from the local database, including fields such as a developer's *user nam*e and the *projects* in which they are involved. Since one developer can participate in multiple projects, and each of those projects were not necessarily included within the sample of 200 projects we selected, the *developer-project* table that we created consisted not only of developers involved in the 200 projects of the subset sample, but also all other developers who were members of the additional 1,300 projects. This resulted in a total of approximately 6,000 developers in the global network. Pajek[2] was applied to transform the two-mode affiliation networks (the actors are developers, and the event is projects) to one-mode developer networks. Measures of degree centrality were computed by Pajek.

Construct Operationalization

Dependent variable—or project performance—is regarded as multiple dimentional constructs, including product quality and project velocity.

Product quality (QUALITY). The quality measure captures the number of unique problems reported by either project participants or end users. To control for the size effects, this measure is normalized by the product size measured in a thousand lines of code. So we measured *prodcut quality* as the ratio of thousand lines of code to the total number of bugs (errors or faults) found in products given a fixed period of time.

Project velocity (VELOCITY). Project velocity was operationalized as the number of stable releases per month over thousand lines of code in this study. If the number of defects (quality) is a technical-oriented success measure, then the number of releases (poject velocity) would be a development-oriented success measure. The measure of project velocity was adjusted from prior OSS studies which used the number of new releases as one of indicators of success among users of OSS projects (Crowston et al. 2003;Stewart et al. 2006). According to Stewart et al. (2006), one type of success relevant in OSS setting is the ability of the project to attract interest and input from the development community. Because open source projects often rely on voluntary input, attracting and motivating contributors is a key factor in project success. It has been suggested by the just-cited authors that one way of assessing success is by looking at the level of activity on a project. The specific measure used in this study counted the number of new releases produced during a fixed period. However, there are two possible limitations to this measure: First, without controling for the size of software product, the number of releases may not correctly reflect the level of activity on a project (for example, smaller projects are more likely to have more releases than bigger projects when all else is equal.); Second, *new* releases in OSS projects may be dedicated to fixing bugs without adding any new functionality. Therefore, based on the above analysis, we used the number of stable releases per thousand lines of code (i.e., size) during a fixed period (one month) as the construct operationalization.

Independent variables include two design structure constructs: degree of decomposition and degree of coupling; and one developer network structure construct: degree centrality.

Degree of decomposition (DECOMPOSITION). Degree of decomposition, one dimension of modularity, was defined as the extent to which a software product is partitioned into modules (i.e., "classes" in Java program). It was operationalized as the number of classes per thousand lines of code. This measurement was adapted from Nambisan's work (2002), in which the author operationalized the overall degree of product modularity as the ratio

[2] Pajek is a program for analysis and visualization of large networks.

between the total number of product design modules and the product size. The number of classes were obtained from running the tool (LDM) on source code downloaded from SourceForge.

Degree of coupling (COUPLING). Degree of coupling, another dimension of modularity, was defined as the degree to which a change to any single class caused a (potential) change to other classes within the product. Based on prior studies, we measured degree of coupling as the sum of all affected classes within a product, divided by the total number of classes a product has. We applied the tool of LDM on source code downloaded from SourceForge and obtained the sum of all affected classes.

Degree centrality (CENTRALITY). Degree centrality is based on the number of nodes (individuals) to which a node is adjacent. This construct was measured as the number of direct ties a developer has. Thus, average degree centrality of a project team—calculated as average number of direct ties all developers have in a project team—represented important channels for developers to access knowledge. The developers associated with each of these projects were considered an affiliation network. Each project could have multiple developers, and each developer could participate in multiple projects. This affiliation network, which is a two-mode network of project to developer, was transformed into a developer network, which is one-mode network of developer to developer,using Pajek 1.14 (de Nooy et al. 2005). This leads to a network of developers with collaboration as a non-directional tie. A tie connects two developers if they have joint membership in a project.

We sought to control for several factors that may influence the dependent variables.

Project age (AGE). The age of a project may serve as a proxy for serveral factors that could be fundamental to success, including the experience of the development group in working together, the entrenchment of the software in the user community, and the development status of the project. Therefore, the number of bugs, downloads and releases are all likely to increase with the age of project. Thus age, measured as the number of days from the time a project registered on SourceForge to its lates release, was used as a control variable.

Project Ranking (RANKING). SourceForge provides a ranking system that applies a single ranking formula to the statistics data for all hosted projects. Since project rankings are generated based on traffic, communication, and development statistics, the higher a project is ranked implies the more hits there are to its project webpage, more page views, or more CVS comimits, etc.

Project scale. (SCALE). We controlled in this model for the effect of project sacle (product size). This factor was expected to be inversely related to product quality. Prior studies of software productivity and quality have identified product size, measured in lines of code, as a primary determinant of product defects (Basili and Musa 1991). This variable was measured in terms of KLOC (thousand lines of code), which has been regarded as a primary metric for assessing product size in many empirical software productivity studies.

Projects per developer (PROPERDEV). This variable was operationalized as the number of projects in which each developer was participating. On one hand, the larger the number of projects one developer works on, the higher his or her information quality will be, thereby positvely impacting product quality. On the other hand, the more projects on which a developer is working, the more he or she is likely to be exposed to excess information, leading to cognitive overload and poorer work performance, and ultimately resulting in lower technical success. Based on this factor, we used the number of projects per developer as a control for product quality.

Team size (TEAM). Prior empirical studies have confirmed that number of developers is a crucial factor for project success. The open source development process has been characterized as a bazaar (Raymond 1998), implying the maxim, "given enough eyeballs, all bugs are shallow." This suggests a positive association between the number of developers in a project with project success.

OSS sponsorship (SPONSOR). As one of the control variables we also took into account whether an OSS project had a sponsor. We used a dummy variable—measured as one if the project got sponsorship and zero otherwise—as a control for the level of development activity or a potential user's perception of software quality and interests (Stewart et al. 2006). We coded sponsorship based on the description of the project provided on SourceForge and by visiting the project home page. Projects that neither stated an affiliation with an organization nor maintained project pages on an organization's website were categorized as having no support from third parties.

Model Specification

Since our dependent variables—product quality and project velocity—are "rate" variables, we used a nonlinear regression approach to avoid heteroskedastic, nonnormal residuals (Hausman et al. 1984). Specifically, we used the negative binomial regression model[3], a generalized form of Poisson regression, because the variance of our dependent variables exceeded its mean, indicating overdispersion in the data. This model took the form of $\ln \lambda_i = \beta' x_i + \varepsilon$, where λ_i equals the mean and variance of y_i, the observed frequencies for a random dependent variable Y (i=1, …N); x_i is the vector of regressors (i.e., independent variables); and exp(ε) has a gamma distribution with a mean of 1 and a variance of α^2, in which α is the reciprocal of the standard deviation of the heterogeneity distribution. The model estimates α from the observed data and thus directly captures any overdispersion . STATA 8 was used to estimate the model.

Results

The following two tables report descriptive statistics for all the variables of interest.

Table 3 Summary Statistics and Correlation Matrix for Top 100 Projects

				Correlation Coefficients										
Variables	**N**	**Mean**	**S.D.**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1. QUALITY	100	9.01	7.19	1.00										
2. VELOCITY	100	1.18	1.80	0.16	1.00									
3. DECOMPOSITION	100	7.10	3.82	0.65	0.38	1.00								
4. COUPLING	100	34.54	61.00	-0.56	0.18	-0.11	1.00							
5. CENTRALITY	100	22.09	12.49	0.48	0.58	-0.15	0.20	1.00						
6. AGE	100	980.39	573.41	0.37	0.29	-0.13	0.21	0.37	1.00					
7. RANKING	100	1148.65	1270.92	0.21	-0.14	-0.06	-0.29	-0.20	-0.15	1.00				
8. SCALE	100	55.95	155.19	-0.41	0.28	0.09	0.38	0.23	0.59	0.22	1.00			
9. PROPERDEV	100	1.10	0.54	-0.35	-0.52	0.34	0.17	0.67	0.23	0.06	0.18	1.00		
10. TEAM	100	15.00	22.74	0.44	0.61	0.33	0.58	0.87	0.33	0.05	0.47	0.22	1.00	
11. SPONSOR	100	0.32	0.47	0.10	0.15	-0.08	-0.03	0.24	0.06	-0.18	0.16	-0.23	0.08	1.00

Comparing the descriptive statistics of top 100 projects and random 100 projects on SourceForge.net, one can obtain some understandings of the basic natures of the data sample. On average, projects which have higher rankings on SourceForge are more likely to have a better quality than random samples. (The mean of QUALITY in the top 100 projects is 9.01, almost twice the mean of random 100 projects 4.43.). In the top 100 projects, developers are connected to each other quite densely. In other words, developers participating in higher ranked projects are more likely to own more direct ties than they would otherwise. In our data sample, the mean of average degree centrality of developers in top 100 projects (22.09) is more than twice of that in random ones (9.30). Through control variables SCALE and TEAM, we can find that, on average, the projects within top 100 rankings have larger scales (with a mean of 55.95 thousand lines of code) and team size (with a mean of 15 developers) than those in the random sample (where the mean is 22.4 and 4.7 respectively). The mean of the control variable PROPERDEV for random 100 projects (7.8) is seven times larger than the counterpart in top 100 projects (1.1), which suggests that developers involved in top ranked projects are more likely to focus on their own projects rather than work on additional ones.

Table 4 Summary Statistics and Correlation Matrix for Random 100 projects

				Correlation Coefficients										
Variables	**N**	**Mean**	**S.D.**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1. QUALITY	100	4.43432	20.89	1.00										
2. VELOCITY	100	3.27	5.46	0.07	1.00									
3. DECOMPOSITION	100	21.56	32.82	0.25	0.39	1.00								
4. COUPLING	100	71.45	94.21	-0.04	0.02	-0.03	1.00							
5. CENTRALITY	100	9.30	17.96	-0.18	-0.21	0.01	0.13	1.00						
6. AGE	100	728.90	521.32	0.08	0.12	-0.05	0.29	0.05	1.00					
7. RANKING	100	16977.76	18273.74	-0.11	-0.12	-0.10	-0.22	-0.06	-0.40	1.00				
8. SCALE	100	22.40	33.87	-0.53	-0.16	0.04	0.24	0.12	0.19	0.17	1.00			
9. PROPERDEV	100	7.80	11.20	-0.37	-0.23	0.19	0.18	0.49	0.36	0.04	0.10	1.00		
10. TEAM	100	4.70	6.68	0.14	0.13	0.34	-0.45	0.32	0.16	0.20	0.13	0.15	1.00	
11. SPONSOR	100	0.20	0.45	0.07	-0.05	0.03	0.03	0.04	0.12	-0.23	0.05	-0.08	0.24	1.00

[3] Negative binomial regression is an extension of Poisson regression, which is used to model the number of occurrences (counts) of an event that allows for over-dispersion.

Model testing

We reported the results of the negative binomial regression analyses and listed them for two different samples respectively. The models for the two dependent variables are individually significant for both samples. (Refer the last row of Table 5 and Table 6: -2 Log-likelihood). Moreover, various results were obtained for the same predictors between the two different samples.

Product quality. Hypothesis 1 posits a curvilinear (an inverted U-shaped) relationship between degree of decomposition and product quality. The results in the model for top 100 projects shown in table 5.A indicate that the linear term—the number of classes per thousand lines of code (DECOMPOSITION)—is positive and significant (β=0.1883, $p<0.1$), while DECOMPOSITION squared is negative and significant (β=-0.0927, $p<0.001$). Similarly, the results for random 100 projects also support H1, as shown in table 5.B.

In Hypothesis 2, we proposed that degree of coupling (COUPLING) would have a negative relationship with product quality. The results provide support this hypothesis for both top 100 projects and the random 100 projects. For the sample of top 100 projects, the coefficient of degree of coupling is negative and significant, supporting the hypothesis (β=-0.027, $p<0.01$). For the sample of random 100 projects, the coefficient is also negative and significant (β=-0.032, $p<0.001$). H2 is also supported by both data sample.

Hypothesis 5 is a test of the average degree centrality (CENTRALITY) of a project team on its product quality. The effect on product quality is positive, as hypothesized in both the top 100 project sample and random 100 project sample. In top 100 projects, we found a positive and statistically significant coefficient as predicted (β=0.0316, $p<0.001$). In random 100 projects sample, the coefficient is positive and significant, too (β=0.0305, $p<0.05$).

Table 5 Estimation of Models

5. A Top 100 projects on SourceForge

Predictor	QAULITY		VELOCITY	
	Coef.	Std. Err.	Coef.	Std. Err.
Constant	0.0036	0.0309	0.0858	0.5881
DECOMPOSITION	0.1883^{+}	0.1827	0.2165*	0.1163
DECOMPOSITION squared	-0.0927***	0.045	-0.0101*	0.0057
COUPLING	-0.027**	0.0011	-0.152**	0.0020
CENTRALITY	0.0316***	0.0087	0.0156**	0.0107
AGE	-0.0007***	0.0002	-0.0006*	0.0002
RANKING	0.0002**	0.0001	-0.0003	0.0001
SCALE	-0.024*	0.0163	-0.0274^{+}	0.0189
PROPERDEV	-0.2544*	0.0489	-0.0559	0.0018
TEAM	-0.3984*	0.2149	0.4723**	0.1024
SPONSOR	0.0206**	0.2262	-0.3485*	0.2603
Log likelihood (LL)	-232.8614		-139.8868	
-2LL	465.7728		279.9937	

5. B Random 100 projects on SourceForge

Predictor	QAULITY		VELOCITY	
	Coef.	Std. Err.	Coef.	Std. Err.
Constant	0.7011	0.7885	-1.0577	.7806
DECOMPOSITION	0.2526*	0.1462	0.3688*	0.1341
DECOMPOSITION squared	-0.0144**	0.0221	-0.0173**	0.0068
COUPLING	-0.032**	0.2330	-0.0035**	0.0026
CENTRALITY	0.0305*	0.0139	-0.0183^{+}	0.0134
AGE	0.0006	0.0004	-0.0001	0.0003
RANKING	0.0001*	9.65e-06	7.80e-06	7.44e-06
SCALE	-0.0058^{+}	0.0005	-0.0185^{+}	0.0005
PROPERDEV	-0.1933*	0.0560	-0.0126*	0.0019
TEAM	-0.083**	0.013	0.057*	0.105
SPONSOR	0.1424***	0.3542	-0.5925**	0.3231
Log likelihood (LL)	-195.9752		-138.3421	
-2LL	391.9504		276.6841	

*** $p<0.001$; ** $p<0.01$; * $p<0.05$; $^{+}$ $p<0.10$

Project velocity. Hypothesis 3 posits an inverted U-shaped relationship between the degree of decomposition of a product and its project velocity. The results for top 100 projects indicate that the linear term is positive and significant (β=0.2165, $p<0.05$), while the squared term is negative and significant (β=-0.0101, $p<0.05$), thus supporting H3. In table 5.B, the coefficient for the degree of decomposition is positive (β=0.3688, $p<0.05$), while that for the squared term of decomposition is negative and significant, supporting the hypothesis (β=-0.0173, $p<0.01$).

Hypothesis 4 posits a negative relationship between the degree of coupling and project velocity. In table 5.A, the coefficient is negative and significant (β=-0.152, $p<0.01$). Similarly, in table 5.B, the coefficient for coupling is significant and negative as predicted. Therefore, H4 is supported by both data samples.

Hypothesis 6 proposed competing predictions for the effect of average degree centrality of a project team on its project velocity. Table 5.A shows the results for top 100 projects, supporting the positive relationship: that high degree centrality is associated with faster project velocity (β= 0.0156, $p<0.01$). Meanwhile, table 5.B shows the results for random 100 projects, supporting the negative relationship: that high degree centrality is associated with lower project velocity (β=-0.0183, $p<0.1$).

Discussion

The goal of this research was to explore the antecedents to success in OSS projects. We have studied how the design architecture of products and the network structure of developers both relate to the success of open source projects. We focused on technical-oriented success (product quality) and development-oriented success (poject velocity). We developed and tested hypotheses regarding the effects of two product modularity characteristics (degree of decomposition and degree of coupling) and one developer network structure attribute (degree centrality) on performance in OSS projects. Results, as summarized in Table 6, were generally supportive of the arguments made in this paper, indicating that both design architectures and designer network structures are important to OSS project outcomes.

Table 6 Comparison of Model Testing Results for Top 100 and Random 100 projects

	DV1: Product quality		**DV2:Project Velocity**	
	Top100	**Random 100**	**Top 100**	**Random 100**
Decomposition	Sig. (+)$^{+}$	Sig. (+)*	Sig. (+)*	Sig. (+)*
Decomposition square	Sig.(-)***	Sig. (-)**	Sig.(-)*	Sig. (-)**
Coupling	Sig.(-)**	Sig.(-)**	sig(-)**	Sig.(-)**
Centrality	Sig.(+)***	Sig.(+)*	Sig.(+)**	Sig.(-)$^{+}$
Age	Sig. (-)***	Non (+)	Sig.(-)*	non.(-)
Ranking	Sig.(-)**	Sig.(-)*	non.(-)	non(-)
Scale	Sig.(-)*	Sig.(-)$^{+}$	Sig.(-)$^{+}$	Sig.(-)$^{+}$
Properdev	Sig.(-)*	Sig.(-)*	Non.(-)	Sig.(-)*
Team	Sig.(-)*	Sig.(-)**	Sig. (+)**	Sig.(+)*
SPONSOR	Sig.(+)**	sig.(+)***	Sig.(-)*	Sig(-)**

*** p<0.001; ** p<0.01; * p<0.05, $^{+}$p<0.10

Overall, the design factors, i.e., the two dimensions of product modularity, play an important role in project performance. Degree of decomposition of a software product had an inverted U-shaped relationship with product quality and project velocity. More specifically, in H1 and H3, we proposed that decomposition had a curvilinear (inverted U-shape) relationship with product quality and project velocity. For the models tested on both samples, the coefficient for decomposition is positive, while that for the squared term of partition is negative and significant, supporting the hypothesis. Moreover, degree of coupling was hypothesized to have a negative association with product quality (H2) and project velocity (H4). H2 and H4 were supported by both data samples. H5 and H6 are focused on the relationship between the network structure metric and two dependent variables. It is interesting to note that hypotheses about the relationship between degree centrality and project velocity are positive and significant for top 100 projects, but negative and significant for random 100 projects. The effects of the control variables were as expected. Among the control variables, SPONSOR, the measure of whether the OSS project got support or involvement from third parties, was positively associated with product quality, but negatively associated with project velocity.

From a theoretical perspective, our results suggest several directions for theory development about the effects of design and designer networks structure on project success. First, it is important to recognize that the effect of product modularity on the project performance varies along two dimensions: the number of classes into which a

product was partitioned and the number of interdependencies among these classes. The finding is consistent with our theoretical development. It would be fruitful for researchers in this domain to explore alternative dimensions of modularity and its effects on performance more deeply. Second, developer networks structure also plays an important role in project success, confirming that both design and designer networks are critical to project outcomes. However, some of the empirical results turned to be somewhat contradictory when tested in two data samples. Theoretical efforts to develop explanations would further enrich the understanding of the role of social and human capital in community-oriented knowledge development systems.

Our research has implications for project managers and developer in OSS environments as well as organizations. One of the interesting results: OSS projects with sponsorship could improve product quality, although with negative associations with project velocity. This may be related to the availability of technical support, upgrades, and other resources needed over the long term by consumers of software products. Since organizational affiliations are factors under the control of the individuals or organizations that start and run OSS projects, perhaps one practical implication of these results is that ensuring some organizational involvement may become a trend for OSS projects to attain further success. The ideas this study proposes enable practitioners to put emphasis on the role of developer networks and organizational arrangements cutting across firm or team boundaries. In our study the results suggested that the effect of a project team's average degree centrality (how many "central" developers it has) on project performance depends on the context being studied. This has contradictory impacts on project performance. What constitutes an enabling network structure for one set of actions may well be disabling for another (Podolny and Baron 1997).

The results overall support the reasoning put forth in the hypotheses, but there are limitations that should be addressed in future work. Perhaps the most significant limitation of the current study is that we used the joint membership to measure collaborative linkages between developers. Sometime the joint membership may not necessarily reflect the real case of whether developers within the same project team communicate with each other. Therefore, in order to capture this information, a more fine-grained measurement is needed. For example, by tracing email lists and forum posts whose topics are about design per se for each project, we could capture the technical interactions between team members; thereby build a developer network in which linkages reflect communication channels.

Other limitations of this work affect the generalizability of the conclusions. A limitation of this study is that we focus only on software products whose programming language is Java. The next step could be to extend the current focus and study software written in multiple programming languages, thus enabling us to address the generalizability issue. Second, this study limited the sample to the project domains of enterprise or infrastructure. We could extend these categories to broader domains such as utilities or software development, etc. Third, we have studied the antecedents of project success via a static view. Therefore, one future research direction could build in dynamics by taking time factors into account. Examining effects of design architecture evolution and developer networks over time should provide new insights. For example, we could analyze whether a developer's previous work experiences or social relationships will impact current project performance, or whether developers' reputations have any effect on the evolving pattern of collaborative networks. Fourth, although we took a relatively broad view of success by considering two dimensions (technical-oriented and development-oriented), the operationalization of each dimension was limited to a single measure. Other measures such as actual software usage or the number of software features added over time could be examined (cf. Stewart et al. 2006). Finally, another intriguing direction would be to examine whether (and if yes, then) how product and correspondent organizational architectures map onto each other (Sosa et al. 2004). Basically, this question addresses a proposition almost forty years old: Conway's Law, which claimed that the structure of the system mirrors the structure of the organization that designed it (Conway 1968). The CVS commit data would provide which developer is in charge of which design component. If we were also to capture the interdependencies between the product components, it would then be possible for us to compare the design structure matrix with the team interaction matrix and compare the alignment or misalignment between two architectures. In addition, we could also explore whether the extent of the alignment is associated with the project performance.

Acknowledgements

We thank the Boston University School of Management's Institute for Leading in a Dynamic Economy (BUILDE) for providing support for completion of this study. We gratefully acknowledge the advice and support of Professor John Henderson through the whole course of this project.

APPENDIX A DESIGN STRUCTURE MATRICES AND ITS APPLICATIONS TO SOFTWARE DESIGN

This study employs the tool Lattix Inc's dependency manager (LDM) to extract the system architecture (modules and their dependencies) from the source code of software products. The LDM tool uses a standard notion of dependency, in which module A depends on a module B if there are explicit references in A to syntactic elements of B (Sangal et al 2005). In this study, at least at its current stage, we only focus on products written in Java programming language. A module, in this case, is a Java class. The notion of dependency works well for understanding design dependencies, in which modifications to one module might affect another. The technique applied by LDM is supposed to capture static dependencies, which are extracted from code not in an execution state and use source code for input. Dynamic calls are extracted from code not in an execution state and use both executable code and the program state as input (MacCornmack et al 2006), which are beyond the focus of this study.

Figure A.1 shows the design structure matrix in its original binary form, where the design tasks to be performed are each represented by an identically labeled row and column of the matrix. The marked elements within each row identify which other tasks must contribute information for proper completion of the design. In other words, a mark in the i^{th} row and j^{th} column represent that the i^{th} task depends on the j^{th}. The diagonal elements in the matrix are essentially meaningless because usually dependencies of tasks on themselves are not considered. For example, the marks in row C are in columns A, B, and D, indicating that execution of task C requires information to be transferred from tasks A, B, and D. Sometimes, the mark "X" is replaced with "1" and otherwise with "0". Such matrices of system modeling that contain only entries of 0 of 1 is called "binary" matrices.

	A	B	C	D
A			X	
B				
C	X	X		X
D	X		X	

	A	B	C	D
A	0	0	1	0
B	0	0	0	0
C	1	1	0	1
D	1	0	1	0

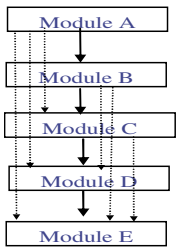
Figure A.1 A simple design structure matrix (binary matrix)

One important criterion that is used to evaluate the design is that the dependency relation should be acyclic. In matrix terms, this means that the tasks can be permutated so that the matrix is lower triangular—that is, with no entries above the diagonal(Sangal et al. 2005). In the above example as shown in Figure 1, tasks C and D are mutually dependent, so the tasks cannot be reordered to make the matrix lower triangular. In such situation, the main use of DSM is to identify a partitioning of design tasks that facilitates the flow of coordinative information in a project so that the cycle can be eliminated or reduced. A DSM, which has been rearranged so that all dependencies either fall below the diagonal or within groups, is in the form called *block triangular*.

The application of the DSM to software, with modules playing the role of tasks, is straightforward and intuitive to capture the inherent structure of design. The criteria that motivate partitioning in product development workflow have analogues in the structure of software systems. The terminology "acyclic" discussed in the previous section is called "layered" in the software infrastructure, often used approvingly of systems in which modules can be partitioned into layers, with each module having dependencies only on modules within its onw layer or belonging to the layer below. In particular, the layering principle sets out some constraints: each layer is a server to the layer above; each layer is a client to the layer below; each layer is permitted to interact with layers immediately above and below. Usually the system cannot meet all of the three constraints. For example, figure A.2 presents layering principle in software design which each layer depends on the layers underneath it.

	A	B	C	D	E
A					
B	X				
C	X	X			
D	X	X	X		
E	X	X	X	X	

a. DSM for layered system



b. Layered system (The arrow represents "dependence")

Figure A.2 Layered system

References

Ahuja, G. "Collaboration Networks, Structural Holes, and Innovation: A longitudinal Study," *Administrative Science Quarterly* (45) 2000, pp. 425-455.

Alexander, C. *Notes on the synthesis of form* Harvard University Press, Cambridge, MA, 1964.

Baldwin, C.Y., and Clark, K.B. *Design rules: the power of modularity* The MIT Press, Cambridge, MA, 2000.

Baldwin, C.Y., and Clark, K.B. "Between "Knowledge" and "the Economy": Notes on the Scientific Study of Design," Harvard Business School, 2005.

Berg, S., Duncan, J., and Friedman, P. *Joint Venture Strategies and Corporate Innovation* Oelgeschlager, Gunn and Hain, Cambridge, MA, 1982.

Blackburn, J., and Van Wassenhove, L. "Improve Software Productivity by Managing Complexity: Analysis of Finnish Software Development Projects," *Working paper, Owen Graduate School of Management # 2002-85*) 2002.

Brooks, F. *The Mythical Man-Month: Essays on Software Engineering* Addison Wesley, 1975.

Carley, K.M. "A theory of group stability," *American Sociological Review* (56:3) 1991, pp. 331-354.

Chidamber, S.R., and Kemerer, C.F. "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering* (20:6) 1994, pp. 476-493.

Conway, M.E. "How do committees invent?," *Datamation* (14:4) 1968, pp. 28-31.

Crowston, K., Annabi, H., and Howison, J. "Defining open source software project success," Proceedings of International Conferrence on Information Systems(ICIS), Seattle,Washington, 2003.

Darcy, D.P., Kemerer, C.F., Slaughter, S.A., and Tomayko, J.E. "The Structural Complexity of Software: Testing the Interaction of Coupling and Cohesion," *IEEE Transactions on Software Engineering* (31:11) 2005, pp. 982-995.

de Nooy, W., Mrvar, A., and Batagelj, V. *Exploratory Social Network Analysis with Pajek* Cambridge University Press, New York, NY, 2005.

DeLone, W.H., and McLean, E.R. "Information Systems Success: The Quest for the Dependent Variable," *Information Systems Research* (3:1) 1992, pp. 60-95.

Demarco, T. *Why Does Software Cost so much and Other Puzzles of the information Age* Dorset house Publishing, New York, 1995.

Ethiraj, S.K., and Levinthal, D. "Bounded Rationality and the Search for Organizational Architecture: An Evolutionary Perspective on the Design of Organizations and Their Evolvability," *Administrative Science Quarterly* (49) 2004a, pp. 404-437.

Ethiraj, S.K., and Levinthal, D.A. "Modularity and Innovation in Complex System," *Management Science* (50:2) 2004b, pp. 159-173.

Faraj, S., and Sproull, L. "Coordinating Expertise in Software Development Teams," *Management Science* (46:12) 2000, pp. 1554-1568.

Grewal, R., Lilien, G.L., and Mallapragada, G. "Location, Location, Location: How network embeddedness affects project success in open source systems," *Management Science* (52:7) 2006, pp. 1043-1056.

Harter, D.E., Krishnan, M.S., and Slaughter, S.A. "Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development," *Management Science* (46:4) 2000, pp. 451-466.

Hausman, J., Hall, B., and Griliches, Z. "Econometric model for count data with an application to the patents-R&D relationship," *Econometrica* (52) 1984, pp. 909-938.

Henderson, R.M., and Clark, K.B. "Architectural Innovation: The Reconfiguration Of Existing Product Technologies and the Failure of Established Firms," *Administrative Science Quarterly* (35:1), Mar 1990 1990, p 9.

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

Ibarra, H. "Network centrality, power, and innovation involvement: Determinants of technical and administrative roles," *The Academy of Management Journal* (36:3) 1993, pp. 471-501.

Jones, C. *Software Quality: Analysis and Guidelines for Success* Thomas Computer Press, London, 1997.

Krishnan, M.S., Kriebel, C.H., Kekre, S., and Mukhopadhyay, T. "An Empirical Analysis of Productivity and Quality in Software Products," *Management Science* (46:6) 2000, pp. 745-759.

Levinthal, D.A., and Warglien, M. "Landscape Design: Designing for Local Action in Complex Worlds," *Organisation Science* (10:3) 1999, pp. 342-357.

MacCormack, A., Rusnak, J., and Baldwin, C.Y. "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code," *Management Science* (52:7) 2006, pp. 1015-1030.

Maltz, E., and Kohli, A.K. "Market intelligence dissemination across functional boundaries," *Journal of Marketing Research* (33 (February)) 1996, pp. 47-61.

Marengo, L.G., Legrenzi, D., and Pasquali, C. "The structure of problem-solving knowledge and the structure of organizations," *Industrial and Corporate Change* (9) 2000, pp. 757-788.

Miller, G.A. "The magical number seven, plus or minus two: Some limits on our capacity for information processing," *Psychological Review* (7) 1956, pp. 88-99.

Nambisan, S. "Complementary Product Integration by High-Technology New Ventures: The Role of Initial Technology Strategy," *Management Science* (48:3) 2002, pp. 382-398.

Nussbaum, B. "Annual Design Awards," in: *Business Week*, 1995, p. 88.

O'Hara-Devereaux , M., and Johansen, R. *Globalwork: Bridging Distance, Culture, and Time* Jossey-Bass, San Francisco, 1994.

O'Reilly, C. "Variations in decision makers' use of information sources: The impact of quality and accessibility of information," *The Academy of Management Journal* (25:4) 1982, pp. 756-771.

Parnas, D.L. "A Technique for Software Module Specification with Examples," *Communications of the ACM* (15:5) 1972, pp. 330-336.

Pil, F.K., and Cohen, S.K. "Modularity: Implications for imitation, innovation, and sustained advantage," *Academy of Management Review* (31:4) 2006, pp. 995-1011.

Podolny, J.M., and Baron, R. "Resources and relationships: Social networks and mobility in the workplace," *American Sociological Review* (62) 1997, pp. 673-693.

Raghunathan, S. "Impact of information quality and decision-maker quality on decision quality: A theoretical model and simulation analysis," *Decision Support Systems* (26:4) 1999, pp. 275-286.

Raymond, E.S. *The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary*, (1st ed.) O'Reilly, Cambridge, Mass., 1999, pp. xi, 268.

Rivkin, J., and Siggelkow, N. "Balancing search and stability: Interdependencies among elements of organizational design," *Management Science* (49) 2003, pp. 290-321.

Sanchez, R. "Modular architectures in the marketing process," *Journal of Marketing* (63:Special Issue) 1999, pp. 92-111.

Sangal, N., Jordan, E., Sinha, V., and Jackson, D. "Using Dependency Models to Manage Complex Software Architecture," Conference on Object Oriented Programming Systems Languages and Applications, San Diego, CA, 2005.

Schilling, M.A. "Toward a general modular systems theory and its application to interfirm product modularity," *Academy of management review* (25:2) 2000, pp. 312-334.

Simon, H.A. *The sciences of the artificial* MIT Press, Cambridge, Massachusetts, 1962.

Slaughter, S.A., Harter, D.E., and Krishnan, M.S. "Evaluating the cost of software quality," *Communications of the ACM* (41:8) 1998, pp. 67-73.

Smelser, N.J., and Swedberg, R. (eds.) *Handbook of Economic Sociology*. Princeton University Press, Princeton,NJ, 1994.

Sorenson, O., and Stuart, T.E. "Syndication networks and the spatial distribution of venture capital investments," *American Journal of Sociology* (106:6) 2001, pp. 1546-1588.

Sosa, M., Eppinger, S.D., and Rowles, C.M. "The Misalignment of Product Architecture and Organizational Structure in Complex Product Development," *Management Science* (50:12) 2004, pp. 1674-1689.

Stewart, K.J., Ammeter, A.P., and Maruping, L.M. "Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects," *Information Systems Research* (17:2) 2006, pp. 126-144.

Subramanyam, R., and Krishnan, M.S. "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," *IEEE Transactions on Software Engineering* (29:4) 2003, pp. 297-310.

Thompson, J.D. *Organizations in action: social science bases of administrative theory* McGraw-Hill, New York, 1967, pp. xi, 192.

Ulrich, K., and Pearson, S. "Assessing the Importance of Design Through Product Archaeology," *Management Science* (44:3) 1998, pp. 352-369.

Venkatraman, N., and Prescott, J.E. "Environment-Strategy Coalignment: An Empirical Test Of Its Performance Implications," *Strategic Management Journal* (11:1) 1990, pp. 1-23.

Walz, D.B., Elam, J.J., and Curtis, B. "Inside a software design team: Knowledge acquisition, sharing, and integration," *Communications of the ACM* (36) 1993, pp. 63-77.

Weick, K.E. "Educational organizations as loosely coupled systems," *Administrative Science Quarterly* (21) 1976, pp. 1-19.