

**Impact of Product Architecture on Product Development and Project Effectiveness in the  
Online Collective Production of Information Goods**

**ABSTRACT**

Millions of distributed volunteers around the world create digital goods through the Internet. Examples include Wikipedia (<http://www.wikipedia.org>), Distributed Proofreaders Group (<http://www.pgdp.net>), NASA Clickworkers (<http://clickworkers.arc.nasa.gov/top>), and open source software (OSS) development (e.g. <http://sourceforge.net>). Projects such as these, dedicated to the online collective production of information goods (OCPIG), are characterized in part by production resources at a central project location on the net, self-assignment to tasks, and small units of work. Participation in OCPIG projects and OCPIG project effectiveness are not likely uniformly distributed across projects. I empirically explore the impact of product architecture on OCPIG product development and project effectiveness in a sample of OSS projects.

## **Impact of Product Architecture on Product Development and Project Effectiveness in the Online Collective Production of Information Goods**

The emergence and increasing use of the Internet has enabled many new forms of collaboration among people around the world. Web discussion forums, email, instant messenger, and weblogs, technologies that allow people to communicate virtually, enable new forms of interaction. One important example of a new type of interaction facilitated by the Internet is the online collective production of information goods (OCPIG). Many successful examples of such interaction exist. Wikipedia is a free online encyclopedia created by volunteers around the world who contribute their knowledge via articles about various topics (<http://www.wikipedia.org>). More than 523,000 registered users have contributed and modified over 2,400,000 articles in 187 languages. Project Gutenberg (<http://www.gutenberg.org>) and the Distributed Proofreaders group (<http://www.pgdp.net>) are both online communities in which volunteers digitize books in the Public Domain to make them freely available. As of 2003, more than 35,000 volunteers have made over 17,000 Public Domain books in 46 languages available. The open source software (OSS) movement is another popular example. People around the world voluntarily collaborate using the Internet to develop software. Popular examples include Linux, Apache, Perl, and MySQL. Several websites host thousands of projects and provide central access to thousands of users and potential contributors (e.g. <http://sourceforge.net>, <http://freshmeat.net>).

Projects dedicated to the OCPIG exhibit four important attributes that facilitate participation – Internet access, central project location, small units of work, and task self-assignment. First, all online collective production projects are hosted on the Internet, and thus are accessible to anyone with Internet access. Internet access reduces or eliminates the geographic and time constraints that operate in collocated work. This unlimited access thus

provides access to a potentially enormous labor pool. For example, anyone with Internet access can participate and contribute to a project, either by proofreading a page of an electronically scanned book, by editing content in the encyclopedia, or downloading and modifying the source code of an OSS project. People around the world, who may have been previously excluded from collocated development projects, may contribute to the online production of these digital goods.

Second, every project supports a central project location comprised of a specific technological architecture, generally a website that describes work opportunities and contains information necessary to complete work tasks. Sometimes participants actually perform their tasks at this central location, while others perform their work independently and then submit their contributions to this central location. This central location also provides resources to participants to help them perform their work, such as tools, tutorials, and FAQs<sup>1</sup>. The centralization of resources allows people to focus their cognitive effort on the task at hand, rather than search for information in external locations. The entire suite of technologies available through the central project location reduces the difficulty of information transfer between participants and projects by standardizing the tools used by participants to perform their work and communicate with each other (von Hippel, 1998). Standardized tools also allow people to work on several projects more easily because they can focus their cognitive effort on project tasks, such as software development, rather than learning new tools to communicate or perform tasks.

Third, each OCPIG project offers people the opportunity to contribute small units of work. For example, a volunteer can proofread just a single page of a book in the Distributed

---

<sup>1</sup> FAQs are web pages containing answers to frequently asked questions that new members of the project community may have. These pages provide information detailing work processes and participation procedures specific to the project, as well as norms and values of the community. FAQs are generally meant to orient newer members to the projects so that established members can spend more time working rather than answering common questions (Moon & Sproull, 2000).

Proofreaders group for Project Gutenberg, a person can edit a subsection of an article in Wikipedia, or a person can explore and attempt to fix one problem, or bug, in an OSS project. Finally, online collective production projects exhibit a high degree of task self-assignment among project members. Projects lack an imposed or predetermined hierarchical management structure that directs the delegation and assignment of tasks. Most participants are volunteers who select for themselves projects to work on and tasks to complete. A task list<sup>2</sup> may exist to help participants identify major project tasks to complete, but members themselves determine whether they will complete these identified tasks or other personally-determined tasks. Through task self-assignment, managers and project leaders are able to focus cognitive effort on tasks other than task delegation.

Thus, these four attributes of the online collective production of information goods essentially lower the costs incurred by volunteers to participate. However, participation in product development within these projects is not likely uniformly distributed across projects. It is therefore important to consider other factors that may differentially affect participation costs across projects to help understand variations in this participation and product development. One important set of factors includes characteristics of the product structure, such as degree of modularity and complexity. Some researchers have examined these effects of product structure on product quality in the online collective production of information goods (e.g. Baldwin & Clark, 2006; Benkler 2002; MacCormack, Rusnak, & Baldwin, 2006; Sagers, 2004). However, to my knowledge researchers have yet to examine how the product structure affects product development and project effectiveness in these OCPIG projects. Therefore, this paper addresses the question: *How do attributes of product structure impact product development and project*

---

<sup>2</sup> A task list is a structure used to help project members determine which tasks to complete. Such task lists include bug tracking systems or feature request files.

*effectiveness in the online collective production of information goods?* Answering this question is important because other aspects of the project or community may affect whether and how people participate, even if contributors' motivations are understood (e.g. Hars & Ou, 2002; Hertel, Niedner, & Herrmann, 2003; Lakhani & Wolf, 2005; Roberts, Hann, & Slaughter, 2006; Wasko & Faraj, 2000).

## **THEORETICAL BACKGROUND**

Many types of online communities exist that create information goods. This paper examines community-based OCPIG projects in which project members perform the work voluntarily. Without a product, the project community would not likely exist. It is therefore important to investigate factors that may significantly impact member participation in the project, particularly participation associated with actual product development. Product development, as defined in this paper, refers to actual changes and enhancements to the product accepted by the community. General community discussion and communication, such as problem reports, are not included in this definition of product development. Project members who make changes to the product are referred to as product developers. Because the efforts of community members are voluntarily, reducing costs associated with the voluntary work should positively impact the voluntary behavior of the project members. Therefore, I focus on costs of participation in these communities, specifically the cognitive demands and coordination costs associated with product development.

Attributes of the product architecture likely affect both the cognitive demands on project participants and the cost of coordination among project participants. One important characteristic of product architecture is modularity. The general concept of modularity suggests that a modular system consists of several smaller components, which are intentionally designed to be

independent from one another and self-contained (Baldwin and Clark, 2000; Sanchez and Mahoney, 1996; Schilling, 2000). Another important attribute of product architecture is the complexity of the product. Similar to product modularity, organization researchers have spent considerable effort theorizing about the most efficient ways for employees to work by examining the structure of the tasks performed. Several researchers suggest dividing work into small units and assembling them in a specific order to maximize worker efficiency and effort, such that any person could perform the work and achieve maximum results (e.g. principles of scientific management (Taylor, 1911, reprinted 2001)). Thus, understanding the implications of product architecture on cognitive demands and coordination costs may help to explain variations in product development and product quality across projects.

People have limits to the amount of information they can process and comprehend in a given time period, a concept labeled bounded rationality. By decreasing the complexity of the information, the likelihood that a person can fully process the information increases (March & Simon, 1993). In addition, lower task complexity is associated with lower problem-solving dependencies between tasks. By decreasing problem-solving dependencies, less information must be considered to complete the task. Lower problem-solving dependencies also increase task performance efficiency (von Hippel, 1990). Increasing system modularity should directly affect the complexity of the system and reduce problem-solving dependencies. For example, dividing a complex system into various subsystems allows each subsystem to be functionally less complex than the larger system as a whole (Simon, 1996). The size of each subsystem is smaller than the full system (Johnson, 2002), which also contributes to lower system complexity. Decreasing the level of system complexity by dividing it into smaller, independent components also increases the manageability of working with the system (Alexander, 1964). Increasing the

number of components within a system, holding system size constant, lowers the average size of the system components. Decreasing the interdependence between tasks that must be performed also decreases the level of complexity of the system, which improves task performance and innovation outcomes (von Hippel, 1990). While increasing system modularity should decrease the average complexity within system components, the complexity of maintaining the relationships among system components should increase. Thus, while a negative relationship between system modularity and complexity is hypothesized, at some point the relationship may become positive:

*Hypothesis 1. Degree of product modularity should be negatively related with degree of product complexity.*

In OCPIG projects, increasing the degree of product modularity should increase the number of available modules or units, and the average size and average complexity of each module or unit should decrease. As the average size of a module decreases, the quantity of information to be processed within a module should decrease, thus lowering the cognitive demands of product developers. A larger number of smaller and less complex modules decreases the amount of system information to be processed. Since lower complexity should reduce the quantity of information processed, the cognitive demands associated with altering less complex products should also decrease. As the cognitive demands associated with each product change decreases, product developers should be able to make additional changes for the same total cognitive cost, positively impacting product development:

*Hypothesis 2a. A higher degree of product complexity at T1 will be associated with a greater number of changes made to the product at T2.*

Since increasing modularity lowers the complexity of the system or product, increasing modularity should also decrease the level of difficulty associated with modifying the system or product. By enforcing relative independence between system components, changing components in one area of the system will not necessarily mandate changing components in other areas of the system (Sanchez & Mahoney, 1996). Software researchers have demonstrated that increasing the modularity of software products using information hiding techniques and module abstractions increases the ability to change one module without changing other modules in the product (Parnas, Clements, & Weiss, 1984). Therefore, increasing component independence should reduce the cognitive demands on product developers because they do not need to alter additional components to make the change. Greater ease in making product changes should result in a greater number of total product changes.

A more modular system may also increase the level of product standardization as independent components become interchangeable. Interchangeable components facilitate product changes (Sanchez & Mahoney, 1996; Ulrich, 1995), which favorably impact cost, product performance, and product development (Ulrich, 1995). Being able to reuse components allows product developers to focus on new product development or features, rather than reinventing the wheel. By reducing the cognitive demands of product developers, projects with products with a higher degree of modularity should be associated with a larger number of product changes.

Increasing system modularity should decrease the complexity of the system and facilitate making improvements to the system. It then follows that increasing system modularity should also decrease coordination efforts among the product developers. Increasing the number of components within a system and maintaining the independence of the system components allow

changes to be made to one area of the system without coordinating actions with the rest of the system. In addition, the increased number of product components reduces the average number of participants per module and promotes decentralization. Since fewer people work with the same component, fewer people must coordinate their efforts, effectively decreasing the costs of coordination (Johnson, 2002; MacCormack, Rusnack, & Baldwin, 2006). As the need for coordination decreases, project members can more easily work in parallel. Increasing modularity should lower coordination costs, which should increase the number of changes made by product developers in an OCPIG project. Overall, I hypothesize that increasing product modularity will decrease cognitive demand and coordination costs, resulting in a larger number of product changes made within an OCPIG project:

*Hypothesis 2b. A higher degree of product modularity at T1 will be associated with a greater number of changes made to the product at T2.*

Depending upon the product created by the OCPIG community, the amount of domain knowledge possessed by project members may directly affect a member's ability to participate in product development activities. For example, software development, or programming, is a cognitive exercise. The output of programming is not physical, and programs must be very precise and logical. To become a proficient software developer, people must learn at least one programming language and the structure and process of developing a program. Therefore, the level of skills or knowledge possessed by potential OSS project members should affect their ability to participate in product development activities. Project members with prior programming experience, knowledge of a particular programming language, or knowledge regarding a particular aspect of the software (e.g., security or networking) will find it easier to participate because, relative to their less experienced peers, they know better how and where to

make a development contribution. These experienced developers are more likely to contribute to product development than those with lower programming skill and knowledge levels because they can alter the product more easily. Developers with more technical skills should be able to contribute with less effort than those with fewer technical skills.

The degree of product modularity of OCPIG projects should affect the degree of pre-existing knowledge required to participate. Increasing product modularity should decrease the complexity of the individual system units or modules. Accordingly, the level of prior skills or knowledge required to participate and contribute to a simpler system should be lower. In software development, for example, a program may exhibit both security and user interface functionality. By separating the security functionality and the user interface functionality into separate modules, a potential developer only needs knowledge of either the security module or the user interface module, and not both, in order to make a product change. Therefore, increasing the modularity of the product should decrease the entry level of prior skills or knowledge required to contribute to the OCPIG project. By reducing the cognitive demands through increased modularity and reduced complexity, the number of product developers within a project should increase.

However, by decreasing the minimum skill level required to participate in product development activities, the likelihood of introducing product defects increase, because, on average, the participating product developers may be less talented and more likely to make mistakes. Raymond (1995) argues that the benefits of additional developers outweigh the costs of less-skilled developers, because as the number of developers increases, the probability of finding and fixing a bug increases in OSS development. In addition, while decreasing the skill level require to participate may negative affect product quality, many OCPIG project communities

have adopted norms and procedures, such as peer-reviews and governance mechanisms, to ensure a minimum level of product quality (Lattemann, & Stieglitz, 2005; O'Mahony, 2003; O'Mahony & Ferraro, 2003). In spite of the potential drawbacks associated with lowering the average skill level necessary to participate in product development activities, the benefits gained by increasing the number of product developers should outweigh these costs.

The level of product modularity will directly affect the proportion of the product that must be understood by a developer in order to make product changes. Increased product modularity increases the number of units of self-contained functionality or components within the product. By increasing the number of units, the average complexity of the product modules should decrease, and each module should contain more tightly related information. For example, in an empirical examination of software modularity, Parnas, Clements and Weiss (1984) demonstrated that increased software modularity eased understanding of the software by new developers joining the project. The quantity of the product to be examined and understood decreases since the developer can choose to evaluate only relevant modules. Thus, when a product developer intends to understand how specific aspects of the product function or relate, he or she can focus on only those units that deal with that specific information or functionality instead of learning the entire complex system. By decreasing the quantity of the product to be considered before participating in product development activities, due to the increased specificity of the product components, a larger pool of potential product developers should be able to make changes to the product due to a decrease in cognitive demands. It is more likely that the developer's level of existing skills and knowledge will be sufficient to learn and understand the product to make a product change. In addition, lowered average complexity of each module should decrease the time necessary to understand product components before making changes.

For these reasons, I hypothesize that projects that create products with a higher degree of modularity will have a larger number of people directly contributing to the development of the product:

*Hypothesis 3a. Projects with products exhibiting a higher degree of product modularity will be associated with a greater number of people making changes to the product*

*Hypothesis 3b. Projects with products exhibiting a lower degree of product complexity will be associated with a greater number of people making changes to the product*

Increased modularity is expected to decrease the degree of complexity within each system component or module. Decreased complexity should also be associated with lower cognitive demands on the product developer, which should facilitate making product changes. If product changes are easier to make, it then follows that the likelihood of introducing a mistake into the product also decreases, thus positively affecting product quality. In addition, as the cognitive demands on product developers decrease through lower product complexity, developers should be able to identify and resolve existing mistakes or problems in the product more easily. Greater ease in solving existing problems should also positively affect product quality. Finally, increased modularity should lower the coordination efforts among product developers as they make changes to the product. Increased task interdependence increases the probability that a developer can change his or her component without directly or indirectly changing other system components. Increased task independence should also positively affect product quality because he or she is less likely to introduce changes or mistakes outside of the relevant module(s). Therefore, I hypothesize that products with a higher degree of modularity and lower product complexity should exhibit higher product quality than products with a lower degree of modularity and higher complexity:

*Hypothesis 4a. Projects with products exhibiting a higher degree of product modularity will be associated with higher quality products*

*Hypothesis 4b. Projects with products exhibiting a lower degree of product complexity will be associated with higher quality products*

Not only is it important for OCPIG projects to create a quality product and attract product developers, OCPIG projects should also attract and retain additional project members who may not be project members. These members may use the information good created by the community, give feedback to the product developers, and provide support to other project members who use the product. These additional members may also promote the use of the product to friends, family, and colleagues, directly impacting the user base. Project effectiveness, as defined here, thus refers to the ability of an OCPIG project to create a usable information good, attract and retain product developers, project members, and a user base, and maintain an active community through communication and product development. I expect product modularity to be associated with project effectiveness. Projects with products exhibiting a greater degree of modularity and lower complexity should be associated with increased product development, more product developers, and higher quality products. As products are developed more quickly and are of higher quality, more potential users may be attracted to the project community. In addition, as project members provide feedback to product developers about the product, their suggestions may be incorporated more quickly into the product, enhancing the experience of the project member. Therefore, projects with products manifesting a higher degree of product modularity and lower complexity should be more effective than projects with products of lower modularity and higher complexity:

*Hypothesis 5a. Projects with products exhibiting a higher degree of product modularity will be more effective than projects with less modular products*

*Hypothesis 5b. Projects with products exhibiting a lower degree of product complexity will be more effective than projects with more complex products*

## **METHOD**

### **Data**

Due to the popularity of the OSS movement in the public and corporate space, OSS development projects will be used as the context for examining the online collective production of information goods. OSS development has been recognized as a viable method of software development, and researchers have spent the past five to seven years observing and learning about its development practices and processes (e.g., Halloran & Scherlis, 2002; Mockus, Fielding & Herbsleb, 2001; Raymond, 2000; Scacchi, 2001; von Hippel, 2001). The self-assigned tasks performed by members of OSS communities are relatively consistent in nature across developers. They include writing new code, fixing existing code, critiquing code and ideas, and identifying problems with the software. The software evolves as members contribute modifications, and newer versions are released as the project grows. Essentially all work revolves around the development of the software, which can be partitioned into the handful of activities described. Due to the lack of hierarchical assignment of these duties, project members cannot be forced to participate or perform specific tasks. Therefore, OSS projects are constrained by the talent and motivation level of project members. Without contributing members, the project ceases to grow.

A random sample of OSS projects hosted on SourceForge.net is analyzed to examine how attributes of product structure impact product development and project effectiveness.<sup>3</sup>

SourceForge.net and the Open Source Technology Group, Inc. (OSTG) have partnered with researchers at the University of Notre Dame to make parts of the SourceForge.net database available for academic and scholarly research. SourceForge.net provides a monthly data dump to Notre Dame researchers, who make the data available through a data warehouse. Data from the October 2006 data dump, which include data through October 31, 2006, were used to construct the sample of projects. Thus unit of analysis is the OSS project.

I examine projects registered between November 1, 2004 and November 1, 2005, a registration span that allows sufficient time (between one and two years) for the projects to exhibit product development and community involvement through October 31, 2006, the ending period of the data set used.<sup>4</sup> I limited the sample to projects written in Java. In addition, project must have had at least one software release and designate a relatively mature development stage (Beta or Production/Stable) to ensure the existence of an initial code base from which project members could work. Finally, projects needed to exhibit a minimum level of project activity to ensure a pool of potential product developers.<sup>5</sup> Eight projects meet these criteria and are used to evaluate the research question posed in this paper (refer to Table 1 for a description of the projects).

---

<sup>3</sup> SourceForge.net (<http://sourceforge.net>) is the largest and most popular website for hosting OSS projects on the Internet, providing space for hosting project web pages as well as space and tools for storing and managing the software under development for a given project. SourceForge.net also provides management tools, such as tracking systems, discussion forums, and email distribution lists (<http://sourceforge.net/docs/B02/en/>). Since the project started, SourceForge.net has hosted more than 140,000 projects and boasted more than 1,800,000 registered users.

<sup>4</sup> November 2004 was selected as the lower bound on project registration, as November 2004 is the start of consistent monthly data dumps from SourceForge.net to the Notre Dame repository.

<sup>5</sup> Project activity refers to any communication behavior made by OSS project members (e.g. posts to discussion forums, bug trackers, email lists, etc.).

-----  
Insert Table 1 about here  
-----

## Measures

***Project architecture.*** Two measures of project architecture characteristics are used in this study. First, *degree of modularity* describes the relationship among software modules within a specific application, specifically the interdependences between software modules. Measurements are calculated by directly evaluating the source code using metrics devised to evaluate the design quality of object-oriented software based on the interdependence of components in the design (Martin, 1994). Refer to Appendix A for the exact calculation used in this paper. Second, *software complexity* describes the complexity of the control flow logic within an application by examining the number of linearly independent paths within the source code. A greater number of logic paths indicate greater complexity. McCabe's Cyclomatic Complexity measure, a standard metric used in software engineering, represents software complexity in this paper (McCabe, 1976). Both degree of modularity and software complexity were calculated for each package within the application and weighted by the size of the library (log of the lines of source code). These measures were constructed from the source code included in the first software release made by each project sampled.

***Product development.*** Two metrics are calculated to represent product development. First, product changes denotes the number of changes to source code stored in the source code repository. Both CVS and SVN repository tools provide detailed history logs recording changes to source code files and the authors of the changes over time. Product changes is thus a count of the number of changes to source code files stored in the repository during the specified time period. This metric was calculated two times: *6 month product changes* represent product

changes occurring during the six months following the first software release, and *total product changes* represents all product changes occurring after the first release (actual time period varies across projects). Second, *product developers* represents the number of people authoring the product changes during the specified time period.<sup>6</sup> This metric includes people making changes to the product at any time after the first release.

**Product quality.** Many OSS researchers have used problem, or bug, reports as a proxy for software quality. However, not all project communities use specific bug tracking tools – some may use email lists or discussion boards to manage community member reported bugs making it difficult for researchers to filter identified bugs from other communication. Additionally, projects with smaller communities may experience fewer bug reports because of the small user base. In this case, however, fewer bug reports does not necessarily indicate better software quality. Therefore, static source code analysis tools were used to evaluate software quality in this study. The results of three tools, FindBugs (<http://findbugs.sourceforge.net/>), JLint(<http://archo.com/jlint/>), and PMD (<http://pmd.sourceforge.net/>), were combined to provide an objective measure of product quality, called *number of bugs*.<sup>7</sup> The results of these tools were combined because each uses a different algorithm to identify different types of problems or defects within the source code. This measure should be interpreted in a relative sense, as each tool exhibits some degree of false positive and false negative reporting. Number of bugs was constructed from the source code contained in the first release. A larger number of bugs or defects indicates lower product quality.

---

<sup>6</sup> Not all project members have write access to the source code repository. Authors of source code changes who do not have write access are often credited in the comment section within the change log. Thus, the number of authors includes the number of people with write access who made changes to the source code plus the number of unique authors without write access credited in the comments section for their contribution.

<sup>7</sup> Each tool produces a specific number of defects. Product quality is the sum of the defects identified by each tool.

*Project effectiveness.* SourceForge.net reports a ranking for every hosted OSS project, which is a measure of project activity, calculated for each OSS project using the amount of communication activity, development activity, and project website traffic. Specifically, project traffic incorporates the number of times the project website has been viewed and the number of times project files have been downloaded. Communication activity considers the volume of project communication by examining the number of email messages, discussion forum posts, and project tracker items. Finally, development activity captures the number of changes made to the source code, the date of the most recent file release, and how recently PAs have logged into the project website.<sup>8</sup> *Project effectiveness* is a binary measure constructed here to represent relative project effectiveness (0 = ineffective, 1 = effective), controlling for the size of the target audience. Thus, better-ranked projects are generally considered more effective than worse-ranked projects.<sup>9</sup>

## RESULTS

Due to the small sample size and exploratory nature of this study, correlations are used to evaluate hypotheses. Table 2 shows the correlations between all the variables. I predicted that degree of modularity would be negatively associated with product complexity (H1). Their correlation ( $r = -0.20$ ) suggests this relationship may be true, but due to the small number of projects in the current sample, this correlation is not significant. I also predicted that degree of modularity would be positively associated with product changes (H2a) and product complexity would be negatively associated with product changes (H2b). The correlation between degree of

---

<sup>8</sup> Crowston, et al, (2004) have proposed the most complete set of project success measures to date based on theoretical development and empirical examination: size of the development team (measured using posts to the bug tracker), the number of downloads, SourceForge.net activity ranking, and time to fix bugs.

<sup>9</sup> The GenoViz, software used to visually represent biological genomes, and VARS, software used to annotate video data such as scientific deep-sea video data, projects have a relatively smaller target audience, than other projects, such as Eclipse SQL Explorer and ZK, both tools useful to a wide range of software developers.

modularity and product changes (6 months:  $r = 0.27$ ; total changes:  $r = 0.15$ ) is in the right direction, but not significant due to the small sample. The correlation between product complexity and product changes (6 months:  $r = 0.78$ ,  $p < 0.05$ ; total changes:  $r = 0.85$ ,  $p < 0.01$ ) suggests a significant relationship, although in the opposite direction hypothesized. Projects with more complex software exhibit more changes to the source code.

-----  
Insert Table 2 about here  
-----

I also predicted that degree of modularity and product complexity would be significantly associated with the number of product developers, or people making changes to the product (H3a and H3b). The correlation between modularity and number of product developers ( $r = 0.17$ ) is not significant, but in the hypothesized direction. The correlation between product complexity and number of developers ( $r = 0.49$ ) is also not significant, and is not in the direction hypothesized. Neither H3a nor H3b is supported.

Hypotheses 4a and 4b suggest significant relationships between degree of modularity and product complexity with product quality. The correlation between modularity and number of bugs ( $r = 0.21$ ) is not significant, and is the opposite direction hypothesized. The correlation between product complexity and number of bugs ( $r = 0.68$ ,  $p = 0.06$ ) is marginally significant and in the direction hypothesized. Greater product complexity is associated with more bugs in the product (lower product quality).

Finally, hypotheses 5a and 5b suggest significant relationships between degree of modularity and product complexity with project effectiveness. The correlation between modularity and project effectiveness ( $r = 0.66$ ,  $p = 0.07$ ) is marginally significant and in the direction hypothesized. Projects that have products with a greater degree of modularity are more

likely to be effective. The correlation between product complexity and project effectiveness ( $r = 0.45$ ) is not significant, and not in the direction hypothesized. Thus, marginal support is provided for H5a, and H5b is not supported.

## DISCUSSION

Initial analysis of correlations between degree of product modularity and complexity with dependent variables, such as product changes, number of product developers, and product quality, revealed interesting findings. First, the relationship between degree of product modularity and product complexity is negative, but not significant. This may result from the possibility that at some point, maintaining the relationships between product components becomes more complex than adding additional modules. Additionally for H2a, degree of modularity is positively associated with the number of changes to the product, as predicted, but the relationship is not significant. The relationship between degree of product modularity and the number of product developers (H3a) is also positive, but the magnitude of this relationship is not significant as well. Surprisingly, the relationship between degree of product modularity and number of bugs (H4a) was positive, however not statistically significant. Finally, degree of product modularity is statistically positively related to project effectiveness (H5a). Overall, relationships related to the degree of product modularity are not statistically significant, but are generally in the direction predicted. Future studies with a larger number of OSS projects analyzed may increase the power of these relationships, subsequently providing support for the hypotheses.

An unexpected and significant positive relationship between degree of product complexity and the number of product changes was found. As product complexity increases, more changes were made to the product in the following six months as well as through the end of

the sampling period. Two arguments may help to explain this finding. First, product developers may recognize the degree of product complexity and may be working to reduce the complexity to encourage future product development. Second, developers may spend time fixing bugs over time that are introduced into the product as new product changes are made, leading to a larger number of product changes. These arguments should be empirically examined in future work by measuring product complexity and product quality in future time periods, and more closely examining the types of product changes made.

The positive relationship between degree of product complexity and number of product developers was unexpected, though not significant. This finding may be attributable to the operationalization and actual measurement of the number of product developers. Alternatively, this finding may support the software engineering concept known as the “mythical man-month,” suggesting that increasing the number of software developers increases the amount of overhead and possible software complexity (Brooks, 1975, reprinted 1995). Future work should address the measurement of the number of product developers and additionally explore the working relationships among these developers. The significant positive relationship between degree of product complexity and number of bugs supports H4b as predicted: higher product complexity was associated with lower product quality, or a greater number of defects in the source code. Finally, the unexpected, though not significant, positive relationship between degree of product modularity and project effectiveness suggests that projects that create more complex products are more effective. Products with greater complexity may require additional communication among project members to resolve issues and add new functionality, inflating one of the components of project effectiveness. In addition, findings suggest that projects with more complex products may be associated with more product developers. However, increased product complexity should not

directly translate into a larger user base. This relationship between product complexity and project effectiveness should be evaluated more closely in future studies.

### **LIMITATIONS AND FUTURE WORK**

While this study is the first to use metrics from software engineering to measure software modularity, software complexity, and software quality in relation to software development in OSS projects, some limitations exist. First, the sample size used to evaluate proposed hypotheses is small and consists of OSS projects not well known to many people outside of the OSS community, and possibly within the OSS community. Second, the time period of possible product development within the projects sampled is much shorter than desired, which may result in measurement error and the inability to capture the intended phenomenon empirically. These limitations are partly due to the starting period of projects eligible for sampling. Future work will consider more projects with a much longer tenure (e.g. registered in 2000 or 2001), which likely have established a much larger user base and community over time. Increasing the number of OSS projects analyzed will also enable more sophisticated statistical analyses of the relationships among variables. A third probable limitation of the current study is the measurement of the number of product developers. Some projects may not credit developers in the repository log, thus reducing the number of developers actually participating in product development activities. Future work will determine the feasibility of extracting author information from the tools used within each OSS project to manage product development (e.g. discussion boards, email lists, bug trackers).

Additional measures may be extracted in future work to further explore the relationships among product modularity, product complexity, product development, and project effectiveness. For example, in addition to the number of changes made to the product, the size of the changes

made to the product may prove significant. Products that are less modular or more complex may be associated with larger changes than more modular and less complex products. In addition, examining the tenure of the product developers and their level of activity over time related to product architecture may also provide more insight into this relationship. Finally, the size of the project member community (project members who are not product developers) communicating via email, discussion boards, and project tracking tools should directly affect project effectiveness. Thus exploring the relationship between product architecture and the size of the project member community may provide additional understanding into the relationship between product architecture and project effectiveness.

A final key issue identified both theoretically and empirically in this study is the nature of the relationship between degree of product modularity and degree of product complexity. What is the optimal degree of product modularity to minimize both component complexity as well as the complexity of relationships among product components? Several empirical issues must be addressed to answer this question. First, how can the complexity of the relationships be separated from the complexity of the components? What are the costs associated with maintaining a particular modular structure relative to the complexity of the components? Answers to questions such as these would provide significant insight into the relationship between product modularity and complexity, and likely extend beyond the context of software development and the OCPIG into organization theory.

## APPENDIX A

The metrics used to evaluate the design quality of object-oriented software based on the interdependence of components in the design (Martin, 1994) measure the relationships between source code packages, and whether techniques of information hiding and module abstraction are used appropriately. The following metrics are calculated from the source code:

- *Afferent coupling* (AC): Number of other packages within the project that depend upon classes within the package; a measure of the package's responsibility (higher values indicate higher responsibility)
- *Efferent coupling* (EC): Number of other packages within the project that classes within the package depend upon; a measure of the package's independence (higher values indicate lower independence)
- *Abstractness* (A): Ratio of the number of abstract classes/interfaces to the total number of classes within a package; 0 indicates a completely concrete package, 1 indicates a completely abstract package
- *Instability* (I): Ratio of a package's independence to total independence and responsibility [ $EC / (EC + AC)$ ]; measure of a package's resilience to change; 0 indicates complete stability (changes may be made to other packages without affecting this package), 1 indicates complete instability (changes made to other packages will likely directly affect this package)
- *Distance from the main sequence* (D): perpendicular distance of a package from the idealized line  $A + I = 1$ ; measure of the balance between abstractness and instability of a package; 0 indicates ideal balance, 1 indicates complete imbalance

Projects with high design quality should exhibit the following characteristics:

- Packages with higher values of A should have lower values of EC (high independence)
- Packages with lower values of A should have lower values of AC (low responsibility)
- As a package's I increases, its A should decrease and vice versa (thus, low values of D)

In this paper, D is used as a proxy for source code modularity. To calculate D for a single project, each package's D is weighted by the logarithm of the source lines of code (LOC) within the package. The weighted package D's are then summed and divided by the sum of the logarithm of the lines of code (LOC):

$$D_{\text{project}} = \frac{\sum D_i * \log(\text{LOC}_i)}{\sum \log(\text{LOC}_i)}$$

for every package i in the project. Finally,  $D_{\text{project}}$  is multiplied by -1, so that greater values indicate higher degrees of modularity.

## REFERENCES

- Brooks, F. P. 1975, reprinted 1995. *The mythical man-month: Essays on software engineering, 20<sup>th</sup> anniversary edition*. Addison-Wesley Professional.
- Baldwin, C. Y., & Clark, K. B. 2000. *Design rules: The power of modularity (Vol. 1)*. Cambridge, MA: The MIT Press.
- Baldwin, C., & Clark, K. B. 2006. The architecture of cooperation: Does code architecture mitigate free riding in the open source development model? *Management Science*, 52(7): 1116-1127.
- Benkler, Y. 2002. Coase's penguin, or Linux and the nature of the firm. *The Yale Law Journal*, 112(3): 369-446.
- Crowston, K., Annabi, H., Howison, J., & Masango, C. 2004. *Towards a portfolio of FLOSS project success measures*. Paper presented at the Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering, International Conference on Software Engineering, Edinburgh, Scotland.
- Halloran, T.J., and William L. Scherlis. 2002. *High quality and open source software practices*. Presented at the Meeting Challenges and Surviving Success: The 2nd Workshop on Open Source Software Engineering, 24th ICSE, Orlando, FL.
- Hars, A., & Ou, S. 2002. Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce*, 6(3): 25-39.
- Hertel, G., Niedner, S., & Herrmann, S. 2003. Motivation of software developers in open source projects: An Internet-based survey of contributors to the Linux kernel. *Research Policy*, 32: 1159-1177.
- Johnson, J. P. 2002. Open source software: Private provision of a public good. *Journal of*

*Economic and Management Strategy*, 11(4): 637-662.

Lakhani, K. R., and Robert Wolf. 2005. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. In J. Feller, B. Fitzgerald, S. Hissam & K. R. Lakhani (Eds.), *Perspectives on free and open source software*: Cambridge, MA: MIT Press.

Lattemann, C., & Stieglitz, S. 2005. *Framework for governance in open source communities*. Paper presented at the 38th Hawaii International Conference on System Sciences, Hawaii.

MacCormack, A., Rusnak, J., & Baldwin, C. 2006. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, 52(7): 1015-1030.

March, J., & Simon, H. A. 1993. *Organizations* (2nd ed.). Cambridge, MA: Blackwell Publishers.

Martin, Robert. 1994. **OO design quality metrics: An analysis of dependencies**. Retrieved 1/13/2006, 2006, from <http://www.objectmentor.com/resources/articles/oodmetrc.pdf>

McCabe, Thomas J. 1976. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4): 308-320.

Mockus, A., Fielding, R. T., & Herbsleb, J. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Proceedings on Software Engineering and Methodology*, 11(3): 309-346.

O'Mahony, S. 2003. Guarding the commons: How community managed software projects protect their work. *Research Policy*, 32: 1179-1198.

- O'Mahony, S., & Ferraro, F. 2003. *Managing the boundary of an 'open' project*. Paper presented at the Workshop on The Network Construction of Markets, Supported by the Hewlett Foundation grant to the Co-Evolution of States and Markets Program, organized by John Padgett and Woody Powell, Santa Fe, NM.
- Parnas, D. L., Clements, P. C., & Weiss, D. M. 1985. The modular structure of complex systems. *IEEE Transactions on Software Engineering*, 11: 259-266.
- Raymond, E. S. 2000. *The Cathedral and the Bazaar*. Retrieved 2/1/2004, 2004, from <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/>
- Roberts, J., Hann, I.-H., & Slaughter, S. A. 2006. Understanding the motivation, participation, and performance of open source software developers: A longitudinal study of the Apache projects. *Management Science*, 52(7): 984-999.
- Sagers, G. W. 2004. *The influence of network governance factors on success in open source software development projects*. Paper presented at the Twenty-fifth International Conference on Information Systems, Washington, D.C.
- Sanchez, R., & Mahoney, J. T. 1996. Modularity, flexibility, and knowledge management in product and organization design. *Strategic Management Journal*, 17: 63-76.
- Scacchi, W. 2001. *Software development practices in open software development communities: A comparative case study*. Paper presented at the Making Sense of the Bazaar: First Workshop on Open Source Software Engineering, 23rd International Conference on Software Engineering, Toronto, Ontario, Canada.
- Schilling, M. A. 2000. Toward a general modular systems theory and its application to interfirm product modularity. *Academy of Management Review*, 25(2): 312-334.
- Simon, H. A. 1996. *The sciences of the artificial* (3rd ed.). Cambridge, MA: MIT Press.

- Taylor, F. W. 1911, reprinted 2001. Principles of scientific management. In J. M. Shafritz & J. S. Ott (Eds.), *Classics of organization theory* (5th ed.). Belmont, CA: Wadsworth Group/Thomson Learning.
- Ulrich, K. 1995. Role of product architecture in the manufacturing firm. *Research Policy*, 24: 419-440.
- von Hippel, E. (1990). Task partitioning: An innovation processing variable. *Research Policy*, 19(5): 407-418.
- von Hippel, E. 1998. Economics of product development by users: The impact of 'sticky' local information. *Management Science*, 44(5): 629-644.
- von Hippel, E. 2001. Innovation by user communities: Learning from open-source software. *MIT Sloan Management Review*, 42(4): 82-86.
- Wasko, M., & Faraj, S. 2000. 'It is what one does': Why people participate and help others in electronic communities of practice. *Journal of Strategic Information Systems*, 9(2-3): 155-173.