

How Modularity and Contribution Networks affect Software Development and Quality in Open Source Software Development

Summary: My thesis examines the impact of product modularity and contribution structure on product development and product quality in voluntary or community-based open source software (OSS) development projects. My main propositions, which draw from cognitive psychology, social psychology and social network theories, posit that product modularity and the contribution network structure affect characteristics of product contributions and the quality of the product produced. Data used to test these hypotheses are derived from software releases from 50 OSS projects hosted on SourceForge.net.

OSS development is an instance of a larger phenomenon, the voluntary, distributed, and decentralized production of digital products and services. Following the spread of the Internet over the last decade, a growing variety of products and services are now collectively produced by distributed volunteers (for instance, Wikipedia (<http://wikipedia.org>) and Project Gutenberg (<http://www.gutenberg.org>)). None of these projects uses a formal “organizational” hierarchy to allocate and enforce tasks, yet volunteers self-organize to create high-quality products and services.¹ Because OSS is one of the earliest instances of the voluntary, distributed and decentralized production of digital products, and because it involves a product whose characteristics can be described and compared in relatively straightforward ways, OSS is a good context to study how product characteristics and the relationships among contributors affect product quality and contributions in such a work environment. In addition to exploring OSS as an important and emerging type of organization – the voluntary, distributed, and decentralized production of digital products and services – the context of OSS projects offers a new window into the process of organizing and developing software, providing another opportunity to address the enduring question of how to organize for effectiveness.

¹ Management structure is not absent in these projects. As the project members work together and establish norms of collaboration, a normative hierarchy emerges that guides future interactions and task assignment.

Theoretical Model

Preliminaries: In general, the goal of any OSS project is to create a working piece of software (for example, Linux) through software product development. I define *product development* as comprising product contributions accepted by the community.² A product contribution could entail adding a new component, modifying an existing component, or deleting an existing component.³ Product contributors are people who make changes to the product.

The concept of modularity has been useful in several contexts. For example, organizational scholars have found modular organizational forms to increase the flexibility of organizations, enable new product development, and lower costs (Lei, Hitt, & Goldhar, 1996; Sanchez and Mahoney, 1996; Schilling & Steensma, 2001). In general, *product modularity* is a strategy for efficiently organizing complex products (Baldwin & Clark, 1997). Modular systems are comprised of components “that are designed independently but still function as an integrated whole” (p. 86, Baldwin & Clark, 1997). Interfaces “describe in detail how the [components] will interact, including how they will fit together, connect, and communicate” (p. 86, Baldwin & Clark, 1997). Thus, modularity describes the degree to which components within a product or system are independent or loosely coupled from one another, yet function as an integrated whole using interfaces (Baldwin and Clark, 2000; Sanchez and Mahoney, 1996). Products with a low degree of modularity are considered highly integrated products, because the functionality is tightly coupled and interdependent, regardless of the number of components.

Modularity in software is a measure related to a product’s *architecture*, defined as “the scheme by which the function of the product is allocated to physical components” (Ulrich, 1995, p. 240). Components refer to “a separable physical part or subassembly... or distinct region of the product” (Ulrich, 1995, p.421). Relationships among individual components are described by coupling, such that “two components are coupled if a change made to one component requires a change to the other

² The process by which contributions are accepted into the community varies across projects, and is not specifically examined in my thesis.

³ General community discussion and communication, such as problem reports, are not considered product contributions in my thesis, but will be considered in a subsequent research.

component in order for the overall product to work correctly” (Ulrich, 1995, p. 422). Refer to Figure 1 for a conceptual illustration of product modularity. Prior work has established that software modularity improves software quality and reduces programmatic effort in both the development and maintenance of centrally-developed software (Parnas, Clements, & Weiss, 1984; Sullivan, Griswold, Cai, & Hallen, 2001). It therefore seems natural to expect the degree of product modularity to affect product development and product quality in OSS projects. I summarize some of my theoretical development in two parts, each of which corresponds to a separate essay in my dissertation.

(A) Modularity, Product Contributions and Product Quality: Product modularity in OSS should affect characteristics of product contributions because a more modular product has a lower level of coupling among components, and thus a lower interdependence of product functionality. To decrease component coupling or increase functional independence, additional functional components or interface components will be created (see Figure 1). Thus, when product modularity is higher, a given product of fixed functionality will be composed of a greater number of independent or loosely coupled components. As the number of components increases, functionality becomes more specialized and isolated within each of the components, and each component contains less functionality on average (Baldwin and Clark, 2000; Parnas, Clements, & Weiss, 1984; Sanchez and Mahoney, 1996; Ulrich, 1995). Contributors should therefore be able to alter the product by changing a smaller portion of the functionality (Korson & Vaishnavi, 1987). This is important because altering a smaller portion of functionality to make a change will be associated with smaller contributions that require less effort to construct. Thus, the average size of a product contribution should decrease.

Not only should the average size of product contributions be lower for more modular products, the total number of contributions should also be higher. This is because the average size of components is lower when product modularity is higher. The scope of functionality contained within each component therefore decreases on average. Functionality previously contained in a single component will now be divided among several modules (Baldwin and Clark, 2000; Parnas, Clements, & Weiss, 1984; Sanchez and Mahoney, 1996; Ulrich, 1995). Thus, a contribution must be made to each new component in a more

modular product to affect the same functionality as one contribution to the single component in a less modular product. This will increase the total number of contributions to the OSS project.

Modularity should also affect the attributes of product contributors in OSS projects. If higher product modularity is associated with smaller product contributions on average, the level of effort needed to make a contribution will be lower, because there is likely less work to be performed. In addition, if higher modularity is associated with the scope of functionality contained within a component on average, the contributor is required to process less total information and less irrelevant information to make a contribution (March & Simon, 1993; Shaft & Vessey, 2006), which lowers cognitive costs. A lower effort level and skill level will therefore be sufficient to contribute. In summary, increasing product modularity should increase the pool of potential contributors by lowering cognitive and skill thresholds for contribution.

If degree of product modularity affects the number of contributions and the number of contributors, it should also affect the distribution of work across contributors. When product modularity is higher, the degree of component coupling is lower and the number of components is higher. Maintaining component independence should allow changes to one component to be made without affecting other components, effectively decreasing coordination efforts among product contributors because product contributors can more easily work separately and in parallel (Ghosh, 2003; Johnson, 2002; MacCormack, Rusnack, & Baldwin, 2006). Distributing the work among more people leads to a more equal distribution of work among project contributors. Thus, modularity will affect the distribution of contributions across product contributors

To summarize, I propose that product modularity will positively affect both product contributions and contributors.

Proposition 1: Degree of product modularity will positively influence characteristics of product contributions and contributors

I now consider how modularity might affect product quality in OSS projects. First, higher product modularity should positively affect product quality by reducing the number of errors or problems in the

product. This is because, by definition, more modular products have more loosely coupled components. Therefore, product contributors should more easily be able to identify an existing error or problem within a component due to lower information processing requirements and less irrelevant information to find the error. In addition, smaller and more loosely coupled components foster problem resolution because problem identification is more likely localized to specific modules (Korson & Vaishnavi, 1987; Parnas, Clements, & Weiss, 1984). As the ease of problem identification and resolution increases as modularity increases, more existing problems should be identified and resolved within the product, increasing product quality in products with higher modularity.

Proposition 2a: Degree of product modularity will positively influence product quality

Next, the software engineering literature suggests that lowering the size of a software program and increasing the simplicity of source code written reduces the number of errors or bugs in the product (Dunn, 1984; Marciniak, 1994; Parnas, Clements, & Weiss, 1984; Ward, 1989). Since the average size of each contribution is smaller in more modular products, the probability that the contribution contains an error or bug should decrease. If products with higher modularity elicit a greater number of smaller sized contributions, then each change is more likely to affect a smaller percentage of total product functionality. In addition, the probability that a change to one component will inadvertently affect another component decreases due to more loosely coupled components (Korson & Vaishnavi, 1987; Parnas, Clements, & Weiss, 1984; Ulrich, 1995). If a contribution affects a smaller portion of functionality and fewer components, the contribution is less likely to introduce an error into the product. This increases product quality.

Proponents of the open source software movement claim that one reason for the creation of high quality software results from the large number of people involved in writing the source code or looking for bugs (Raymond, 2000). Projects with more modular products should attract a greater number of unique product contributors. More product contributors should increase the probability that existing problems or bugs are found since more people are available to try to fix identified problems. Each person

may try to find a unique solution to the problem, thus increasing the probability the problem is solved and decreasing the time to solve the problem, resulting in higher product quality.

Proposition 2b: Characteristics of product contributions and contributors will mediate the effect of degree of modularity on product quality

Refer to Figure 2 for an illustration of the proposed relationships among degree of product modularity, contribution and contribution characteristics, and product quality.

(B) Contribution Networks, Product Contributions and Product Quality: I introduce the concept of a *contribution network* in this thesis, which is a bimodal network that describes the structure of relationships among product contributors within an OSS project. One set of nodes, member nodes, represents project contributors who have made at least one product contribution. The other set of nodes, component nodes, represents the source code components of the software. A tie exists between a member node and a component node when a project contributor has made a product contribution to the component node. No direct ties exist between member nodes or between component nodes. Two project contributors relate to each other when they make product contributions to the same software component.

Each node in the contribution network has a *degree centrality*, which reflects the number of ties each node has to other nodes in the network (Proctor & Loomis, 1951; Shaw, 1954; Wasserman & Faust, 1994). Degree centralization describes the dispersion or range of each node's degree centrality by comparing each node's value to the maximum value (Freeman, 1979; Wasserman & Faust, 1994). As the dispersion of node degree centrality increases, the centralization of the network increases because fewer nodes have a high number of ties. Networks with low degree centralization are more decentralized because each node has approximately the same number of ties. The *density* of the contribution network refers to the average standard node degree centrality, and is a recommended measure of group cohesion. Network density is highest when each node in the network is connected to all other network nodes (Blau, 1977; Wasserman & Faust, 1994). Refer to Figure 3 for a conceptual illustration of these properties of contribution networks.

These two properties of the contribution network structure should affect both product contributors and product contributions in OSS projects. A person's position in the network affects his or her contribution behavior – the more centrally located (i.e. the greater his or her degree centrality), the more likely he or she will contribute (Wasko & Faraj, 2005). As the degree of each node increases, network density increases because the nodes are more interconnected. Thus the density of the network should also affect contribution behavior. As contribution network density increases, it is more likely that each product contributor will contribute to product components. In a contribution network with maximum density, each product contributor contributes to all product components. Additional effort will be required to coordinate work (Ghosh, 2003), negatively affecting effort spent performing work, which should reduce the total number of product contributions and contributors. In a low density contribution network, network nodes will exhibit fewer connections on average. Fewer network ties should allow product contributors to work more independently and thus lower costs of coordination, resulting in more product contributors and product contributions.

Contribution network structure density should also influence the joining behavior of new product contributors. Networks that generate value are more likely to attract new members (Butler et al, 2001), and networks that generate value through productive exchanges are more likely to attract new resources and strengthen the motivation of members to continue participating, which increases network sustainability (Oliver & Marwell, 1988). Product contributors in high density contribution networks have a large number of ties to other network nodes, enabling easy access to information and resources for all project members (Burt, 1992). New product contributors may then identify any existing product contributor to easily find necessary information and resources, generating value for the new product contributor. Thus, increasing network value by facilitating access to project information and resources in dense contribution networks should increase the number of unique product contributors. To summarize:

Proposition 3: Properties of contribution network structure will positively influence characteristics of product contributions and contributors

Properties of the contribution network structure should also affect product quality in OSS projects. Contribution networks with lower density will exhibit nodes with lower degree centrality. Therefore, a smaller number of product contributors will work with each product component, resulting in fewer coordination requirements. As coordination requirements decrease, the probability of introducing an error decreases since the likelihood that a contribution affects another person's code decreases. In addition, projects with contribution networks having a larger number of member nodes (i.e. more unique product contributors) have more product contributors examining the product, which should increase the probability that existing problems will be found and solved. Decreasing the probability of introducing new errors as well as increasing the probability of finding and solving existing errors should increase product quality. To summarize:

Proposition 4a: Properties of contribution network structure will positively influence product quality

A product contributor's position in the network should affect his or her project role and access to project information and resources (Burt, 1992). In contrast with product contributors with low degree centrality, contributors with high degree centrality will have made contributions to several components relative to other product contributors, and thus will have greater access to project information and resources. As product contributors continue to make product contributions, the number of ties to existing components will increase and ties to existing components will strengthen, increasing their degree centrality. Most people join OSS projects as peripheral product contributors, who are contributors who make few product contributions and are not likely central project members. As peripheral contributors make more and more contributions, they may become core product contributors who exhibit high degree centrality since they make the majority of product contributions. These core contributors have access to better project information and resources (Burt, 1992), and often act as project leaders and maintain the project knowledge, influencing and enforcing project goals and direction (Rogers & Kincaid 1981).

Core product contributors exploit existing project knowledge, while peripheral product contributors explore new possibilities (March, 1991). Projects with only core product contributors will

exhibit strong leadership and direction, which is useful in the short run as the product grows and exhibits new features. However, the lack of peripheral developers who find and fix problems may lead to lower software quality over time. Similarly, projects with only peripheral product contributors will exhibit high software quality in the short run as many problems will be identified and solved. However, these projects will lack strong leadership and direction, thus lowering quality the long run as each peripheral product contributor leads the product in different directions. Thus, OSS projects should have a balance of core and peripheral product contributors, or a more centralized contribution network, to maintain leadership and direction, and to improve product quality over time.

Proposition 4b: Characteristics of product contributions and contributors will mediate the effect of properties of contribution network structure on product quality

Refer to Figure 4 for an illustration of the proposed relationships among contribution network structure, contribution and contribution characteristics, and product quality.

Method

Data: I analyze 2 samples of OSS projects hosted on SourceForge.net.⁴ The project sampling frame is limited to communities that do not exhibit obvious corporate or organizational sponsorship or involvement, products written in Java, projects with at least one software release in a relatively mature development stage (Beta or Production/Stable) to ensure the existence of an initial code base from which project members could work, and projects exhibiting a sufficient level of project activity to ensure a sufficiently large pool of potential product contributors.⁵ The frame includes approximately 180 projects from which 2 random samples of 25 are drawn. Each project contains multiple software releases. Data for

⁴ SourceForge.net (<http://sourceforge.net>) is the largest and most popular website for hosting community-based 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 149,000 projects and more than 1,590,000 registered users.

⁵ Project activity refers to any communication behavior made by OSS project members (e.g. posts to discussion forums, bug trackers, email lists, etc.) using SourceForge tools.

the relevant measures were derived for each major software release (e.g. 1.x, 2.x) and the subsequent development period before the next major release.

Measures: Table 1 summarizes the measures used to test hypotheses derived from the propositions. Information extracted from the detailed history logs provided by source code repository tools, which record changes to source code files and the authors of the changes over time, are used to calculate the metrics for product contributions, product contributors, and contribution network structure. Modularity and product quality metrics were derived from the actual source code for each software release examined.

Analysis Plan

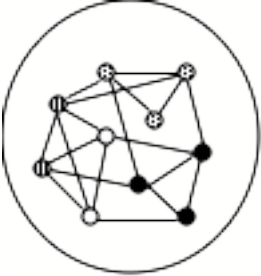
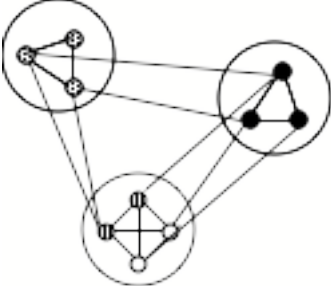
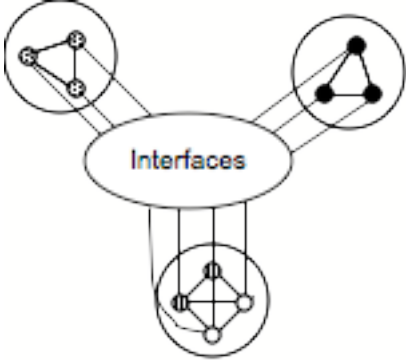
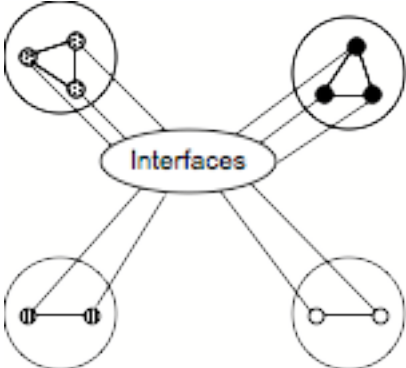
Since my data is a panel collected for a cross-section of OSS projects and over time, I will use fixed effects regression models to test the theoretical models. For example, to test the relationship between modularity and software quality, two fixed effects regression models will be analyzed using the software releases as observations in the 25 groups, using OSS project as the group (that is, controlling for unobserved heterogeneity across projects). One model examines whether *modularity* significantly predicts *number of bugs*, and the second examines whether *modularity* significantly predicts *software complexity*. Study 1 will investigate the relationship between degree of modularity, product development and product quality (Proposition 1, Proposition 2a, and Proposition 2b) using one sample of 25 projects. Study 2 will investigate the relationship between contribution network structure properties, characteristics of product contributions and contributors, and product quality (Proposition 3, Proposition 4a, and Proposition 4b) using the second sample of 25 OSS projects. I will also examine how proxies for what I cannot fully observe about heterogeneity across application types and governance structures (that is, the rules and norms that define how contributions are accepted by the OSS community) affect my results.

Implications of the Work

Empirically examining the implications of degree of product modularity will further our understanding of the impact of product design on organizing activities in voluntary, decentralized and distributed development projects. In addition, observing essentially all organizing processes in these emerging organizations will complement and challenge existing theories of organizing. The empirical

findings regarding the relationship between modularity and product quality will have implications for the practice of software development. These findings may prove very useful to software development project managers who need practical and quantifiable techniques for monitoring the software development process and subsequent product quality. The measures of modularity and product quality empirically examined and tested may also help software development managers evaluate existing software and software under development. As corporations adopt technologies that support distributed, decentralized, collaboration, understanding how outcome measures like quality and quantity are related to the work design are also important. Further, as more knowledge is created and used by such voluntary distributed efforts, society will become increasingly reliant on their quality. Understanding the connection between how these projects are “architected”, the relationships among their members and the eventual quality of their outputs is thus likely to have practical implications that are quite broad, and towards which I hope my dissertation will contribute.

Figure 1: Conceptual illustration of product modularity^a

| <i>Product design</i> | <i>Description of degree of modularity</i> | <i>Number of Components</i> |
|---|--|-----------------------------|
|  | <p>Low modularity, even though there is a single component that is not tightly coupled to other components, because all product functionality is tightly integrated within a single component</p> | <p>1</p> |
|  | <p>Low modularity – related functionality has been separated into different components, but the new components are still tightly coupled</p> | <p>3</p> |
|  | <p>Higher modularity – interfaces have been created to facilitate component interaction; components containing functionality are now loosely coupled; components are independent of one another, and related functionality is tightly integrated within each component</p> | <p>4</p> |
|  | <p>Highest modularity – only related product functionality contained within each component; all components are loosely coupled with one another and interact via interfaces</p> | <p>5</p> |

^a Functionality is represented by the small circles within the components: ● ○ ○ ○ The patterns within the circles represent related product functionality.

Figure 2: Relationship between product modularity, contribution and contributor characteristics, and product quality

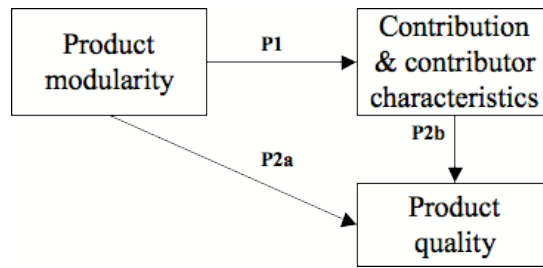


Figure 3: Conceptual illustration of contribution network structure^a

| <i>Network configuration</i> | <i>Network structure description</i> |
|------------------------------|---|
| | <p>High density network – each product contributor makes many contributions to several components, and each component receives many contributions from several product contributors</p> <p>Low degree centrality – each product contributor makes many contributions to the same number of components, and each component receives many contributions from the same number of product contributors; low variability in node degree centrality</p> <p>All product contributors are core contributors</p> |
| | <p>Low density network – each product contributor makes few contributions to few components, and each component receives few contributions from few product contributors</p> <p>Low degree centrality – each product contributor makes few contributions to few components, and each component receives few contributions from few product contributors; low variability in node degree centrality</p> <p>All product contributors are peripheral contributors</p> |
| | <p>Low density network – most product contributors make few contributions, and components receive contributions from few contributors</p> <p>High degree centrality – high variability in node degree centrality (one contributor makes many contributions to many components; others make few contributions to a single component)</p> <p>The center product contributor is a core contributor, while the outer product contributors are peripheral contributors</p> |

^a Product contributors are represented by ■ and product components are represented by ●. Arrows represent contributions made by product contributors to the component. The thickness of the arrow indicates the relative number of contributions made by the contributor to the component.

Figure 4: Relationship between contribution network structure, contribution and contributor characteristics, and product quality

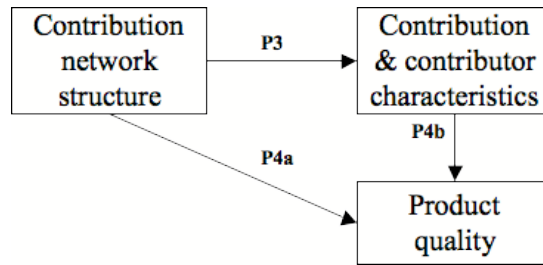


Table 1: Summary of measures

| <i>Measure</i> | <i>Description</i> |
|---|--|
| <i>Product contributions and contributors^a</i> | |
| Product contributions | A count of the number of changes to source code files stored in the repository during the specified time period |
| Product contribution size | The number of lines of code (LOC) added or deleted in a single product contribution |
| Product contributor | The person who makes a product contribution (author of the source code change) |
| <i>Modularity^b</i> | |
| Degree of modularity | Describes the interdependences among software modules within a specific application. Software engineering metrics are used to evaluate the design quality of object-oriented software based on the interdependence of components in the design (Martin, 1994). The measure is derived from the appropriateness of the types of interdependencies that exist among packages in the software. Refer to Table 2 for the exact formulas used in this calculation. Calculated for each package within the application and weighted by the size of the package (log of LOC). |
| <i>Contribution network structure^c</i> | |
| Degree centrality | The proportion of components to which the product contributor has made a contribution |
| Degree centralization | The variance of the degree centrality of member nodes and component nodes normalized by the maximum possible variance given the observed degree centralities |
| Network density | The average standardized degree centrality |
| <i>Product quality</i> | |
| Number of bugs | Static source code analysis tools are used to identify the number of bugs in the software. Results of three tools, FindBugs (http://findbugs.sourceforge.net/), JLint (http://artho.com/jlint/), and PMD (http://pmd.sourceforge.net/), were combined to provide an objective measure of product quality. A larger number of bugs or defects indicates lower product quality. |
| 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. I use McCabe's Cyclomatic Complexity measure, a standard metric used in software engineering (McCabe, 1976). McCabe's Complexity is strongly correlated with number of bugs (Dunn, 1984; Glover, 2006a; Glover, 2006b; Ward, 1989) |
| <i>Control variables^d</i> | |
| Weeks since project registered | Total elapsed time (in weeks) between the date the project was registered on SourceForge and the date of the specified release |
| Weeks since previous release | Total elapsed time (in weeks) between each release within a given project; this value is always zero for the first release of a given project |

^a Calculated for the time elapsed after the current release until the next release

^b Constructed from the source code included in each major software release (e.g. 1.x, 2.x) made by each project sampled

^c A weighted graph describing the contribution network will be created using the product contributor and product contributions measures, as well as source code characteristics. Member nodes represent product contributors and component nodes represent the source code packages within the product. A tie exists between a member nodes and a component node when the product contributor has made at least one product contribution to the particular package. Ties are weighted by the number of product contributions made by the product contributor to the particular package.

^d Used to capture the time element of each release since each OSS project has multiple releases over time

Table 2: Calculation of measure of modularity

| <i>Source code metrics</i> | |
|-------------------------------------|---|
| 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 |

These metrics measure the relationships between source code packages, and whether techniques of information hiding and module abstraction are used appropriately (Martin, 1994).

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)

D_{release} is used as a proxy for source code modularity. To calculate D_{release} for a release within an OSS 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):

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

for every package i in the release. Finally, D_{release} is subtracted from 1, so that larger values indicate higher degrees of modularity.

References

- Baldwin, C. Y., & Clark, K. B. (1997). Managing in an age of modularity, *Harvard Business Review* 75(5), 84-93.
- Baldwin, C. Y., & Clark, K. B. (2000). *Design rules: The power of modularity (Vol. 1)*. The MIT Press: Cambridge, MA.
- Blau, P. M. (1977). *Inequality and Heterogeneity*, New York: Free Press.
- Burt, R. S. (1992). *Structural holes: The social structure of competition*. Cambridge, MA: Harvard University Press.
- Butler, B. S., Sproull, L., Kiesler, S., & Kraut, R. E. (2007). Community effort in online groups: Who does the work and why? In Weisband, S. (Ed.), *Leadership at a distance*. Mahwah, NJ: Erlbaum.
- Dunn, R. H. (1984). *Software Defect Removal*. McGraw-Hill, New York.
- Freeman, L. C. (1979). Centrality in social networks: I. Conceptual clarification. *Social Networks*, 1, 215-239.
- Ghosh, R. A. (2003). Clustering and dependencies in free/open source software development: Methodology and tools. *First Monday*, 8(4), np.
- Glover, A. (2006a). In pursuit of code quality: Monitoring cyclomatic complexity. Retrieved 3/5/2007, from <http://www-128.ibm.com/developerworks/java/library/j-cq03316/>
- Glover, A. (2006b). In pursuit of code quality: Tame the chatterbox. Retrieved 3/5/2007, from <http://www-128.ibm.com/developerworks/java/library/j-cq06306/>
- Johnson, J. P. (2002). Open source software: Private provision of a public good. *Journal of Economic and Management Strategy*, 11(4), 637-662.
- Korson, T. D., & Vaishnavi, V. K. (1987). An empirical study of the effects of modularity on program modifiability. In E. Soloway & S. Iyengar (Eds.), *Empirical Studies of Programmers: First Workshop*. Norwood, NJ: Ablex Publishing, 168-186.
- Lei, D., Hitt, M. A., & Goldhar, J. D. (1996). Advanced Manufacturing Technology, Organization Design and Strategic Flexibility. *Organization Studies*, 10(4), 501-523.
- 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. (1991). Exploration and exploitation in organizational learning. *Organization Science*, 2(1), 71-87.
- March, J., & Simon, H. A. *Organizations* (2nd ed.), Blackwell Publishers, Cambridge, MA, 1993.
- Marciniak, J. J., (1994). *Encyclopedia of Software Engineering*, New York, NY: John Wiley & Sons, 131-165.
- Martin, R. (1994). OO design quality metrics: An analysis of dependencies. Retrieved 1/13/2006, from <http://www.objectmentor.com/resources/articles/oodmetric.pdf>
- McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, 2(4), 308-320.
- Oliver, P. E., & Marwell, G. (1988). The paradox of group size in collective action: A theory of the critical mass. II. *American Sociological Review*, 53(1), 1-8.
- Parnas, D. L., Clements, P. C., & Weiss, D. M. (1985). "The modular structure of complex systems," *IEEE Transactions on Software Engineering*, 11(3), 259-266.
- Proctor, C. H., & Loomis, C. P. (1951). Analysis of sociometric data. In Jahoda, M., Deutsch, M., & Cook, S. W. (Eds.), *Research methods in social relations*, New York: Dryden Press, 561-586.
- 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/>
- Rogers, E., & Kincaid, D. (1981). *Communication networks*. New York: Free Press.
- Sanchez, R., & Mahoney, J. T. (1996). Modularity, flexibility, and knowledge management in product and organization design. *Strategic Management Journal*, 17, 63-76.

- Schilling, M. A. & Steensma, H. K. (2001). The use of modular organizational forms: An industry-level analysis. *Academy of Management Journal*, 44(6), 1149-1168.
- Shaft, T. M., & Vessey, I. (2006). The role of cognitive fit in the relationship between software comprehension and modification. *MIS Quarterly*, 30(1), 29-55.
- Shaw, M. E. (1954). Group structure and the behavior of individuals in small groups. *Journal of Psychology*, 38, 139-149.
- Sullivan, K. J., Griswold, W. G., Cai, Y. & Hallen, B. (2001). The structure and value of modularity in software design, In the *Proceedings of the 8th European Software Engineering Conference*, Vienna, Austria.
- Ulrich, K. (1995). Role of product architecture in the manufacturing firm. *Research Policy*, 24, 419-440.
- Ward, W. T. (1989). Software defect prevention using McCabe's Complexity Metric. *Hewlett-Packard Journal*, Retrieved 3/12/2007 from http://findarticles.com/p/articles/mi_m0HPJ/is_n2_v40/ai_7180008
- Wasko, M., and Faraj, S. (2005). Why Should I Share? Examining Knowledge Contribution in Electronic Networks of Practice. *MIS Quarterly*, 29(1), 1-23.
- Wasserman, S., & Faust, K. (1994). *Social network analysis: Methods and applications*. New York: Cambridge University Press.