An Empirical Study on the Relationship among Software Design Quality, Development Effort, and Governance in Open Source Projects

Eugenio Capra, Chiara Francalanci, and Francesco Merlo

Abstract—The relationship among software design quality, development effort, and governance practices is a traditional research problem. However, the extent to which consolidated results on this relationship remain valid for open source (OS) projects is still an open research problem. An emerging body of literature contrasts the view of OS as an alternative to proprietary software and explains that there exists a continuum between closed and OS projects. This paper takes this perspective and hypothesizes that, as projects approach the OS end of the continuum, governance becomes less formal. In turn, a less formal governance is hypothesized to require a higher quality code as a means to facilitate coordination among developers by making the structure of code explicit and observable and, at the same time, facilitate quality by removing the pressure of deadlines from contributors. However, a less formal governance is also hypothesized to increase development effort due to a more cumbersome coordination overhead. The verification of research hypotheses is based on empirical data from a sample of 75 major OS projects. Empirical evidence supports our hypotheses and suggests that software design quality, mainly measured as coupling and inheritance, per se does not increase development effort, but represents an important managerial variable to implement the more open governance approach that characterizes OS projects, which, in turn, increases development effort.

Index Terms—Organizational management and coordination, qualitative process analysis, software complexity measures, software cost estimation, software quality concepts, structural equation modeling.

1 Introduction

The development process of Open Source (OS) projects is generally perceived as informal, geographically distributed, and voluntary. Compared with the governance of traditional software projects, this looser coordination is considered beneficial to software design quality. OS code is perceived to have higher quality and, in particular, to be more secure, bug free, and dependable [19], [39].

In previous literature, software design quality is viewed as an investment, as it involves a cost, but it provides a payoff by reducing unit maintenance effort over time, with an overall positive balance [98]. In closed-source projects, a short-term view is often favored, and companies may sacrifice quality in order to achieve other objectives, such as fulfilling their deadline commitments [98]. This causes a quality degradation of source code, which can reach a break-even point where replacement becomes convenient with respect to the maintenance of old code [103]. As a consequence, formal governance practices are often perceived as detrimental to software design quality and, conversely, a more open governance—as in OS projects—is considered advantageous. Even though some OS projects adopt formal schedules for their release management [84], numerous other OS projects are managed informally, without the pressure of strict deadlines. These less formal governance practices allow OS administrators to take a long-term view, invest in quality, and reduce maintenance effort over time. This general perception of lower OS costs is further reinforced by the fact that OS development is a cooperative effort and costs can be shared within a broader community.

Contrary to the common perception, a number of OS projects are actually launched and maintained by commercial companies, which view OS as a way to pursue their revenue objectives [40], [46], [47], [94]. Companies can be assumed to develop software that they consider interesting from a commercial point of view. The literature suggests that, from a business perspective, the coordination effort of distributed, informal, and voluntary groups can be demanding [45]. Fogel [41] notes that building an OS community for commercial purposes involves new coordination roles, such as community managers and evangelists (for example, Solid,[1] a formerly proprietary database engine, hired a team of community managers when it was released under an open license in October 2006). The development process can be slower [41], [46] and new roles can have a full-time project involvement. These considerations raise doubts on the cost benefits that OS projects can credit to their leading companies.

The relationship among software design quality, development effort, and governance practices is a traditional research problem [98], [108]. Governance is a complex concept and impacts a wide range of measurable variables. This paper focuses on the governance variables that distinguish the governance practices of OS projects. An

• *The authors are with the Department of Electronics and Information, Politecnico di Milano, via Ponzio 34/5, I-20133 Milano, Italy. E-mail: {capra, francala, merlo}@elet.polimi.it.*

Manuscript received 1 Oct. 2007; revised 27 June 2008; accepted 30 July 2008; published online 5 Aug. 2008.
Recommended for acceptance by P. Devanbu.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2007-1-0281.
Digital Object Identifier no. 10.1109/TSE.2008.68.

1. http://www.solidtech.com.

emerging body of literature contrasts the view of OS as an alternative to proprietary software and explains that viewing OS as a definite and coherent phenomenon can lead to misperceptions [13]. This *contingent* perspective indicates that there exists a continuum between closed and OS projects [25]. Our governance variables measure where a project is positioned along this continuum. The extent to which consolidated results on the relationship among software quality, governance practices, and development effort remain valid as projects move toward the OS end of the continuum is still an open research problem.

This paper provides empirical evidence to support the idea that software design quality is an enabler of more *open* governance approaches, i.e., governance approaches that distinguish projects closer to the OS end of the continuum. Our data also indicate that a more open governance approach increases development effort due to a more cumbersome coordination overhead. Overall, this paper's empirical findings suggest that quality, mainly measured in terms of coupling and inheritance, is a must in OS projects to enable more open governance mechanisms, which, in turn, increase development effort, while quality per se does not increase effort.

The presentation is organized as follows: Section 2 reviews the literature on software design quality metrics, effort models, and governance mechanisms. Section 3 discusses our research hypotheses, while Section 4 presents the empirical testing methodology. Section 5 explains the statistical approach and reports the results of empirical testing. Section 6 provides an in-depth discussion of findings. Threats to validity and future work are discussed in Section 7.

2 Related Work

This section reviews the main research works that have addressed the relationship among software quality, governance practices, and development effort. The review starts from a discussion of software design quality and then analyzes the cost implications of quality in Section 2.2. Governance is finally reviewed in Section 2.3 to show how it can influence the consolidated results on the relationship between quality and effort in OS projects.

2.1 Software Design Quality

The measurement of software quality is traditionally based upon 1) complexity and 2) design quality metrics. The first research contributions were aimed at providing operating definitions and metrics of software complexity, focusing on the analysis of the code's information flow. Cyclomatic Complexity [81], Software Science [51], and Information Flow Complexity [54] represent the most widely used metrics from this early research.

Over time, design quality has become of increasing importance to cope with the continuously growing size of software systems. Research started to distinguish between the complexity due to poor design quality and the inherent complexity of software due to requirements [89], [106]. The main contribution of these studies was to show that design quality is necessary to handle the complexity caused by challenging requirements.

With the advent of the object-oriented programming paradigm, coupling, cohesion, inheritance, and information hiding have been identified as the basic properties of software design quality [30], [36], [97], [101]. Based on these four basic properties, a number of metrics have been proposed to evaluate the design quality of object-oriented software. The most widely known metrics were first proposed by Chidamber and Kemerer [31] (WMC, NOC, DIT, RFC, LCOM, and CBO) and by Brito e Abreu [22] (COF, PF, AIF, MIF, AHF, and MHF). These milestone contributions have started a lively debate within the software engineering community on the consistency and generality of such metrics [9], [32], [52], [57], [96].

The literature concurs that no individual metric provides a complete assessment of quality since there exist multiple aspects of software design that can affect overall quality [76]. A fundamental issue is how to select a coherent subset of metrics that provide an overall assessment of software design quality. The ISO 9126 and ISO 25000 standard series identify 27 independent dimensions of quality [62], [63]. Code-based metrics typically lie on the maintainability, reusability, and portability dimensions [21], [56]. Although it has been noted that there exist mutual relationships among dimensions and related metrics [79], there is no scientific evidence of the impact of code-based metrics on other ISO quality dimensions.

Most research has focused on code-based metrics of maintainability. A consistent body of literature has analyzed the relationship between object-oriented code-based quality metrics and fault-proneness, which represents a fundamental indicator of corrective maintainability (Subramanyam and Krishnan [101] provide a detailed review of the literature). Other classes of maintainability, including adaptive, preventive, and perfective maintainability, are recognized to impact on the cost of evolutionary maintenance [29], [78]. Evolutionary maintenance has been found to be responsible for more than 75 percent of the total cost of maintenance [28]. As a consequence, most results on the relationship between code-based quality metrics and maintainability focus on effort and are reviewed in the next section.

2.2 Development Effort Models

The literature provides several methodologies and models for software cost estimation [2], [8], [16], [17], [102] and for cost benchmarking [37], [66]. The typical goal of most of these models is to predict the effort required to develop or to maintain a software system. Prediction is based on measures of various software properties, such as size or complexity, or on the analysis of productivity drivers measured by change management systems [85], [91], such as CVS or SVN. For example, Mockus and Graves [85] propose an approach to quantify the extent to which properties of changes to software, such as bug fixing or integration of new functionalities, affect the implementation effort.

Initial development costs are typically distinguished from the cost of subsequent maintenance interventions. The latter have been empirically found to account for about 75 percent of the total cost of an application over its entire life cycle [6], [78]. A consolidated body of literature studies the impact of software design quality on maintenance effort. As noted by Slaughter et al. [98], the assumption made by many of these research works is that maximizing quality is economically convenient as the reduction of maintenance

TABLE 1
Empirical Studies on the Relationship between Software Design Quality and Maintenance/Development Effort

Study	Metric of effort	Metric of design quality	Main findings
Li and Henry, 1993 [77]	Number of lines changed (added, removed, modified) in a class	Chidamber and Kemerer's [31] quality metrics	Maintenance effort increased by higher values of Chidamber and Kemerer's [31] metrics
Binkley and Schach, 1998 [15]	Number of hours spent in maintaining each system module	Coupling	Greater coupling increases maintenance effort
Chidamber et al., 1998 [32]	Number of hours of development time	Coupling and cohesion	Greater coupling and lack of cohesion increase development effort
Banker et al., 1998 [6]	Time required to accomplish a maintenance intervention	Complexity	Greater software complexity increases maintenance effort
Banker and Slaughter, 2000 [5]	Evolutionary maintenance effort	Complexity	Data complexity (number of data elements per unit of functionality) increases evolutionary maintenance effort
Harter et al., 2000 [53]	Person-months required to develop the software product	Process maturity and quality dimensions	Greater process CMM maturity associated with greater software quality reduces development effort
Darcy et al., 2005 [34]	Recorded time to complete given comprehension and maintenance tasks	Coupling and cohesion	Greater coupling and lack of cohesion increase comprehension and maintenance effort

costs largely overcomes the increase of initial costs of developing a higher quality software artifact.

The *propagation cost model* proposed by MacCormack et al. [80] investigates the impact of coupling on maintenance costs, arguing that the degree of interdependence among different modules can be used as a proxy of costs. Tan and Mookerjee [103] analyze the effect of software entropy on maintenance costs on a sample of closed source software projects. Software entropy has been originally introduced by Bianchi et al. [14] and represents an overall measure of the degree of quality degradation induced by maintenance interventions, which tend to make code increasingly "chaotic." Tan and Mookerjee [103] hypothesize that maintenance costs grow exponentially with entropy and that there exists a break-even time when reimplementation becomes economically convenient. Li and Henry [77] and Harter et al. [53] prove that maintenance effort is significantly increased by higher values of the quality metrics defined by Chidamber and Kemerer [31]. Binkley and Schach [15], Chidamber et al. [32], and Darcy et al. [34] show a correlation between increasing values of different metrics of coupling and both development and maintenance effort, whereas Banker et al. [6] and Banker and Slaughter [5] point out that effort is also influenced by complexity. Table 1 summarizes the key studies focusing on the evaluation of the impact of software quality on development and maintenance effort. Clearly, most contributions focus on the direct relationship between quality and effort. Slaughter et al. [98] provide evidence showing that it is more profitable to invest in quality early in a project, in order to maximize the benefits of quality improvements. Further, they suggest that it is most cost-effective to implement software quality improvement initiatives specifically aimed at reducing effort in both development and governance. This indicates that the relationship between design quality and development effort should be analyzed by taking into account other variables and, in particular, governance. Other works that study the maturity of the software development process confirm the importance of considering governance when analyzing the relationship between quality and effort [64], [54].

2.3 Governance

The governance of software projects is defined as the complex process that is responsible for the control of project scope, progress, and continuous commitment of developers [93]. Governance is recognized to have a key role in enabling software project success [108] (a comprehensive summary of the literature on governance is provided by Kirsch [69]). Different from manufacturing, software processes are more human intensive and creative and have unique characteristics that lead to a wide variance in individual productivity. Human labor is difficult to standardize and control [73]. Cooperation among codevelopers is crucial, but, contrary to what happens in manufacturing, hierarchical control structures may inhibit personal creativity, instead of increasing productivity [18].

Historically, structured and disciplined governance approaches have been associated with greater efficiency also in the software engineering literature. For example, Basili and Weiter [7] provide empirical evidence indicating that disciplined development methodologies based on structured programming increase development efficiency. In general, the traditional waterfall software development model is highly structured and relies on detailed development plans. More recently, formal frameworks such as the Capability Maturity Model (CMM), Bootstrap, and SPICE [33], [55] define metrics to assess the impact of governance and formal organizational practices on the performance of the software development process.

New agile software development models rely on a looser approach to governance. Agile methods focus on individuals as opposed to processes, on code as opposed to paperwork, on output as opposed to deadlines and

planning [18]. In 1999, Beck [11] proposed the eXtreme Programming method, which explicitly aims at eliminating as much non-code-developing activities as possible. These methods foster informal governance mechanisms and autonomous coordination among developers.

In this respect, even though OS development cannot be identified with an agile form of development, significant similarities have been found in several areas [72]. From a general standpoint, OS clearly represents a software development model consistent with a less formal approach to governance. OS projects leverage cooperative development, innovation, and informal leadership [40], [47], [80], [99]. OS communities exploit the distributed intelligence of all participants and lack the traditional hierarchical structure and governance roles. These communities promote new coordination mechanisms based on consensus, meritocracy, and the so-called "do-ocracy" [105], i.e., decisions are made by the developers who more actively contribute to the project.

A recent stream of literature explains that there exists a continuum between OS governance practices and the traditional approaches to the governance of closed-source projects [23]. Although governance is a complex concept, this paper focuses on the variables that allow us to position a project along this continuum of governance practices. These variables are thoroughly explained in Section 4.3 and are the result of an empirical analysis of current OS governance practice. However, the literature indicates that different typical categories of hybrid projects can be recognized along the continuum [94]. *Commercial OS* projects represent the most frequently cited hybrid category that corresponds to a number of projects owned by companies, usually not available from OS online communities but released on an open license according to the OSI standard (for example, MySql, EnterpriseDB, SugarCRM, Jaspersoft, Zimbra, Alfresco, Funambol). *Community OS* is another hybrid category, where code is fully open, there is no single company owning the project, but some or most developers can be employed by a company and paid to take active part in the community and develop specific components of a software program according to their company needs. Clearly, releasing code on an open license is only one of the governance variables that affect the position of a project along the continuum. However, for the sake of simplicity, we label the two ends of the continuum as *fully closed* and *fully open* and use the term *degree of openness* to refer to the position of a project on the governance continuum [23], [24].

3 Research Hypotheses

In previous literature, increasing software design quality is viewed as a costly activity that pays back in the long term by reducing the cost of subsequent maintenance interventions [98]. Companies usually take a short-term perspective and tend to develop code faster at the expense of quality, which, in turn, tends to decrease over time [103]. Tan and Mookerjee [103] observe that the deterioration of quality over time leads to a break-even time when a short-term perspective becomes economically inefficient and companies should invest in quality. This can be obtained either by replacing an application with new software of higher quality or by launching a maintenance initiative aimed at increasing quality without necessarily developing new functionalities, commonly referred to as *refactoring* [43], [83].

In OS applications, these phenomena are difficult to observe. Some projects become *inactive* when they reach the end of their lifecycle and, until then, they are continuously maintained. However, projects reach their end for a number of reasons that may not be related to quality deterioration. For example, *solo* projects, i.e., projects launched by individual programmers, are often active for a very short period of time and come to an end due to lack of interest from the OS community.

The most successful projects, such as Linux and PostgreSQL, are still active even though they are considered mature. Long-lasting projects undergo multiple releases, traced by versioning systems. As a general observation, the release of a new version may or may not coincide with a refactoring. Koch [70] has noted that, in OS projects, refactoring tends to be a continuous process and developers allocate time and effort to quality improvements when needed. In a previous paper [26], we have studied the refactoring process of a sample of 95 OS applications (1251 versions) from *SourceForge.net*. Empirical analyses showed that the number of versions between two subsequent refactorings is highly variable. On average, a significant quality improvement can be observed in 40 percent of the total number of versions, while Tan and Mookerjee [103] indicate that, in their sample of closed source applications, refactorings occur in about 10 percent of versions.

Previous literature indicates that the cost benefits of quality improvements are reaped over time. In contrast, in OS projects, refactoring is a continuous effort and the cost benefits of quality improvements should be visible even in the short term. Previous literature provides only partial evidence to demonstrate that quality investments have a positive balance [98]. Based on their theoretical model, Tan and Mookerjee [103] suggest that quality investments represent a zero-sum game. However, the only clear empirical result is that quality involves an investment and, in the short term, it represents a cost. OS projects challenge this result since continuous refactoring practices should release similarly continuous cost benefits. Therefore, we posit that the short-term investment costs of quality need further verification in OS projects and formulate H1, accordingly. H1 may not be verified, showing a negative or no correlation, depending on the short-term balance of quality investments in OS projects.

***H1**: A development initiative that increases software design quality has a greater unit development effort than a development initiative that does not increase software design quality.*

As discussed in Section 1, open code is commonly perceived to be of higher quality compared to proprietary code. The common perception is that the looser governance approach of open projects removes the pressure of deadlines and encourages the individual motivation of developers toward the production of a unique artifact [58].

A recent stream of literature states that a higher design quality is necessary to enable cooperation among distributed and looser groups of developers [80], [99]. These authors put forward that a higher design quality is not a consequence but rather a requirement of more open governance. A fundamental goal of face-to-face meetings is to share tacit knowledge. When face-to-face meetings are not possible or rare, knowledge must be made explicit. The

TABLE 2
Metrics of Code Design Quality

Metric	Definition	Theoretical definition	Operational definition
CBO (Couplings Between Objects)	Number of distinct classes to which a given class is coupled, excluding inheritance relationships	CK1994 [31]	Basili1996 [9]
COF (COupling Factor)	Number of relations in pair wise sets of classes	Brito e Abreu1995 [22]	Harrison1998 [52]
WMC (Weighted Methods per Class)	Weighted method complexity for a class	CK1994 [31]	CK1994 [31]
DIT (Depth of Inheritance Tree)	Maximum depth of the inheritance graph of each class	CK1994 [31]	Basili1996 [9]
NOC (Number Of Children)	Number of immediate subclasses of a given class	CK1994 [31]	Basili1996 [9]

quality of code can be seen as a means to facilitate knowledge sharing by making the structure of code explicit and observable [104]. For example, a modular structure of code facilitates the job allocation and the coordination among geographically distributed programmers [80]. Empirical evidence suggests that, in OS projects, most work is performed by a small group of core developers [99], but often core developers do not work in the same place and can benefit from higher quality. Moreover, a modular structure of code is required to easily integrate the high number of small contributions from the whole community. Ignoring or misusing these contributions may result in demotivating the community and losing momentum [41].

Overall, common belief views quality as a consequence of looser governance and the literature provides empirical evidence indicating that quality is an enabler of more open governance. Both views imply a causal relationship between quality and a higher degree of openness in the governance practices.[2] This leads us to our second research hypothesis:

***H2**: A higher degree of openness in the governance practices leads to greater software design quality.*

The literature does not provide empirical evidence on the relationship between a more open approach to governance and development effort. As discussed in Section 2.3, the traditional software engineering literature views formal (i.e., closer) governance as a way to guarantee project effectiveness (i.e., output and timeliness) rather than cost efficiency. However, the general management literature shows a positive effect of formal governance on cost reduction [61].

From a cost perspective, it seems reasonable to hypothesize that a more open approach to governance leads to higher costs, due to a number of coordination overheads. Community OS developers may never meet in person and interact entirely by e-mail, forums, and IRC channels. This makes the decision-making process slower and possibly troublesome as discussion is rarely synchronous. Community members can reside in different geographical locations, with significant time zone differences that can delay decision making and cause reworks. OS projects tend to be very informal, without roadmaps, deadlines, and tight schedules. Whereas this may lead to higher design quality, it may also translate into lower efficiency due to a lack of planning [86]. For example, multiple developers may autonomously decide to work on the same set of functionalities, with a consequent duplication of tasks. Alternatively, they may decide to develop a plug-in that is not consistent with subsequent versions of the same application and, therefore, is of little use. These considerations lead us to our third research hypothesis.

***H3**: A higher degree of openness in the governance practices leads to a higher unit development effort.*

4 Variable Definition and Operationalization

In this section, we provide the formal definition and operationalization of the variables used in our study. Sections 4.1, 4.2, and 4.3 discuss software design quality, effort, and governance, respectively.

4.1 Design Quality Metrics

Table 2 shows the operating definition of the metrics of software design quality used in this study along with the research papers where 1) they have first been presented and 2) they have been provided the operating definition adopted in this paper. The metrics reported in Table 2 represent reference quality parameters that are widely used and have also been included in a number of CASE tools, including JBuilder and Rational Rose. Overall, these metrics provide measures of *coupling* and *inheritance*. As discussed in Section 2.1, there exists a third set of fundamental quality metrics measuring information hiding. These metrics are not considered since they primarily impact software correctness and dependability [44], which, in turn, are important effectiveness metrics but have a less clear impact on efficiency, including effort, and are rarely considered in the software economics literature.

Quality metrics were measured by means of a tool developed ad hoc performing static analyses of Java bytecode. The consistency of results was verified by cross-checking the values computed by our tool with the values obtained by analyzing the corresponding source code with several commercial tools, showing that bytecode-based measurements perform as well as source code-based measurements.

2. Note that, when a variable, say A, is an enabler of another variable, say B, A is a dependent variable of B and B leads to A. Intuitively, A is a necessary condition for B or, in logical terms, "if B then A." The opposite is not true as A is not a sufficient condition for B.

TABLE 3
Summary Statistics from the Effort Survey

On-line community	Team member class	N (total)	n (active)	Active average hr/week	St. Dev.	Conf. int. (α/2= 0.025)	α (avg)	β (avg)
SourceForge.net	Administrator	32	29	8.09	±9.65	±4.24	0.17	0.91
	Developer	102	56	8.62	±12.55	±3.86	0.22	0.55
Apache.org	Administrator	34	34	20.45	±18.60	±7.49	0.36	1.00
	Developer	82	65	12.10	±13.19	±3.76	0.21	0.79
Tigris.org	Administrator	104	58	10.30	±12.26	±3.71	0.19	0.56
	Developer	135	68	11.39	±12.61	±3.51	0.20	0.50
Commercial OS	Administrator	-	-	35.00	-	-	0.64	1.00
	Developer	-	-	35.00	-	-	0.64	1.00
GLOBAL	Administrator	170	121	12.48	±14.47	±2.99	0.22	0.71
	Developer	319	189	11.03	±12.79	±2.10	0.20	0.59

4.2 Development Effort Metrics

Development effort is measured in person-days. We define the development time $t_k(i)$ of version i of application k as the number of days elapsed between the release of version i and previous version $i-1$. Development effort $E_k(i)$ measured in person-days is obtained by multiplying development time $t_k(i)$ by the following correction factors:

- $n_k(i)$, the number of project administrators or developers;
- α_k, the average fraction of time (on a seven-day week) each contributor (administrator or developer) devotes to the project;
- β_k, the percentage of active members of the project team.

Different correction factors are used for project administrators and developers, leading to the following complete expression for development effort:

$$E_k(i) = t_k(i) \cdot \left[\alpha_k^{admin} \cdot \beta_k^{admin} \cdot n_k^{admin}(i) + \alpha_k^{devel} \cdot \beta_k^{devel} \cdot n_k^{devel}(i)\right]. \quad (1)$$

The values of the correction factors were empirically estimated by surveying 3,346 OS project administrators and developers involved in 268 different projects, including our sample of projects.[3] The questionnaire was customized to individual contributors in order to gather the actual time devoted by each respondent to each project he or she was involved in. A total of 489 contributors (310 of whom currently active) responded to our survey, allowing us to estimate the exact correction factors for about 25 percent of the projects in our sample. Average values were used for the rest of the projects. Different average values of correction factors were computed for different online communities.

Table 3 presents a summary of the overall results of the survey.

Average α values were obtained by considering a 40 hour working week, consistent with the definition of the Full-Time Equivalent (FTE) working week provided by the US Office of Management and Budget.[4] For commercial OS projects, our survey indicated that only 90 percent of contributors work full time and we considered a 35 hour working week, accordingly.

When available, the number of administrators $n_k^{admin}(i)$ and developers $n_k^{devel}(i)$ were gathered from the homepage of each project. For commercial OS projects, we used the values obtained during the face-to-face interviews. For a small number of projects (less than 10 percent of our sample), no information was provided and we estimated the number of developers as the number of active team members with commit right.

The number of new methods (or C functions/procedures) $\Delta M_k(i, i-1)$ added to version i with respect to previous version $i-1$ of application k is defined as the set theoretical difference between the sets of method (or C functions/procedures) signatures of versions i and $i-1$. A tool developed ad hoc was used to extract the complete method (or C function/procedure) signatures of each version and compute the set theoretical difference. Unit maintenance effort $e_k(i)$ is then defined as the ratio of maintenance effort $E_k(i)$ to the number of new methods (or C functions/procedures) $\Delta M_k(i, i-1)$:

$$e_k(i) = E_k(i)/\Delta M_k(i, i-1). \quad (2)$$

As an example, let us consider two versions of the JEdit project, v.4.3pre6 and v.4.3pre7, released on 4 August 2006 and 3 October 2006, respectively. From the release dates, it is clear that the development activities of version v.4.3pre7 lasted 60 days. The project's homepage on SourceForge.net reports that the project team is composed of nine

3. The complete questionnaire is available from http://www.fondazione politecnico.it/pagine/pagina.aspx?ID=Survey01001&L=IT.

4. Note that $t_k(i)$ measures the elapsed time in days between two subsequent releases, whereas the effort is measured in person-days, where a person-day is defined as the average work performed in one day by one person who works 40 hours a week. The parameter α converts elapsed days into person-days. For example, if a developer declares working 40 hours a week, then $\alpha = 5/7 = 0.71$, as weekends are not to be accounted for. $\alpha = 1$ only if a person declares working 56 hours a week (8 hours for seven days).

TABLE 4
Operating Definition of the *Code* Dimension of Governance

Value	Description
1	100% of the code is closed
2	<=80% of the code is open
3	>80% of the code is open
4	100% of the code is open

administrators and 110 developers, thus $n_{JEdit}^{admin} = 9$ and $n_{JEdit}^{devel} = 110$. From Table 3, $\beta^{admin} = 0.91$, $\beta^{devel} = 0.55$, $\alpha^{admin} = 0.17$, and $\alpha^{devel} = 0.22$ for the *SourceForge.net* online community. Based on these data, the total development effort for version v.4.3pre7 is

$$E_{JEdit}(v.4.3pre7) = 60 \cdot [0.17 \cdot 0.91 \cdot 9 + 0.22 \cdot 0.55 \cdot 110] = 882.14 \text{ person-days}.$$

Since the number of new methods of v.4.3pre7 is $\Delta M_k(v.4.3pre7, v.4.3pre6) = 974$, the unit development effort is

$$e_{JEdit}(v.4.3pre7) = 882.14/974 = 0.91 \text{ person-days/method}.$$

Note that existing effort metrics, such as those proposed by Mockus and Graves [85] or Ramil [91], are defined based on data derived from change management systems. Our approach is to ask developers and administrators the actual amount of time spent in development or maintenance activities. The effort metrics based on CVS or Subversion data involve an error since they do not account for the idle time between subsequent commits or subversions or for the fraction of time spent in development activities by contributors. The main advantage of our approach is to provide a direct measure of effort that does not depend on code-based metrics and, therefore, allows the analysis of correlations with code-based metrics, as required by our hypotheses.

Our effort metric involves an error if the development of a version does not start immediately after the end of the previous version. However, the branching/tagging practices that are widely adopted in OS projects imply that development activities are almost continuous [92]. From our survey, nearly 67 percent of respondents answered that 1) there is no idle time between the development of two subsequent versions or 2) development starts within the following five working days. Based on these results, we have decided not to correct for the idle time between versions.

4.3 Governance Metrics

Our metrics of governance aim at positioning a project along the continuum between fully open and fully closed governance practices. They represent the result of a preliminary empirical analysis conducted by means of individual face-to-face interviews with 25 project managers of major software projects along the continuum (the list of these projects can be found in online Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2008.68). Project managers were asked which governance dimensions were more significant to measure the degree of openness of a software project. We identified four fundamental governance dimensions: code, contribution, project leadership, and working practices. These dimensions were selected as 1) they were cited by all project managers and 2) they had the highest positions in the ranking of all cited dimensions (a thorough discussion of this preliminary empirical analysis can be found in [23] and [25]). Each dimension is measured on an ordinal scale (1 to 4), where 1 and 4 indicate a fully closed and a fully open approach, respectively. The operating definition of each dimension is provided below.

4.3.1 Code

This dimension indicates the percentage of code (measured in SLOC, according to Park [90]) that is available under an OS license (as defined by the OSI [88]). Commercial OS projects can have both open and closed (corporate) versions. However, they usually keep part of the code closed. A community-based OS project is 100 percent open. Table 4 shows our metric of openness along this dimension. The 80 percent threshold has been chosen because, in all projects that adopt dual licensing, the commercial version of the application overlaps with the open version for at least 80 percent of the code [23]. Projects with less than 80 percent of open code typically release only specific functionalities with an open license. For example, Yahoo! releases its Web mailer as OS, although its applications are mostly closed source.

4.3.2 Contribution

This dimension indicates the amount of voluntary code development. Most commercial OS companies resemble proprietary software companies in their reliance on paid development [94]. In community-based OS projects, code is usually developed on a voluntary basis. However, as noted in Section 2.3, contributors may be employed by a company that takes part in the project to accomplish specific tasks (e.g., to implement a new feature or to fix a bug). A significant difference between employed and voluntary developers is that the former have to abide by the plans of their employers, whereas the latter are free to work according to their own objectives.

Table 5 shows our metric of the degree of openness along this dimension. The 80 percent threshold has been chosen as, in commercial OS projects, the percentage of code committed by voluntary programmers is typically below 20 percent [23]. In community-based projects, less than 50 percent of code is developed by employed developers [23].

TABLE 5
Operating Definition of the *Contribution* Dimension of Governance

Value	Description
1	100% of the code is developed by employed developers
2	>80% of the code is developed by employed developers
3	>50% of the code is developed by employed developers
4	Most of the code is developed by voluntary developers

TABLE 6
Operating Definition of the *Project Leadership* Dimension of Governance

Value	Description
1	The development process is formalized and led by an organization (company, institution, or committee) that has a predominant leadership role, makes decisions, and defines a formal schedule for the project.
2	The development process is formalized and led by an organization (company, institution, or committee) or a benevolent dictator that define a formal schedule for the project. The community is strongly involved in the decision process, although final decisions are made the leader.
3	The community is ruled by common principles and formal rules. Decisions are made mainly by voting or by governance bodies directly elected by contributors.
4	The community lacks a formal organization and governance bodies. Decisions are made by informally by discussing issues.

TABLE 7
Operating Definition of the *Working Practices* Dimension of Governance

Value	Description
1	Developers work in the same site, communicate by traditional means, and have regular physical meetings.
2	Most developers work in the same site and have regular physical meetings. Teams may use virtual communication tools.
3	The community is dispersed and most developers communicate through virtual communication tools. Subsets of developers may work in the same location and meet regularly.
4	The community is dispersed and all developers communicate electronically. Physical meetings are totally absent or very rare (1-2 per year).

4.3.3 Project Leadership

This dimension indicates the degree to which the coordination structure of a project is hierarchical. Commercial OS projects are led by a company, which usually defines a roadmap and sets schedules. Companies might also play a significant role in guiding and managing community projects. There exist community OS projects that are indirectly governed by a predominant company which defines the roadmap of the project and leverages the community to reach its own goals [25]. Communities may be led not only by a company but also by a foundation or by an independent committee. Some projects, such as Linux, are managed by a "benevolent dictator": Participation and discussion are fostered, but final decisions are made by the project leader or an entrusted committee. Even fully open communities often implement bottom-up coordination mechanisms. Decisions are made through voting systems or by governance bodies which are directly elected by active contributors (as in the Apache project [38]). Less formal communities adopt the *lazy consensus approach*, i.e., issues are discussed within forums and mailing lists and decisions are made when nobody has anything to argue [105]. Our metric along this dimensions aims at positioning these approaches to leadership on an ordinal scale, as shown in Table 6. This scale is consistent with the projects that we have analyzed in our preliminary empirical analysis (see [23]).

4.3.4 Working Practices

This dimension indicates the degree to which the working and communication practices of a project are geographically distributed and virtual. Proprietary software projects and many commercial OS projects primarily rely on a closed community working for a single employer, often in close physical proximity. A community OS project can lack a corporate management structure and tends to have a geographically dispersed network of contributors. With little funding to support face-to-face meetings, these projects heavily rely on virtual collaboration tools. Table 7 presents our metric of the degree of openness along this dimension.

4.3.5 Observations

Our framework provides a qualitative assessment of the degree of openness and, accordingly, scales are ordinal.

Four-level scales allow us to represent all of the information that has emerged from our preliminary analysis to assess the degree of openness of a project. Assessing governance by means of ratio variables is not only difficult but may also be misleading [48], [100]. Our framework is intended for comparing projects, rather than providing absolute assessments. Our statistic methodology described in Section 5 is designed to be applied to ordinal variables, consistent with Briand et al. [20]. The data collection process is also consistent with our metrics as it is based on face-to-face or telephone interviews of key project personnel (see Section 5.1).

4.3.6 Examples

We discuss two examples to explain how our governance metrics apply to software projects.

MySQL (www.MySQL.com) is distributed by MySQL, AB, now acquired by Sun Microsystems, both as a free GPL-licensed community edition and as a commercial-licensed enterprise edition. The code base of the two editions is almost the same, but the enterprise edition includes a few additional functionalities. Consequently, it scores 3 on the *code* dimension. Even though code is open, it is mainly developed (99 percent) by employees of the company. The community is invited to submit new code, which is reviewed according to strict internal standards before it is accepted. However, this is quite rare, given the size and complexity of the code base. This accounts for score 2 on the *contribution* dimension.

MySQL, as a company, controls the governance of the project. The corporate culture is very open to discussion, which is fostered by means of online communication tools, such as blogs, wikis, and forums, but MySQL, as a traditional software house, makes the final decisions. Thus, we assign a score of 2 to the *project leadership* dimension.

Although MySQL is managed as a traditional company, its working practices resemble those of community projects. Developers are located in 26 countries around the world and work from home, meeting only once or twice a year. They mainly communicate through asynchronous tools, such as highly specific internal IRC channels, shared task lists, and e-mails, to overcome time zone differences. This accounts for score 4 on the *working practices* dimension.

OpenOffice (OpenOffice.org) is a widely used OS office suite with more than 100 million downloads. Code is 100 percent open, leading to score 4 on the *code* dimension. Although everybody can contribute to the project and can earn commit right, Sun Microsystems and IBM have historically contributed almost 90 percent of the code, paying more than 90 developers for their work. Other companies, such as RedHat and Novell, also contribute to development. This leads to score 2 on the *contribution* dimension.

The community has a very structured governance model, based on a Community Council and an Engineering Steering Committee. The Community Council is constituted by members of the community, but is strongly influenced by the companies that control the project. The project has a clear and shared roadmap. All of these factors lead to score 2 on the *project leadership* dimension.

Communication within the community mainly takes place on mailing lists and IRC channels. However, most of Sun's developers work in Hamburg and meet daily. As a result, issues are often discussed in face-to-face meetings. Occasional cross-corporation meetings are held a few times a year. Consequently, OpenOffice scores 3 on the *working practices* dimension.

5 Methodology and Results

The next two sections provide a description of the data sample (Section 5.1) and statistical approach (Section 5.2). Testing results are then reported in Section 5.3.

5.1 Data Sample

Our sample is comprised of 75 OS software projects. The most influential and well-known stand-alone projects (i.e., projects that do not belong to any online community), both commercial (e.g., *Eclipse*, *Jaspersoft*, *MySQL*, *SugarCRM*) and community-based (e.g., *JavaDB*, *OpenOffice*, *Mozilla*), are included in the sample. The sample also includes a selection of medium to large community-based projects hosted on the most widely known software online communities, such as *SourceForge.net*, *Apache.org*, *Tigris.org*, *Java.net*, and *Google Summer of Code*. Since mining online communities can lead to controversial results because of the varying quality of available data [59], we carefully selected community-based projects by applying the following criteria:

- *Project maturity*: beta status or higher. Less mature projects were excluded because of their instability and low significance.
- *Language*: Java, in order to make design quality metrics comparable.
- *Availability of source code*: Projects were selected only if at least part of their source code was available for download (i.e., at least score 2 on the code dimension, see Section 4.3) in order to be able to assess our quality metrics on at least part of the code.
- *Version history*: at least five versions released (minor, major, or release candidate).
- *Number of active contributors*: at least three active contributors.

A complete list of the projects in our sample can be found in the online Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2008.68. Note that quality metrics were calculated for Java applications only, that is 48 out of 75 applications. Therefore, hypotheses H1 and H2 involving quality were tested on this subset, while H3, addressing the relationship between cost and governance, was tested on the entire sample of 75 applications. The projects in our sample vary by age, size, adoption, and domain. Tables 8 and 9 show how these parameters vary across the sample.

As discussed in Section 4.3, our governance metrics were defined based on a preliminary empirical analysis of 25 case studies (see online Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2008.68). Only 12 out of these 25 projects were included in the final sample, while 13 projects were excluded as both cost and quality data were missing. Our governance metrics are defined on

TABLE 8
Age, Size, and Adoption of Projects in the Sample

Variable	Minimum value	Average value	Median value	Maximum value	Project with maximum value
Age [year]	<1	8	5	30	Ingres
Size [number of developers]	3	78	11	1100	OpenSolaris
Size [kSLOC]	1	670	103	6 000	OpenOffice
Adoption[5] [thousand of downloads]	2	25 000	2 350	200 000	BerkeleyDB

[5] *The number of downloads may not coincide with the number of users as people may download several versions of the same application.*

TABLE 9
Domain Distribution of Projects in the Sample

Domain	Percentage
Software development	32%
Database	15%
Communication	11%
Infrastructural applications	11%
Office automation	8%
Content management	7%
CRM and Business intelligence	4%
Other (Education, Graphics, Scientific, Security)	12%

TABLE 10
Summary Statistics on Survey Responses for Governance Dimensions

Dimension	1	2	3	4
Code	0%	3%	12%	85%
Contribution	16%	32%	4%	48%
Project leadership	18%	18%	46%	18%
Working practices	4%	16%	28%	52%

four-point ordinal scales. The data collection process was consistent with this qualitative scale as it was based on face-to-face or telephone interviews of key project personnel, namely, community managers, quality assurance managers, vice presidents of engineering, committers, and project leaders. All key roles were interviewed for each project and responses were triangulated. Whenever inconsistencies were found, recalls were made to find an agreement. If a single key person was available, especially in smaller projects, responses were triangulated with reviews of project (or company) websites.

Table 10 reports the demographics of our governance metrics. Whereas our governance framework covers all the spectrum of software projects, including fully closed ones, we could not include in our final sample applications with score 1 for the code dimensions as not enough information was available for these projects.

5.2 Statistical Approach and Methodology

All statistical analyses have been performed with SPSS 15 on the sample of 75 projects. Note that SPSS handles missing data and, therefore, we were able to include all 75 projects, even those with null values for either quality or governance metrics (see online Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2008.68). Tables 11 and 12 report the results of the principal component analysis (PCA) on our quality metrics. Results show that there is one component underlying all five metrics of quality, with a composite factor reliability (CFR) always greater than 0.88. Tables 13 and 14 report the results of PCA on our governance metrics. Only one factor has been extracted, with a CFR always greater than 0.77. As all CFR values are above 0.70, results support the assumption that all quality metrics and all governance metrics provide measures of only one aggregate concept [4], [42].

TABLE 11
Correlation Matrix of Design Quality Metrics (Pearson Index)

	CBO	COF	NOC	WMC	DIT
CBO	1.000	0.917	0.910	0.661	0.910
COF	0.917	1.000	0.990	0.858	0.987
NOC	0.910	0.990	1.000	0.857	0.999
WMC	0.661	0.858	0.857	1.000	0.851
DIT	0.910	0.987	0.999	0.851	1.000

Determinant = 6.36E-007 - The significance of all coefficients is above 99.9%

TABLE 12
Principal Component Analysis of Design Quality Metrics

Variable	Composite factor reliability (CFR)
NOC	0.995
COF	0.994
DIT	0.993
CBO	0.921
WMC	0.881

TABLE 13
Governance Metrics Correlation Matrix (Pearson Index)

	Code	Contribution	Project leadership	Working practices
Code	1.000	0.502**	0.600**	0.561**
Contribution	0.502**	1.000	0.693***	0.786***
Project leadership	0.600**	0.693***	1.000	0.636***
Working practices	0.561**	0.786**	0.636***	1.000

*** Coefficients significant at 99% - ***Coefficients significant at 99.9%*

TABLE 14
Governance Metrics Principal Component Analysis

Variable	Composite factor reliability (CFR)
Contribution	0.885
WP	0.883
Leadership	0.862
Code	0.768

AMOS 7 [3] was used to analyze the research model by means of structural equation modeling (SEM).[6] SEM is a common statistical technique for testing and estimating causal relationships [65], [70]. An important strength of SEM is its ability to model constructs as *latent* variables, i.e., variables that are not measured directly but are estimated in the model from a set of proxy variables. In this research, design quality and governance represent the latent variables as they are measured with multiple proxies. SEM allows latent variables within a model and can verify research hypotheses that involve latent variables. In contrast, simple multivariate regression requires computing actual values for all variables.

Fig. 1 shows the research model. According to SEM's graphic format, rectangular boxes represent observed variables, oval boxes represent latent variables (or principal components), arrows represent relations between variables, and circles represent the Gaussian errors associated with each variable. The value above each arrow reports the standardized regression coefficient. The value above the

6. Note that AMOS 7 deals with incomplete data and, consequently, allowed us to use the whole sample of 75 projects.

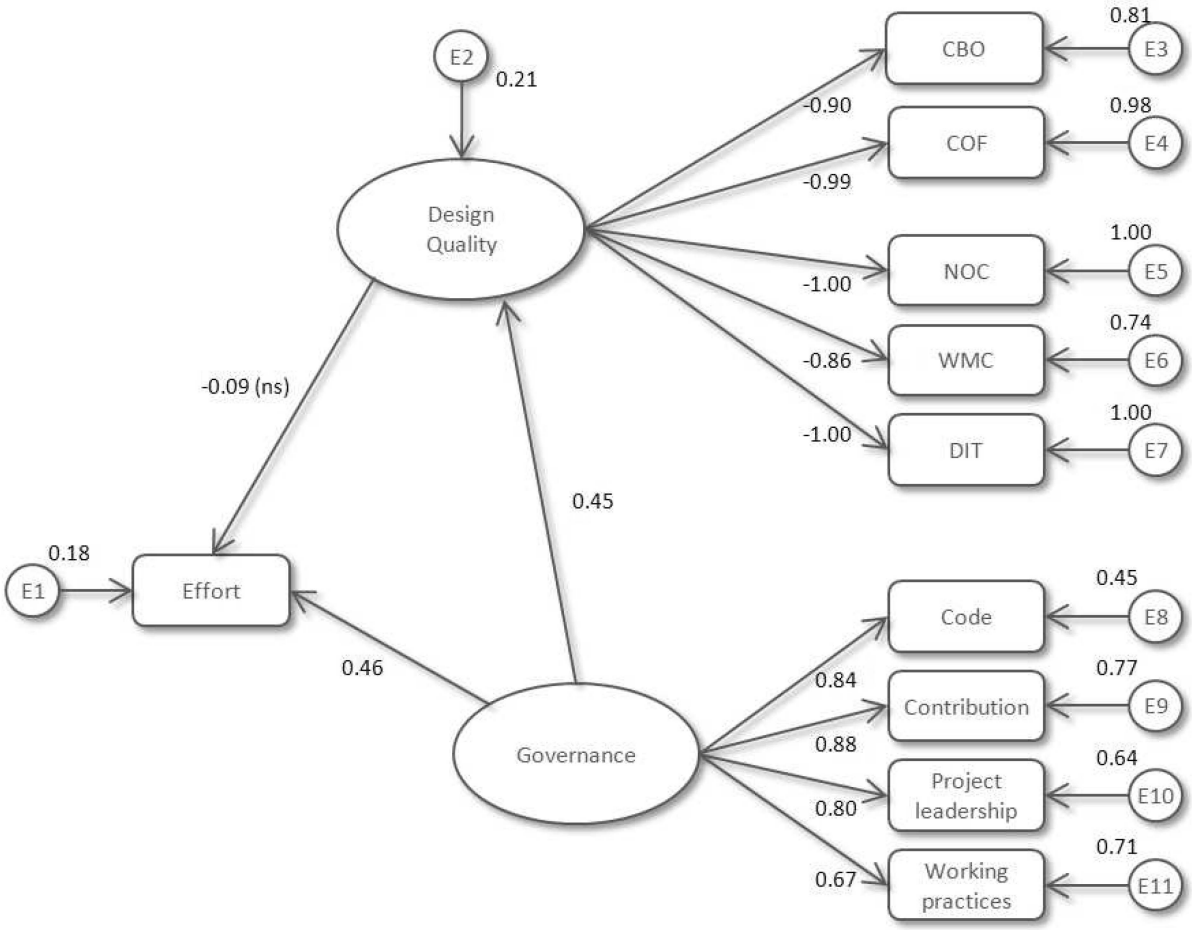

Fig. 1. Research model (ns = not significant).

TABLE 15
Goodness-of-Fit Indices for the Research Model

	Research model	Desired levels
χ^2	54.13	
d.f.	33	
χ^2 / d.f.	1.64	< 3.0
P	0.01	<0.01
NFI	0.87	>0.90
TLI	0.90	>0.90
CFI	0.94	>0.90

Gaussian error associated with each dependent variable represents the squared multiple correlation, which expresses the extent to which the variance of the dependent variable is explained by the independent variables.

The effort for each application k is $e_k(i)$, computed as shown by (2), where i indicates the most recent stable version. According to our hypotheses, all quality metrics refer to version $i-1$. Governance metrics refer to the corresponding project.

To assess the fit of the model, several indices were used, as suggested by Kline [70]. Results are reported in Table 15. The ratio of χ^2 to the degrees of freedom lower than three (1.64) indicates a good fit for the model [27]. The Normed Fit Index (NFI) compares the hypothesized model to the null model, with a value close to 1 indicating a perfect fit. The value of NFI is very close to the recommended value of 0.90 [50]. The Tucker-Lewis Index (TLI) represents the improvement in the proportion of total covariance explained by the hypothesized model over that explained by the null model [107]. The Comparative Fit Index (CFI) accounts for the reduction in the model misfit of the hypothesized model compared to the null model [12]. Values of TLI and CFI exceeding 0.90 are recommended [10], [60]. Table 15 shows that the overall model fit is

TABLE 16
Estimates of Regression Weights for the Research Model

Dependent variable	Independent variable	Estimates of standardized regression weights (b)	Estimates of non-standardized regression weights	S.E.	p
CBO	Design quality	-0.898	-1.000		<0.001
COF	Design quality	-0.989	-5.456	0.593	<0.001
NOC	Design quality	-1.001	-6.416	0.666	<0.001
WMC	Design quality	-0.860	-0.908	0.146	<0.001
DIT	Design quality	-0.998	-7.020	0.738	<0.001
Code	Governance	0.667	0.183	0.051	<0.001
Contribution	Governance	0.877	1.000		<0.001
Project leadership	Governance	0.800	0.704	0.153	<0.001
Working practices	Governance	0.842	0.625	0.126	<0.001
Effort	Design quality	-0.090	2.855	7.173	n.s.
Design quality	Governance	0.453	0.072	0.034	<0.05
Effort	Governance	0.462	2.324	1.239	<0.05

TABLE 17
Summary of Results

Hypothesis	Metric	Main findings
H1	Effort	Quality investments cannot be credited reductions of unit effort.
	Design quality	Quality metrics cannot be taken as predictors of unit effort.
H2	Governance Design quality	More open governance is associated with higher design quality.
H3	Governance Effort	More open governance causes higher unit effort.

satisfactory. All goodness-of-fit indices of the SEM estimation are above the recommended threshold values except for NFI, which nevertheless is very close to the acceptance threshold. Overall, the model represented in Fig. 1 is supported by data. Alternative models that reversed causal relationships were nonsignificant or less significant than the model in Fig. 1.

5.3 Results of Structural Model Testing

The estimation results of the research model represented in Fig. 1 are shown in Table 16. All of the relationships between quality metrics (CBO, COF, NOC, WMC, and DIT) and the corresponding latent variable (design quality) are significant, with $p < 0.001$. The same applies to governance. This confirms that factorization was performed correctly.

Our hypotheses are represented by specific regression relationships within the structural equation model. The main difference between regression and correlation is that the former assumes an interpretation framework to define meaningful causal relationships, which we have proposed in Section 3. Regression hypotheses are supported by findings if 1) the sign of regression weights is consistent with the direction of the hypothesized causal relationships and 2) regression weights are statistically significant, with p-values lower than 5 percent.

Since the estimation of the regression weights of the relationship between effort and design quality is not statistically significant $(p = 0.69)$, H1 (*a development initiative that increases software design quality has a greater unit development effort than a development initiative that does not increase software design quality*) is not verified. Note that the same results have been obtained by testing the regression between individual design quality metrics and effort. Hypothesis H2 (*a higher degree of openness in the governance practices leads to greater software design quality*) is verified since the regression weights between governance and design quality are positive and statistically significant $(b = 0.453, p < 0.05)$. Finally, hypothesis H3 (*a higher degree of openness in the governance practices leads to a higher unit development effort*) is verified as regression weights are positive and statistically significant $(b = 0.462, p < 0.05)$. Table 17 summarizes the main results of our testing by specifying the metrics involved in verifying different hypotheses.

6 Discussion

Empirical results do not support hypothesis H1 (*a development initiative that increases software design quality has a greater unit development effort than a development initiative that does not increase software design quality*) and, therefore, we cannot state that a development initiative that increases software design quality, mainly measured as coupling and inheritance, has a greater unit development effort in our sample. This indicates that quality does not necessarily increase unit development costs. Previous literature has found similar results by analyzing the relationship between quality and cost in the long term [98]. In contrast, quality has been found to increase costs in the short term [98], [103]. This paper takes a short-term perspective and, consistent with previous literature, hypothesizes a direct and positive relationship between quality and costs. Findings do not support H1, unlike previous literature.

In Section 3, we have noted that the assumptions underlying previous literature on the relationship between quality and costs may not be verified in OS projects. Refactoring initiatives aimed at increasing software design quality are periodic in traditional closed-source projects; OS projects are more likely to follow the practice of continuous refactoring to keep quality consistently high [72]. A likely explanation for the nonsignificant direct relationship between quality and cost is that all development initiatives are in charge of some refactoring, but, at the same time, benefit from the reductions of maintenance costs enabled by previous refactoring.

It is interesting to note that previous literature does not provide conclusive results demonstrating that investments in design quality have an overall positive balance (cf. Slaughter [98]). Investments in design quality are assumed to involve a cost in the short term and to allow cost reductions in the long term. However, assessing their overall balance is still an open research issue. Our findings seem to indicate that, from a pure cost perspective, quality investments represent a zero-sum game. Implementing higher quality software is more costly, but a higher quality reduces maintenance costs, with a net balance close to zero.

From a scientific standpoint, these results hint that design quality metrics should not be taken as measures or predictors of effort. The work of MacCormack et al. [80] proposes the use of a metric of coupling, called propagation cost, as a predictor of effort. This approach may not apply to OS projects, where quality and effort do not seem to show a direct relationship. Furthermore, previous research results identifying a break-even time when either refactoring or replacement becomes economically convenient [103] cannot be used in OS projects, where quality investments seem to be continuous.

Hypothesis H2 posits that a more open governance leads to a higher software design quality. As discussed in Section 3, the literature provides two different perspectives on the association between the degree of openness and design quality. On one hand, design quality can be seen as an enabler of a more open approach to governance as it represents a means to facilitate coordination in more distributed, informal, and virtual groups of developers. On the other hand, a higher quality can be seen as a consequence of a more open governance since developers of OS code tend to be more motivated, have a higher sense of ownership over their software artifact, and be less concerned with project deadlines. These different perspectives on the relationship between open governance mechanisms and software design quality seem to be complementary rather than conflicting. Our findings confirm the complementarity of these perspectives. We noted in Section 3 that, from a logical point of view, stating that design quality is an enabler of open governance is equivalent to stating that open governance implies design quality.

Our preliminary interviews with OS project managers confirm that quality is indeed a consequence of the governance approach [23]. When code is open, all team members take personal pride in writing clean and understandable pieces of code, in polishing the design of their artifacts, and in commenting their work since they feel exposed to the judgment of the whole community of developers and, consequently, pay particular attention to quality. When development is voluntary, with no pressure from managers or customers, time can be more easily allocated to improving the design of code. At the same time, design quality is actually necessary when several developers have to work on the same application by meeting and communicating only virtually. Different groups of developers typically work on different modules of the same application. A low degree of coupling among modules allows separate groups to be as independent as possible of each other and facilitates coordination.

The fact that different quality metrics factorize means that, in our project sample, they represent different observable variables of a single underlying phenomenon. Conceptually, they remain distinct variables which focus on specific design characteristics. Their factorization indicates that a design quality improvement involves a concurrent change in multiple variables. This reinforces the observation made by Lanza and Marinescu [76] stating that no single metric alone can represent quality (see also Nagappan et al. [87]). Our findings indicate that our set of metrics coherently measure the same phenomenon. Further research is required to integrate our set of metrics with other design quality metrics. In particular, understanding whether a piece of code has sufficient quality to be effectively opened to a broad community is still an open research issue.

Hypothesis H3 suggests that development initiatives in projects that are managed with a higher degree of openness have a higher unit development effort. This seems to confirm that the communication and coordination overhead that is required to enable open governance approaches involves a tangible additional effort. It also implies that the cost advantages of OS may not come from cost reductions, especially for OS projects that are led by a single company that undertakes the majority of costs. Community OS projects may benefit from cost reductions related to the sharing of the overall development effort within a broader community. However, this type of cost sharing is not the objective of commercial OS projects, where more than 90 percent of the code is developed by the employees of a single company. Even within community OS projects, a significant part of the code is often written by developers hired by companies that pursue specific revenue objectives, although they do not explicitly associate their brand with the software product [25].

In all of the OS projects where contribution is not totally voluntary, companies and stakeholders should pursue

benefits other than cost reductions. OS communities, fostered by open paradigms, are an important source of early feedback on functionalities, bug fixing, and noncode development support (e.g., editing of documentation, localization activities, production of promotional material). They are also an invaluable means to obtain an indirect but highly effective marketing through word of mouth [74]. Various types of commercial agreements with partners specializing in professional services, personalization, training, and post-sale assistance can also stem from a higher degree of openness [46], [47]. Image is clearly enhanced by OS and companies are aware that the visibility of their higher quality code can augment their brand equity [47].

A last contribution of this study is our empirical metric of effort. This paper's effort metric provides a measure of person-days given the current state of the art of project management tools in OS projects. Several tools keeping track of the changes made to an application and the time actually spent by developers to make those changes have been proposed and recently integrated in OS development environments, such as Robbes' tool [95] and the Mylyn tool[7] of Eclipse. The challenge is making developers use these tools, as suggested by Kersten and Murphy [67], [68]. Mens and Demeyer [82] indicate that objective assessments of effort could be particularly beneficial in do-ocracy contexts. They observe that these tools should not be seen as control instruments but as a way to objectively measure individual contribution and facilitate cooperation within a community.

7 Threats to Validity and Future Work

This work presents some threats to external and internal validities [109]. The criteria we have adopted to select our sample of OS projects may constitute a threat to external validity. First, the projects have been selected according to their maturity. Maturity has been measured based on the quantitative project maturity assessment provided by online software repositories. This assessment is not peer reviewed, but it represents the only available quantitative criterion. Our interviews confirmed that it is very rare that a project administrator declares that his/her project is mature if it is still in beta status, whereas the contrary is more frequent. Consequently, our maturity criterion for selecting the sample can be considered conservative.

Second, our sample is limited to 75 applications and may not be representative of the whole OS world. As explained in Section 5.1, our sample includes a broad range of projects and related governance practices. However, as our sample primarily includes projects with a very large user base, results may not be verified for one-person projects or smaller communities.

Third, we have focused on the Java language to obtain comparable quality metrics. As a consequence, our data set has missing values and the hypotheses involving the quality variable have been tested on a subset of 48 applications.

Finally, it should be noted that we have not analyzed fully closed projects. This is due to greater difficulty in accessing the code of fully closed applications. Open governance practices can be also applied in closed-source projects by leveraging dimensions of governance other than those related to openness. While our results indicate that a less formal approach to governance has tangible quality advantages, further research is required to understand how to position a project to benefit from the most profitable cost to quality trade-offs.

With regard to internal validity, the operationalization of our metrics raises several threats. This paper suggests that software design quality represents an important managerial variable to implement an open governance approach, consistent with Raymond's OS manifesto [92]. The quality dimensions addressed by this study are related to software design and, therefore, are not directly perceived by users. Consequently, we cannot state that a higher quality translates into effectiveness measures perceived by users. Future work should verify whether a higher design quality translates into perceivable user benefits. Previous software engineering literature has found a positive correlation between design quality and a decreasing number of bugs [9], [21], [32], [49], [110] and between design quality and time to market [1] for proprietary software. These correlations have recently been verified for OS projects as well [35], [71], [111]. Previous literature has used these correlations between design quality and effectiveness to reinforce the importance of research in the field of software quality metrics. This paper has identified a coherent set of relevant design quality metrics, mainly related to coupling and inheritance, which are correlated with the degree of project openness. However, further work is required to extend our set of metrics to other aspects of quality and analyze their correlation with metrics of effectiveness.

Our aggregate metric of governance is based on the assessment of four dimensions, which have been identified according to face-to-face interviews with 25 project managers of major software projects. We have specifically asked project managers to cite the most important dimensions for evaluating the "openness" of a software project according to their experience and selected the highest ranked dimensions (as explained in Section 4.3). Given that it represents a result of interviews, this metric may be flawed by subjectivity. However, this threat has been mitigated by applying a two-phase approach to the selection and ranking of dimensions [75] based on: 1) interviewing all key roles for each project, 2) triangulating responses, and 3) making recalls to find an agreement on inconsistencies.

Our metric of effort is subject to a measurement error, as discussed in Section 4.2. We have conducted a survey with 489 active OS developers to refine and consolidate our metric, but the resulting effort metric still involves estimates. In particular, 1) this metric could be integrated with data from CVS or other configuration versioning systems, 2) we have estimated the percentage of active developers based on the online community, rather than actually measuring the percentage per project, and 3) we have assumed that the effort spent for version i before version $i-1$ was released is negligible. Overall, while this is partly inherent to our goal of providing a direct measure of effort, future work could further validate our metric by analyzing its relationship with less direct but less subjective measures of effort.

Acknowledgments

The authors would like to thank all of the OS project leaders who have participated in the survey and face-to-face interviews. Particular thanks are expressed to Carlo Ghezzi and Anthony I. Wasserman for their mentorship and feedback on the early versions of this work.

7. http://www.eclipse.org/mylyn.

REFERENCES

[1] M. Agrawal and K. Chari, "Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects," *IEEE Trans. Software Eng.,* vol. 33, no. 3, pp. 145-156, Mar. 2007.

[2] Y. Ahn, J. Suh, S. Kim, and H. Kim, "A Software Maintenance Project Effort Estimation Model Based on Function Points," *J. Software Maintenance Evolution: Research and Practice,* vol. 15, no. 2, pp. 71-85, 2003.

[3] J.L. Arbuckle, *Amos 5.0 Update to the Amos User's Guide.* Small Waters, 2003.

[4] R.P. Bagozzi and Y. Yi, "On the Evaluation of Structural Equation Models," *J. Academy of Marketing Science,* vol. 16, no. 1, pp. 74-94, 1988.

[5] R.D. Banker and S.A. Slaughter, "A Study on the Effects of Software Development Practices on Software Maintenance Effort," *Proc. Int'l Conf. Software Maintenance,* pp. 197-205, 1996.

[6] R.D. Banker, G.B. Davis, and S.A. Slaughter, "Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study," *Management Science,* vol. 44, no. 4, pp. 433-450, 1998.

[7] V.R. Basili and R.W. Weiter Jr., "A Controlled Experiment Quantitatively Comparing Software Development Approaches," *IEEE Trans. Software Eng.,* vol. 7, no. 3, pp. 279-401, 1981.

[8] V.R. Basili, L. Briand, S. Condon, Y.-M. Kim, W.L. Melo, and J.D. Valett, "Understanding and Predicting the Process of Software Maintenance Releases," *Proc. 18th Int'l Conf. Software Eng.,* pp. 464-474, 1996.

[9] V.R. Basili, L.C. Briand, and W.L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Trans. Software Eng.,* vol. 22, no. 10, pp. 751-761, Oct. 1996.

[10] A. Battacherjee, "Understanding Information Systems Continuance: An Expectation-Confirmation Model," *MIS Quarterly,* vol. 25, no. 3, pp. 351-370, 2001.

[11] K. Beck, *Extreme Programming Explained.* Addison-Wesley, 1999.

[12] P.M. Bentler, "Comparative Fit Indexes in Structural Models," *Psychological Bull.,* vol. 107, no. 2, pp. 238-246, 1990.

[13] J. Bergus, *The5 Types of Open Source Projects,* http://www.powerpostgresql.com/5_types, 2005.

[14] A. Bianchi, D. Caivano, F. Lanubile, and G. Visaggio, "Evaluating Software Degradation through Quality," *Proc. Seventh IEEE Int'l Software Metrics Symp.,* pp. 210-219, 2001.

[15] A.B. Binkley and S.R. Scatch, "Validation of the Coupling Dependency Metric as a Predictor of Run Time Failures and Maintenance Measures," *Proc. 20th Int'l Conf. Software Eng.,* pp. 452-455, 1998.

[16] B. Boehm, *Software Engineering Economics.* Prentice Hall, 1981.

[17] B. Boehm, *Software Cost Estimation with COCOMO II.* Prentice Hall, 2000.

[18] B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison-Wesley, 2002.

[19] A. Boulanger, "Open Source versus Proprietary Software: Is One More Reliable and Secure than the Other," *IBM Systems J.,* vol. 44, no. 2, p. 239, 2005.

[20] L.C. Briand, K. El Emam, and S. Morasca, "On the Application of Measurement Theory in Software Engineering," *J. Empirical Software Eng.,* vol. 1, no. 1, pp. 61-88, 1996.

[21] L.C. Briand, J.W. Daly, and J.K. Wust, "A Unified Framework for Coupling Measurement in Object Oriented Systems," *IEEE Trans. Software Eng.,* vol. 25, no. 1, pp. 91-121, Jan./Feb. 1999.

[22] F. Brito e Abreu, "The MOOD Metrics Set," *Proc. ECOOP Workshop Metrics,* 1995.

[23] E. Capra, "Software Design Quality and Development Effort: An Empirical Study on the Role of Governance in Open Source Projects," PhD dissertation, Dept. of Electronics and Information, Politecnico di Milano, 2008.

[24] E. Capra and A.I. Wasserman, "A Framework for Evaluating Managerial Styles in Open Source Projects," *Proc. Int'l Conf. Open Source Systems,* 2008.

[25] E. Capra and A.I. Wasserman, "Evaluating Software Engineering Processes in Commercial and Community Open Source Projects," *Proc. First Int'l Workshop Emerging Trends in FLOSS Research and Development,* 2007.

[26] E. Capra, C. Francalanci, and F. Merlo, "The Economics of Open Source Software: An Empirical Analysis of Maintenance Costs," *Proc. Int'l Conf. Software Maintenance,* pp. 395-404, 2007.

[27] E.G. Carmines and J.P. McIver, "Analyzing Models with Unobserved Variables: Analysis of Covariance Structures," *Social Measurement: Current Issues,* G.W. Bohrnstedt and E.F. Borgatta, eds., pp. 65-115, Sage Publications, 1981.

[28] T. Chan, S. Chung, and T. Ho, "An Econometric Model to Estimate Software Rewriting and Replacement Times," *IEEE Trans. Software Eng.,* vol. 22, no. 8, pp. 580-598, Aug. 1996.

[29] N. Chapin, J.E. Hale, K.M. Khan, J.F. Ramil, and W.-G. Tan, "Types of Software Evolution and Software Maintenance," *J. Software Maintenance and Evolution: Research and Practice,* no. 13, pp. 3-30, 2001.

[30] J.Y. Chen and J.F. Lu, "A New Metric for Object-Oriented Design," *J. Information System and Software Technology,* vol. 35, no. 4, pp. 232-240, 1993.

[31] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Software Eng.,* vol. 20, no. 6, pp. 476-493, June 1994.

[32] S.R. Chidamber, D.P. Darcy, and C.F. Kemerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis," *IEEE Trans. Software Eng.,* vol. 24, no. 8, pp. 629-639, Aug. 1998.

[33] M.B. Chrissis, M. Konrad, and S. Shrum, *CMMI Guidelines for Process Integration and Product Improvement.* Addison-Wesley, 2003.

[34] D.P. Darcy, C.F. Kemerer, S.A. Slaughter, and J.E. Tomayko, "The Structural Complexity of Software: An Experimental Test," *IEEE Trans. Software Eng.,* vol. 31, no. 11, pp. 982-995, Nov. 2005.

[35] J. Ekanayake, A. Bernstein, and M. Pinzger, "Improving Defect Prediction Using Temporal Features and Nonlinear Models," *Proc. Int'l Workshop Principles of Software Evolution),* pp. 11-18, 2007.

[36] T.J. Emerson, "A Discriminant Metric for Module Comprehension," *Proc. Seventh Int'l Conf. Software Eng.,* pp. 294-431, 1984.

[37] N. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach.* Int'l Thomson Computer Press, 1997.

[38] R.T. Fielding, "Shared Leadership in the Apache Project," *Comm. ACM,* vol. 42, no. 4, pp. 42-43, 1999.

[39] B. Fitzgerald, "A Critical Look at Open Source," *Computer,* vol. 37, no. 7, pp. 92-94, July 2004.

[40] B. Fitzgerald, "The Transformation of Open Source Software," *MIS Quarterly,* vol. 30, no. 2, pp. 587-598, 2006.

[41] K. Fogel, *Producing Open Source Software.* O'Reilly, 2006.

[42] C. Fornell and D.F. Larcker, "Evaluating Structural Equation Models with Unobservable Variables and Measurement Errors: Algebra and Statistics," *J. Marketing Research,* vol. 18, no. 3, pp. 383-388, 1981.

[43] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 2001.

[44] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering,* second ed. Prentice Hall PTR, Apr. 2002.

[45] P.J. Gomes and N.R. Joglekar, "The Costs of Coordinating Distributed Software Development Tasks," Boston Univ. School of Management Working Paper, Oct. 2004.

[46] R. Goldman and R.P. Gabriel, *Innovation Happens Elsewhere: Open Source as Business Strategy.* Morgan Kaufmann, Apr. 2005.

[47] G. Goth and B. Massey, "Open Source Business Models: Ready for Prime Time," *IEEE Software,* vol. 22, no. 6, pp. 98-102, Nov./Dec. 2005.

[48] M. Griffiths, "Most Software Development Metrics Are Misleading and Counterproductive," *Agile J.,* http://www.agilejournal.com/content/view/107, Oct. 2006.

[49] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical Validation of Object Oriented Metrics on Open Source Software for Fault Prediction," *IEEE Trans. Software Eng.,* vol. 31, no. 10, pp. 897-910, Oct. 2005.

[50] J.F. Hair, R.E. Anderson, R.L. Tatham, and W.C. Black, *Multivariate Data Analysis with Readings.* Prentice Hall, 1995.

[51] M.H. Halstead, "Elements of Software Science," *Elsevier Computer Science Library,* 1977.

[52] R. Harrison, S. Counsell, and R. Nithi, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," *IEEE Trans. Software Eng.,* vol. 24, no. 6, pp. 491-496, June 1998.

[53] D.E. Harter, M.S. Krishnan, and S.A. Slaughter, "Effects of Process Maturity on Quality, Cycle Time and Effort in Software Product Development," *Management Science,* vol. 46, no. 4, pp. 451-466, 2000.

[54] S. Henry and D. Kafura, "Software Structure Metrics Based on Information Flow," *IEEE Trans. Software Eng.*, vol. 7, no. 5, pp. 510-518, 1981.
[55] J.D. Herbsleb and D.R. Goldenson, "A Systematic Survey of CMM Experience and Results," *Proc. 18th Int'l Conf. Software Eng.*, pp. 323-330, 1996.
[56] M. Hitz and B. Montazeri, "Measure Coupling and Cohesion in Object-Oriented Systems," *Proc. Int'l Symp. Applied Corporate Computing*, Oct. 1995.
[57] M. Hitz and B. Montazeri, "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective," *IEEE Trans. Software Eng.*, vol. 22, no. 4, pp. 267-271, Apr. 1996.
[58] J. Howison and K. Crowston, "The Social Structure of Free and Open Source Software Development," *First Monday [Online]*, vol. 10, no. 2, Feb. 2005.
[59] J. Howison and K. Crowston, "The Perils and Pitfalls of Mining SourceForge," *Proc. Int'l Workshop Mining Software Repositories*, pp. 7-12, 2004.
[60] L. Hu and P.M. Bentler, "Cutoff Criteria for Fit Indexes in Covariance Structure Analysis: Conventional Criteria versus New Alternatives," *Structural Equation Modeling*, vol. 6, no. 1, pp. 1-55, 1999.
[61] K. Ishikawa, *Introduction to Quality Control.* Union of Japanese Scientists and Engineers Press, 1989.
[62] *Software Engineering—Product Quality*, ISO/IEC, TR 9126:2003, Int'l Organization for Standardization, Geneva, Switzerland, 2003.
[63] *Software Engineering—Software Product Quality Requirements and Evaluation (SQuaRE)*, ISO/IEC, TR 25000:2005, Int'l Organization for Standardization, Geneva, Switzerland, 2005.
[64] J. Jurison, "Software Project Management: The Manager's View," *Comm. Assoc. Information Systems*, vol. 2, no. 3, article no. 2, 1999.
[65] D. Kaplan, "Structural Equation Modeling: Foundations and Extensions," *Advanced Quantitative Techniques in the Social Sciences Series*, vol. 10, 2000.
[66] C.F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Comm. ACM*, vol. 30, no. 5, pp. 416-430, 1987.
[67] M. Kersten and G.C. Murphy, "Using Task Context to Improve Programmer Productivity," *Proc. 14th SIGSOFT Symp. Foundations of Software Eng.*, pp. 1-11, 2006.
[68] M. Kersten and G.C. Murphy, "Mylar: A Degree-of-Interest Model for IDEs," *Proc. Fourth Int'l Conf. Aspect-Oriented Software Development*, pp. 159-168, 2005.
[69] L.J. Kirsch, "The Management of Complex Tasks in Organizations: Controlling the Systems Development Process," *Organization Science*, vol. 7, no. 1, pp. 1-21, 1996.
[70] R.B. Kline, *Principles and Practice of Structural Equation Modeling.* Guilford Press, 1988.
[71] P. Knab, M. Pinzger, and A. Bernstein, "Predicting Defect Densities in Source Code Files with Decision Tree Learners," *Proc. Int'l Workshop Mining Software Repositories*, pp. 119-125, 2006.
[72] S. Koch, "Agile Principles and Open Source Software Development: A Theoretical and Empirical Discussion," *Extreme Programming and Agile Processes in Software Engineering*, pp. 85-93, Springer, 2004.
[73] M. Komuro, "Experience of Applying SPC Techniques to Software Development Processes," *Proc. 28th Int'l Conf. Software Eng.*, pp. 577-585, 2006.
[74] S. Krishnamurthy, "A Managerial Overview of Open Source Software," *Business Horizons*, pp. 47-56, Sept./Oct. 2003.
[75] S. Kvale, *Interviews: An Introduction to Qualitative Research Interviewing.* Sage Publications, 1996.
[76] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice—Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems.* Springer, 2006.
[77] W. Li and S. Henry, "Maintenance Metrics for the Object Oriented Paradigm," *Proc. IEEE Int'l Software Metrics Symp.*, pp. 52-60, 1993.
[78] B. Lientz and B. Swanson, *Software Maintenance Management.* Addison-Wesley, 1981.
[79] R. Lincke and W. Lowe, "Compendium of Software Quality Standards and Metrics—Version 1.0," technical report, Vaxjo Universitet, 2007.
[80] A. MacCormack, J. Rusnak, and C.Y. Baldwin, "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code," *Management Science*, no. 52, pp. 1015-1030, July 2006.
[81] T.J. McCabe, "A Complexity Measure," *Proc. Second Int'l Conf. Software Eng.*, p. 407, 1976.
[82] T. Mens and S. Demeyer, "Future Trends in Software Evolution Metrics," *Proc. Fourth Int'l Workshop Principles of Software Evolution*, pp. 83-86, 2001.
[83] T. Mens and T. Tourwé, "A Survey of Software Refactoring," *IEEE Trans. Software Eng.*, vol. 30, no. 2, pp. 126-139, Feb. 2004.
[84] M. Michlmayr, "Managing Volunteer Activity in Free Software Projects," *Proc. USENIX Ann. Technical Conf.*, p. 39, 2004.
[85] A. Mockus and T.L. Graves, "Identifying Productivity Drivers by Modeling Work Units Using Partial Data," *Technometrics*, vol. 42, no. 2, pp. 168-179, 2001.
[86] M. Monga, "From Bazaar to Kibbutz: How Freedom Deals with Coherence in the Debian Project," *Proc. Workshop Open Source Software Eng.*, pp. 71-75, 2004.
[87] N. Nagappan, T. Ball, and A. Zeller, "Mining Metrics to Predict Component Failures," *Proc. 28th Int'l Conf. Software Eng.*, pp. 452-461, 2006.
[88] *Open Source Initiative (OSI)*, http://opensource.org, 2008.
[89] E.I. Oviedo, "Control Flow, Data Flow and Programmers Complexity," *Proc. Int'l Computer Software and Applications Conf.*, pp. 146-152, 1980.
[90] R.E. Park, "Software Size Measurement: A Frame Work for Counting Source Code Statements," Technical Report CMU/SEI-92-TR-020, Carnegie Mellon Univ., 1992.
[91] J.F. Ramil, "Continual Resource Estimation for Evolving Software," *Proc. Int'l Conf. Software Maintenance*, pp. 289-293, 2003.
[92] E.S. Raymond, *The Art of Unix Programming.* Addison-Wesley Professional, 2003.
[93] P.S. Renz, *Project Governance: Implementing Corporate Governance and Business Ethics in Nonprofit Organizations.* Physica-Verl, 2007.
[94] D. Riehle, "The Economic Motivation of Open Source Software: Stakeholder Perspective," *Computer*, vol. 40, no. 4, pp. 25-32, Apr. 2007.
[95] R. Robbes, "Mining a Change-Based Software Repository," *Proc. Fourth Int'l Workshop Mining Software Repositories*, p. 15, 2007.
[96] L.H. Rosenberg, "Applying and Interpreting Object Oriented Metrics," technical report, Software Assurance Technology Center NASA SATC, Apr. 1998.
[97] R.C. Sharble and S.S. Cohen, "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods," *ACM SIGSOFT Software Eng. Notes*, vol. 18, no. 2, pp. 60-73, 1993.
[98] S.A. Slaughter, D.E. Harter, and M.S. Krishnan, "Evaluating the Cost of Software Quality," *Comm. ACM*, vol. 41, no. 8, pp. 67-73, Aug. 1998.
[99] S.A. Slaughter, J. Roberts, and I. Hann, "Communication Networks in an Open Source Software Project," *Proc. Int'l Conf. Open Source Systems*, pp. 297-306, 2006.
[100] J. Sonnenfeld, "Good Governance and the Misleading Myths of Bad Metrics," *Academy of Management Executives*, vol. 18, no. 1, pp. 108-113, 2004.
[101] R. Subramanyam and M.S. Krishnan, "Empirical Analysis of CK Metrics for Object Oriented Design Complexity: Implications for Software Defects," *IEEE Trans. Software Eng.*, vol. 29, no. 4, pp. 297-310, Apr. 2003.
[102] C.R. Symons, "Function Point Analysis: Difficulties and Improvements," *IEEE Trans. Software Eng.*, vol. 14, no. 1, pp. 2-11, Jan. 1988.
[103] Y. Tan and V. Mookerjee, "Comparing Uniform and Flexible Policies for Software Maintenance and Replacement," *IEEE Trans. Software Eng.*, vol. 31, no. 3, pp. 238-256, Mar. 2005.
[104] I. Tervonen and P. Kerola, "Towards Deeper Co-Understanding of Software Quality," *Information and Software Technology*, vol. 39, no. 14-15, pp. 995-1003, 1998.
[105] A. Tille, *Custom Debian Distributions*, http://people.debian.org/~tille/debian-med/talks/paper-cdd/debian-cdd.en.pdf, Jan. 2007.
[106] D. Troy and S. Zweben, "Measuring the Quality of Structured Design," *McGraw-Hill Int'l Series in Software Eng.*, pp. 214-226, 1993.
[107] L.R. Tucker and C. Lewis, "A Reliability Coefficient for Maximum Likelihood Factor Analysis," *Psychometrika*, vol. 38, no. 11-10, 1973.
[108] M. Van Genuchten, "Why Is Software Late? An Empirical Study of Reasons for Delay in Software Development," *IEEE Trans. Software Eng.*, vol. 17, no. 6, pp. 582-590, 1991.
[109] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction.* Springer, 2000.

[110] Y. Zhou and H. Leung, "Empirical Analysis of Object Oriented Design Metrics for Predicting High and Low Severity Faults," *IEEE Trans. Software Eng.*, vol. 32, no. 6, pp. 771-789, June 2006.
[111] T. Zimmermann, R. Premraj, and A. Zeller, "Prediction Defects for Eclipse Make to Fix This Bug," *Proc. Workshop Predictor Models in Software Eng. (PROMISE)*, 2007.

Eugenio Capra received the MS degree (magna cum laude) in electronic engineering and the PhD degree in information engineering from the Politecnico di Milano in 2003 and 2008, respectively. He is a postdoctoral researcher at the Politecnico di Milano. As part of his doctoral studies, he was a visiting scholar at Carnegie Mellon University West, Mountain View, California, from September 2006 to March 2007. He was previously a business analyst at McKinsey from 2004 to 2005. His main research interests are software managerial models and economics.

Chiara Francalanci received the MS degree in electronic engineering and the PhD degree in computer science from the Politecnico di Milano. She is an associate professor of information systems at the Politecnico di Milano. As part of her postdoctoral studies, she has worked for two years at the Harvard Business School as a visiting researcher. She has authored articles on the economics of information technology and on feasibility analyses of IT projects, consulted in the financial industry, both in Europe and US, and is a member of the editorial board of the *Journal of Information Technology*.

Francesco Merlo received the MS degree in computer science engineering from the Politecnico di Milano in 2006. He is currently with the Information Systems Group, Department of Electronics and Information, Politecnico di Milano, where he is a PhD student. His research interests are the empirical evaluation of software economics and quality.