Association for Information Systems
AIS Electronic Library (AISeL)

ICIS 2008 Proceedings

International Conference on Information Systems (ICIS)

2008

Does Complexity Matter? The Impact of Change in Structural Complexity on Software Maintenance and New Developers' Contributions in Open Source Software

Vishal Midha
University of Texas - Pan American, vmidha@utpa.edu

DOES COMPLEXITY MATTER? THE IMPACT OF CHANGE IN STRUCTURAL COMPLEXITY ON SOFTWARE MAINTENANCE AND NEW DEVELOPERS' CONTRIBUTIONS IN OPEN SOURCE SOFTWARE

La complexité importe-t-elle? L'impact d'un changement de complexité structurelle sur la maintenance des logiciels et sur les contributions nouvelles des développeurs de logiciels libres

Completed Research Paper

Vishal Midha
University of Texas – Pan American
1201 W. University Dr., Edinburg, TX 78539
vmidha@utpa.edu

Abstract

The research paper focuses on how change in structural complexity impacts Open Source Software maintenance. We analyzed software maintenance in terms of change in number of bugs, change in time taken to fix bugs, and change in the number of contributions from new developers. Data for model validation was collected from SourceForge. The important implications of the study are discussed.

Keywords: Open Source Software, Software Maintenance, Structural Complexity, Developers Contribution

Résumé

La recherche se concentre sur la manière dont le changement dans la complexité de la structure des logiciels Open Source peut avoir des conséquences sur la maintenance de ces logiciels. Nous analysons la maintenance des logiciels en termes de nombre de bugs, de temps passé à réparer les bugs, et les changements dans le nombre de contributions apportées par les nouveaux développeurs. Les données de validation du modèle ont été recueillies par Source Forge. Les implications importantes de cette étude sont alors discutées.

Introduction

A typical Open Source Software (OSS) project starts when an individual (or group) writes the first version of the software and, in order to share it the community with similar needs, the software is released under a license that allows the community to use the software and modify the source code to meet local needs or to improve it. Making software available on the Internet allows developers around the world to contribute code, add new features, report bugs, and submit bug fixes to the current release. The developers of the project then incorporate the features and

fixes into the main source code, and a new release of the software is made available to the community. This process of code contribution and bug fixing is continued in such circular manner. Due to its cyclic and open nature, OSS is often argued to have faster evolution. The rationale is that multiple contributors write, test, or debug the product in parallel, which accelerates software evolution. Raymond (2001) mentioned that more people looking at the code will result in more bugs found, which is likely to accelerate software improvements. The model also claims that this rapid evolution produces better software than the traditional closed model (Open source, 2002).

A ready interpretation of the OSS development process is that of a perpetual maintenance task. Developing an OSS implies continual maintenance for bugs reported by various users (Samoladas, 2004). As most of the OSS projects are a result of voluntary work (Stewart et al 2005, Feller & Fitzgerald 2001, Siedlok 2002), it is crucial to ensure that the volunteers are able to do so with minimal efforts.

Usually a developer maintaining the source code has not participated in the initial development of the original program (Kemerer, 1995). So, a large amount of effort goes into understanding and comprehending the existing source code (Smith et al., 2006). It was reported as early as in the late 1970's that more than fifty percent of all software maintenance effort was devoted to comprehension (Fjelstad and Hamlen 1983). Comprehension of existing source code involves identifying logic among various segments of the source code and understanding their relationships. As software is becoming increasingly complex, the task of comprehension of existing software source code is becoming increasingly tough (Rilling & Klemola, 2003). The comprehension of existing source code, thus, plays a prominent role in software maintenance.

While software maintainability, which is dependent on comprehension of source code, has been linked with source code complexity, prior empirical evidence studying the magnitude of the link is relatively weak (Thaik et al., 1992). This could be attributed to the reason that many of such attempts have been based on experiments involving small pieces of software code written by students (Banker et al, 1993). This observation clearly indicates a need to conduct a study on the impact of complexity of source code written by professionals.

An uncontrolled complexity growth is big concern for OSS development (Nichols & Twidale, 2003) and it may trigger the need for either substantial software re-engineering or the entire system replacement. Hence, it is vital to understand the impact of complexity of source code on software maintenance, and, even more importantly, on the OSS maintenance. A number of studies have examined the impact of complexity on maintainability and have put forth the recommendations to reduce it. But, no study, to the best of our knowledge, has tested if the reduced complexity was beneficial to developers performing software maintenance. In this paper, following Banker et al.'s (1993) recommendation, we analyze the Open Source Software (OSS), written by real world OSS developer community, and study the impact of *change* in complexity on software maintenance. We study the software maintenance in terms of the change in the number of bugs, the change in time taken to fix the bugs, and the change in the number of contributions from new developers in OSS development. This novelty of the research will help software development managers in attracting contributions from developers.

The remaining of the paper is organized in the order as follows: In the next section, we draw on the relevant literature in software development to build upon our theoretical model. Next, we describe the empirical methods used in our study, followed by the evaluation of our model. Lastly, we discuss our results and present our conclusion by identifying the contributions and limitations of our study and its implications for research and practice in OSS development.

Literature and Model Development

When the first version of source code is released to public via the internet, developers from various parts of the world contribute to that code. The reasons as to why developers contribute have received a great deal of attention from researchers (e.g., Lakhani and Wolf, 2005). On the other hand, only a very limited work has been done to study the factors that could lead the developers to not contribute to a project (e.g., von Krogh et al., 2003). According to them, the major concern among the developers was the complexity of the source code and the level of difficulty of the used algorithms in order to achieve the desired goals. Fitzgerald (2005) also pointed that increasing complexity posits a barrier in the OSS development. As software projects progress, they become more complex, leading to a steep rise in the efforts required for comprehending the source code (Fitzgerald, 2005; Rilling & Klemola, 2003).

Comprehension of a source code could be conceptualized as an information processing task in which input informational cues are interpreted and manipulated to create task outcomes (Ramanujan and Cooper, 1994). It is essentially a mental pattern-recognition by the software developer which involves filtering and recognizing

enormous amount of data (Rilling & Klemola, 2003). Wood et al. (1986) argued that performance of such tasks depends on effective processing of cues in the task environment and is influenced by complexity. This is especially relevant in OSS which has been criticized for lack of proper documentation. When a developer wants to contribute to an existing OSS project without the proper documentation, it becomes an extremely challenging work to understand and manipulate relationships between cues presented by the existing source code.

As mentioned earlier, more than fifty percent of all software maintenance effort is devoted to comprehension, suggests that software comprehension plays a major role in OSS development. It must be noted that comprehension is important for traditional software development also. But, it is even more important in OSS, where the software development is not properly documented and is, primarily, done by volunteers.

Complexity and Its Impact

Much of what we know about software comes from analyses of closed source development (e.g. Boehm, 1981). As noted by Stewart et al (2005), the results from those findings have also been applied to OSS (e.g., study of Debian 2.2 development (Gonzalez-Barahona et al, 2001) or the Linux growth (Godfrey & Tu, 2000, 2002)).

The software literature suggests that software complexity has multiple facets, including structural complexity (Adamov & Richter, 1990) and algorithmic complexity (Hartmanis, 1994). The structural complexity of a program comes from "the organization of program elements within a program" (Gorla, 1997) and algorithmic complexity is defined in terms of the time taken to execute a program (Darcy et al., 2005). Dealing with structural complexity primarily expends intellectual resources; whereas algorithmic complexity primarily consumes machine resources.

Kearney et al (1986) suggested that the difficulty of understanding depends, in part, on structural properties of the source code. And, as we are concerned with the impact of complexity on source code comprehension, we primarily focus on structural complexity in this paper. We use structural complexity as the basis of the discussion in the remaining part of this section, where we theorize the impact of complexity on various aspects on OSS development.

Number of Bugs

The main idea behind the relationship between complexity and number of bugs is that when comparing two different solutions to the same problem provided that all other things are equal, the most complex solution generates the most number of bugs. This relationship is one of the most analyzed by software metrics' researchers and previous studies and experiments have found this relationship to be statistically significant (Curtis et al. 1979, Henry et al. 1981).

In order for a programmer to understand the existing source code, the programmer has to understand the flow of logic. And, when a programmer has to deal with a source code with high structural complexity, the programmer has to frequently search among dispersed pieces of code to determine the flow of logic (Ramanujan & Cooper, 1994). Understand and recollecting all such dispersed pieces increases the cognitive load on the programmer making such software with complex source code more liable to human errors. Failure to properly comprehend the source code often leads to bugs in the source code (Klemola, 2000). Complex software, hence, need more maintenance efforts. Gill and Kemerer (1991) reported that the number of bugs in a program is positively associated with maintenance effort and recommended further empirical testing with a larger data set. We, thus, hypothesize that the OSS projects which experience decrease in complexity over its previous release experience an decrease in the number of bugs (over its previous stage). Hence,

H1: *An increase in the source code's structural complexity is positively associated with an increase in the number of bugs in the OSS source code*.

Contributions from New Developers

Because of the important role of volunteer developers in the OSS development, attracting new developers to contribute and keeping them motivated is crucial to OSS development. This is especially important during the early development stage so that the number of developers could reach a critical mass.

Once a new developer is motivated to voluntarily contribute, he/she needs to first spend a large amount of time and resources to understand the existing source code. When the source code is easy to comprehend, it is easier to modify

it. However, when the source code is complex, a developer is required to input additional efforts and resources to understand it. Devoting such efforts and resources may pose as a barrier to the developer's motivation to contribute. Such barriers may also lead the potential developers to not to contribute to the project, or, in worst case, to leave the project. This leads us to our next hypothesis that the OSS projects which experience decrease in complexity over its previous release attract increased contributions from the new developers (over its previous stage). Hence,

H2: *An increase in the source code's structural complexity is negatively associated with an increase in the number of contributions to the OSS source code from new developers.*

Time to Fix Bugs

More complex source code adds to a programmer's cognitive load (Darcy et al., 2005). High cognitive load requires more time-consuming and resource-demanding efforts to familiarize oneself with the code. It is even possible that a source code is too complex to be able to comprehend at all. Thus, it requires resources that could be used for other activities, thereby lowering the productivity of the project.

In other words, a source code with lesser structural complexity does not need as much resources, thus reducing the turn around time to fix repairs. This leads to our next hypothesis that OSS projects which experience increase in the structural complexity over its previous release require longer average time to fix the bugs. Hence,

H3: *An increase in the source code's structural complexity is positively associated with an increase in the average time taken to fix the bugs in OSS source code.*

Methods

To test the hypotheses, we focused on the OSS projects hosted at Sourceforge. Sourceforge is the primary hosting place for OSS projects provided by the OSTG group and houses about 90 percent of all OSS projects. Though not all OSS projects are hosted at Sourceforge, researchers interested in investigating issues related to the OSS phenomenon have predominantly used Sourceforge data (Grewal et al. 2006; Madey et al. 2002; von Hippel and von Krogh 2003; Xu et al. 2005). The data collection took place from October 2006 to March 2007.

Studying all the projects hosted was unfeasible due to the limitation of our resources. To analyze the structural complexity of source code written by OSS users and developers, we limited our data selection to the projects that were targeted for either end users or developers only. In order to avoid ambiguity and to clearly differentiate between user-intended projects and developer-intended projects, we excluded those projects that are targeted at both end users and developers.

Within this category we further controlled for Programming Language and Operating System. When a project is first started, the developer usually specifies in what language or languages the software will be written. And the choice of programming language is rarely changed as the project progresses. Past literature in software engineering suggests that programming language has an explicit impact on complexity (Weyuker, 1988) and program size (Jones 1986). It is also difficult to compare lines of code between "high" and "low" level programming languages. This is because "lower level" programming languages have more lines of code and take longer to develop than higher level programming languages. Hence we controlled for the programming language of the project by allowing projects written in C/C++ only. We did not distinguish between projects written in C/C++ only and those written in multiple languages including C/C++. Secondly, operating system of the project impacts the complexity of the software and the development effort required. Like Programming Language, once Operating System is decided for a project, it rarely changes and is relatively stable. We ensured that all the projects in our data set were designed for either Windows or Linux/Unix operating system.

It is important to note that complexity of source code could only be measured after the software release has been released for use. Only after it has been used, discovered bugs are reported, and the source code is modified to fix the reported bugs. Once significant amount of modifications have been made to the source code, a new version is released to the public. Due to the modifications in the source code, the complexity of the source code changes. In

order to see if the complexity has changed from its previous release, we need to measure the complexity of both the previous and the current release. Similarly, to find the modifications and contributions made to the current release, we have to wait for the next release to be available. As a consequence, we have to study three consecutive releases, which we refer to as 1st, 2nd, and 3rd release.

As described, the change in complexity was measured using the values computed from 1st and 2nd release. Change in number of developers was calculated for 2nd and 3rd release, whereas other project characteristics were calculated from 3rd release. Therefore, our sample was further restricted to the projects that had at least 3 versions. Lastly, we chose projects for which the required data were publicly available (not all projects allow public access to the bug tracking system). Following the above mentioned criterion, we extracted several attributes for each software system, taking measurements over releases for size, developer information, bugs reported, sponsorships to a project, development status, number of downloads, and structural complexity. The first 450 usable projects were studied.

Focal Dependent and Independent Variables

Change in Number of Bugs Reported and Change in Time taken to Fix Bugs

We extracted various elements of data, including the bugs reported, the date on which the bugs were reported, the date on which the bugs were fixed, and the release number, from bug tracking system, and CVS reports. In doing so, we faced one problem that all the bugs in the current release were not closed at the time of our study. To overcome the problem, we decided to study the earlier releases that had more than 90% of the bugs closed at the time of study. From these extracted elements, we calculated the number of bugs reported and the time taken to fix them for different software releases considered in this study. From the number of bugs and time to fix those bugs for each release, we computed the change in number of bugs (*ChgBugsReported*) over previous release and change in average time to fix the bugs (*ChgFixTime*).

Contributions from New Developers

Software developers use Concurrent Versioning System (CVS) to manage software development process. CVS stores the current version(s) of the project and its history. A developer can check-out the complete copy of the code, work on this copy and then check in her changes. The modifications are peer reviewed ensuring the quality. The CVS updates the modified file automatically and registers it as a commit. CVS keeps track of what change was made, who made the change, and when the change was made. This information can be gathered from the log files of the CVS repository of a project. As CVS commits provide a measure of novel invention that is internally validated by peers (Grewal et al 2006, Crowston et al 2003), we use the number of CVS commits as a measure of contributions of developers. If a contribution is made by a new developer, one who did not contribute in the earlier release, it is considered a contribution from a new developer to the current release. From this, we computed the number of contributions made by new developers, which is represented as *ChgNewDevs*

Structural Complexity

There is a large variety of complexity metrics available in literature and being practiced by the software industry. The two most common metrics for structural complexity are Halstead's E and McCabe's cyclomatic. We looked at complexity at the degree of cognitive efforts involved.

McCabe's Cyclomatic Complexity (CC) tends to assess the difficulty faced by the maintainer in order to follow the flow control of the program (McCabe, 1976; Rana et al., 2006). It is considered an indicator of the effort needed to understand and test the source code (Stamelos et al., 2002). Kemerer and Slaughter (1997) used McCabe cyclomatic complexity metric to evaluate decision density, which represents the cognitive burden on a programmer in understanding the source code. To compute the cyclomatic complexity, each source code file was subjected to a commercial software code analysis tool.

Halstead Effort (HE) estimates complexity based on the number of unique operands and operators versus the total number of operands and operators (Halstead, 1977). It has been argued as a measure of lexical/ textual complexity leading to the mental effort required to develop or maintain a program.

To account for the effects of size, the metric was normalized by dividing it by the number of lines of code for each software project. In addition, this also reduces collinearity problems when size is included in the regression models (Gill and Kemerer, 1991). Change in complexity (*ChgMC* or *ChgHE*) was calculated by subtracting complexity

measure of former release from the complexity measure of latter release, i.e. $ChgMC = MC_{latter} - MC_{former}$ or $ChgHE = HE_{latter} - HE_{former}$.

Control Variables

Age

Brook's Law mentions that "Adding more programmers to a late project makes it later". Based on this, adding new developers at later stages will increase the average time taken to fix bugs. Also, the age which represents the legitimacy of the software is more popular. Popular software attracts more developers and we expect that older software will have higher number of contributions from developers. So, we control for the age of the project by calculating *Age* variable as the number of months since a project's inception at Sourceforge. To control for potential non linear effect of project age on the dependent variables, we also incorporate a variable *AgeSq*, square of the Age in the model.

Size

Size of the source code has been found to impact the complexity of the source code. Additionally, larger software is likely to receive more enhancements and more repairs than smaller software, ceteris paribus, as larger software embodies greater amount of functionality subject to change. Larger software will likely receive more repairs as well. The larger the software, the more difficult it is to test and validate the software's functionality. This implies that larger software tend to incorporate more errors. Keeping the above in mind, we control for the size of the software, by calculating a *Size* variable as the number of lines of code, while studying various relationships.

Number of downloads

The key to field testing of a software product is to try it out in as many different settings as possible. Raymond suggests that OSS developers can leverage the law of large number to identify and fix the bugs. Given enough eyeballs, all bugs are shallow. A huge user base for the software implies that – software will be tested in numerous different environments, more bugs will surface, they will be characterized and communicated efficiently to more bug fixers, the fix being obvious to someone, and the fix will be communicated effectively back and integrated into the core of the product. We capture this potential effect, of higher developers' contributions because of more reported bugs, by the number of cumulative downloads (*Downloads*) of the release of the project.

New Developer Knowledge and Skills

The literature on performance has identified individual characteristics such as knowledge and skills as antecedents of participation. Such characteristics are, however, difficult to measure, and are frequently assessed through the use of proxies, such as the level of education and experience. Curtis et al. (1979) reported that in a series of experiments involving professional programmers, the number of years of experience was not a significant predictor of comprehension, debugging, or modification time, but that number of languages known was. Accordingly, they suggested that breadth of experience may be a more reliable guide to ability than length of experience. So, to control for the effect of new developers' skills, we use *ChgDevExp*, which is approximated by the change in the team language skills by the addition of the new developer to the team over the team skills without the new developer.

Sponsorship

The incentives for programmers to participate in an open source project include ego gratification, career concern, and so on, whereas developers build proprietary software primarily for monetary compensation. However, an increasing number of open source projects have opted to receive monetary donations from organizations and users. Although some developers and projects choose to allocate part or all of the incoming donations to SourceForge, most recipients of the donations rely on the monetary supports to fund development time and other key resources that are necessary for the continuation of the projects. So, we postulate that developers getting additional monetary benefits will input extra efforts and time into comprehending and fixing the source code. Therefore, we choose to use the variable *AcceptSponsors* to capture whether a project is accepting external funds and is using monetary compensation as part of its incentive mechanism. It takes the value of 1 if the project is accepting donations and 0 otherwise.

Development Status

To capture the development stage of a project, which is typically determined by the developer in charge of the project on SourceForge, the variable *DevStatus* takes values from 1 to 6 representing development stages of Planning, Pre-Alpha, Alpha, Beta, Production/Stable, and mature respectively. The larger the value of *DevStatus*, the more mature the project is.

Results

Initial investigations revealed that the dependent variable and most of the independent variables were not normally distributed. In such case, linear regression analysis might yield biased and non interpretable parameter estimates (Gelman and Hill 2007). Therefore, as suggested by Gelman and Hill (2007), we performed a logarithmic transformation on the dependent and the not normally distributed independent variables. Investigations also showed that Project Age was highly correlated with its square. The correlations were high enough to raise concerns of multicolinearity, which if uncorrected for may lead to inflated standard errors and, in worst case, inconsistent or unstable estimates (Greene 2003). As suggested by Gelman and Hill (2007), we mean centered the variables before taking the square which reduced the correlation to acceptable level. To avoid any confusion, we represent the log transformed variables by 'ln' attached in front of the variable name. We also represent the mean centered variables by 'mc' attached in front of their name.

We also computed the Variance Inflation Factor (VIF) for all the variables to test for multicollinearity among the variables. VIF is one measure of the effect the other independent variables have on the variance of a regression coefficient (Maddala, 1988). Large VIF values indicate high collinearity. Suggested cutoffs for VIF include 10 (Studenmund, 1992) and 5.3 (Hair et al. 1992). The VIF values for the different variables in our regression analyses are reported (results shown in Table 1, 2, and 3) and in no case exceed 1.25. Although multicollinearity does exist to a certain extent, the low VIF values indicate that it is not a serious problem.

For our dependent measure, *ChgBugsReported* we test the impact of change in complexity on the number of bugs (hypothesis H1) found by estimating the parameters for the following regression model:

$$ChgBugsReported = \alpha + \beta_1 ChgMC/HE + \beta_2 lnSize + \beta_3 lnDownloads + \beta_4 AcceptSponsors + \beta_5 DevStatus$$

A positive and significant estimate of parameter β_1 would indicate that the probability of having bugs in a source code increases as the structural complexity of software increases. The results of the regression are presented in Table 1. The model shows a good fit with the data. The parameter estimate for *ChgMC* is positive and significant (β_{MC}= 0.328, $p < 0.00$). The results suggest that projects with unit increase in structural complexity experience 0.328 units increase in the number of bugs. Similar results were found for *ChgHE* (β_{HE}= 0.252, $p < 0.00$) (H1 supported).

Table 1: Regression Results for ChgBugsReported

Model	McCabe Cyclomatic (MC)			Halstead Effort (HE)		
	β	Sig.	VIF	β	Sig.	VIF
(Constant)		.000			.000	
ChgComp*	.328	.000	1.121	.252	.000	1.036
Size	.238	.000	1.147	.316	.000	1.069
Downloads	.226	.000	1.097	.216	.000	1.096
AcceptSponsors	-.174	.000	1.048	-.189	.000	1.040
DevStatus	-.067	.085	1.039	-.062	.125	1.045
R^2	0.534			0.319		

* Two separate regression analysis with MC and HE were conducted. Both the results are shown above

Next we test the impact of complexity on the number of contributions from new developers (hypothesis H2) by estimating the parameters for the following regression model:

$$ChgNewDevCommits = \alpha + \beta_1 ChgMC/HE + \beta_2 lnSize + \beta_3 lnDownloads + \beta_4 Sponsors + \beta_5 DevStatus + \beta_6 mcAge + \beta_7 mcAgeSq + \beta_8 lnSkillsChg$$

The results of the regression are presented in Table 2. The model shows good fit with the data. The parameter estimate for *ChgMC* is significantly negative (β_{MC} = -0.360, p<0.000). The results suggest that a unit increase in structural complexity decreases the contributions from new developers by 0.360 units. Regression analysis with *ChgHE* also indicated the similar results (β_{HE}= -0.241, p < 0.00) (H2 supported).

Table 2: Regression Results for ChgNewDevCommits

Model	McCabe Cyclomatic (MC)			Halstead Effort (HE)		
	β	Sig.	VIF	β	Sig.	VIF
(Constant)		.125			.042	
ChgComp	-.360	.000	1.154	-.241	.000	1.054
Size	-.155	.000	1.160	-.235	.000	1.091
Downloads	.101	.014	1.215	.117	.007	1.211
Sponsors	.331	.000	1.054	.353	.000	1.047
DevStatus	.101	.009	1.061	.098	.015	1.066
Age	.045	.260	1.161	.017	.679	1.147
AgeSq	.023	.545	1.044	.034	.388	1.043
SkillsChg	-.104	.006	1.014	-.108	.006	1.014
R^2	0.386			0.329		

Finally, we examine the impact of complexity on the time taken to fix bugs (hypothesis H3) by estimating the parameters for the following regression model:

$$ChgFixTime = \alpha + \beta_1 ChgMC/HE + \beta_2 lnSize + \beta_3 lnDownloads + \beta_4 Sponsors + \beta_5 DevStatus + \beta_6 mcAge + \beta_7 mcAgeSq + \beta_8 lnSkillsChg$$

Table 3 summarizes the results of the regression analysis. The model shows a good fit with the data. The parameter estimate for *ChgMC* is significant and positive (β_{MC} = 0.722, p<0.000), indicating that projects that experience a unit increase in structural complexity takes 0.722 units additional time to fix bugs. For *ChgHE*, the regression coefficient was found to be positive and significant (β_{HE}= 0.569, p < 0.00) indicating similar results (H3 supported).

Table 3: Regression Results for ChgFixTime

Model	McCabe Cyclomatic (MC)			Halstead Effort (HE)		
	β	Sig.	VIF	β	Sig.	VIF
(Constant)		.000			.000	
ChgComp	.722	.000	1.154	.569	.000	1.054
Size	.010	.780	1.160	.167	.000	1.091
Downloads	-.099	.005	1.215	-.124	.002	1.211
Sponsor	-.082	.012	1.054	-.114	.002	1.047
DevStatus	-.011	.742	1.061	.003	.938	1.066
Age	-.047	.166	1.161	-.002	.950	1.147
AgeSq	.001	.963	1.044	-.021	.568	1.043
SkillsChg	.069	.031	1.014	.079	.032	1.014
R^2	0.562			0.417		

While performing preliminary analysis, we noted that the variables *ChgMC* and *ChgHE* (independent variables) exhibit bimodal characteristics. Upon careful evaluation, it was found that projects that experienced an increase in complexity over its previous release formed one mode, whereas the projects that exhibited decrease in complexity over its previous release formed the other mode. To find its impact, we created another selection variable, which was coded 1 if the change in complexity was positive and 0 if the change in complexity was negative. We re-regressed the data with the new variable for all three hypotheses. Due to space limitation, only the results for *ChgMC* are summarized in Table 4.

Table 4: Regression Results

Model / Variable (p-value)	ChgBugsReported			ChgNewDevCommits			ChgFixTime		
	1a	1b	1c	2a	2b	2c	3a	3b	3c
Constant	(.000)	(.000)	(.033)	(.125)	(.000)	(.052)	(.000)	(.000)	(.221)
ChgMC	.328 (.000)	.288 (.000)	.528 (.006)	-.360 (.000)	-.502 (.000)	-.337 (.130)	.722 (.000)	.540 (.000)	.674 (.002)
Size	.238 (.000)	.220 (.000)	.102 (.450)	-.155 (.000)	-.120 (.002)	.301 (.076)	.010 (.780)	-.054 (.229)	.484 (.003)
Downloads	.226 (.000)	.249 (.000)	-.203 (.428)	.101 (.014)	.181 (.000)	.380 (.294)	-.099 (.005)	-.154 (.001)	.287 (.377)
Sponsors	-.174 (.000)	-.109 (.009)	-.307 (.068)	.331 (.000)	.268 (.000)	-.101 (.613)	-.082 (.012)	-.109 (.014)	-.138 (.448)
DevStatus	-.067 (.085)	-.061 (.128)	-.246 (.172)	.101 (.009)	.082 (.029)	-.029 (.903)	-.011 (.742)	.003 (.952)	.124 (.568)
Age				.045 (.260)	.072 (.064)	.257 (.229)	-.047 (.166)	-.070 (.116)	.166 (.384)
AgeSq				.023 (.545)	-.20 (.592)	.241 (.291)	.001 (.963)	.023 (.589)	.056 (.783)
SkillsChg				-.104 (.006)	-.069 (.062)	-.137 (.397)	.069 (.031)	.078 (.064)	.209 (.158)
R-square	.534	.367	.624	.386	.463	.561	.562	.392	.644

a = Full Model (all data points); b = +ve Change in Complexity; c = -ve Change in Complexity (MC)

Discussion and Implications

As most of the regression coefficients for McCabe's Cyclomatic number and Halstead's Effort have similar results, both the results are discussed together as Structural Complexity. Additionally, due to space limitations, the discussion of Table 4 is also limited in this paper.

Funami and Halstead (1975) found a 0.98 correlation between the structural complexity and the reported number of bugs. Fitzsimmons and Love (1978) reported the correlations ranging from 0.75 to 0.81. In our data, we also found the correlation between the number of bugs reported and the change in complexity to be 0.43. It is interesting to note that the correlation found in this study was much smaller than the correlations reported in earlier studies for non-open source software. However, it is consistent with the literature on OSS. In the context of OSS, Schröter et al. (2006) reported the correlation value in the range of 0.40.

In literature, the relationship between complexity and the number of bugs has been found to be statistically significant (Henry et al., 1981; Shen et al., 1985). Our data analysis also shows the same. The positive and significant coefficient of regression indicates that as the complexity increases, the number of bugs reported increase.

This result must be carefully interpreted as it does not imply that more bugs are reported because the software is complex. Rather, more bugs are reported because complex software tends to have more bugs.

By mining software histories of two projects, Kim and Whitehead (2006) recommended to use time taken to fix bugs as a quality measure of software. Kemerer and Slaughter (1997) also found that complex software are more frequently repaired. In our analysis, we found that complexity of software has strong positive influence on the time taken to fix bugs. Fixing a bug in one segment of the source code usually causes many adjustments in other segments (Loch et al., 2003). The more complex the software is, the more the adjustments in other segments are. As a consequence, a developer has to simultaneously understand, and recollect all such dispersed pieces in different segments. Handling all the segments together has a detrimental effect on time devoted by a developer because more time is required to follow the flow of logic within the code (Banker et al., 1998). This is supported by several empirical studies that have found that time required to fix bugs increases as the complexity increases (Boehm-Davis et al.1992, Gill and Kemerer 1991). This result has an important implication for the maintainability of the source code. When a developer becomes conscious of the amount of time taken to fix bug, there is tendency for the developer to make the code base less maintainable by producing "quick and dirty" code instead of the slower and more thought through code. Such half-baked efforts lead to a vicious cycle in which the complexity, the number of bugs, and the time taken to fix those bugs feed each other until a dead end is reached with the only option of either re-engineering the project or closing the project.

Another reason could be found in Dymo (2006) observations. Dymo mentions that most people prefer to work on software enhancements by adding features rather than working on fixing bugs. Debugging and understanding the existing code takes more resources especially when it is written by someone else. Consequently, they tend to work on new versions of the software than to continue improving old ones. In proprietary software development, a developer is bound to work on the assigned task and has no option other than to work on fixing the existing code. Whereas, in OSS, most of the work is done on voluntary basis and developers are not contracted for specific work, the developers tend to choose to develop new code as it could also bring more visibility to them in the OSS community than just fixing the existing code.

The third impact of the source code complexity analyzed in the study was on attracting contributions from new developers. Our analysis shows that the structural complexity has a strong negative influence on the number of contributions from new developers. The significant value of coefficients of regression (β_{MC}= 0.360, p-value<0.000; β_{HE}= 0.241, p-value<0.000) indicate that as the structural complexity of source code increases (over the previous release) the number of contributions from new developers decrease (over the previous release). As OSS primarily thrives upon the voluntary contributions, the project managers must actively control the source code complexity in order to attract contributions from new developers. In a complex piece of code, it takes longer for a developer to determine the flow of logic leading to slower progress of the project (Ramanujan & Cooper, 1994). Cavalier (1998) pointed that the willingness of people to continue to contribute to a project is related to the progress that is made in the project. If a large number of activities do not seem to be moving forward, participants loose interest, leading them to leave the project. This leads to a higher likelihood of activities not being completed, and ultimately, the death of the project. Such projects become inactive over time and, further, fail to attract any contributions.

The results of regression models 1b, 1c, 2b, 3b, and 3c, shown in Table 4, clearly illustrate that change in complexity has a significant impact on the change in number of bugs, change in time to fix bugs, and the change in number of contributions from new developers. Only when the complexity was found to decrease, its impact on number of contributions from the new developers was non-significant (Model 2c). The only interpretation would be that when complexity for a source code drops below a certain level, it becomes comprehendible to almost all the new developers making the variance in data to have insignificant impact on their contributions. This also, in a way, supports the argument that decreasing complexity increases the understanding of a source code, which leads to increased contributions from developers, especially new developers.

Size of the source code was expected to have similar effects. Our analysis found strong effects of size on the number of bugs and the number of contributions from new developers. It is often argued that complexity and size are strongly correlated and that could lead to the problem of multicollinearity, which tends to inflate the regression coefficients. As mentioned earlier, we tested for the problem of multicollinearity by computing the variance inflation factor and found the multicollinearity within the permissible limits.

We found that the number of downloads has strong effect on the reported number of bugs, time to fix bugs, and the number of contributions from new developers. The number of downloads indicate the popularity of a project and popular projects attract more user and developers (Krishnamurthy, 2002). As the number of user and developer

community grows, the number of eyes watching the source code increases. As Eric Raymond (1999) repeatedly mentions "to many eyes, all bugs are shallow". When source code is open and freely visible, users can readily identify flaws. The probability of finding a bug, thus, increases with the increase in the number of eyes. With these increases eyes, the number of hands working on code also increases leading to increased contributions from the new developers.

The continued development of a project, represented by the age of a project, gives software the legitimacy, reputation and attention of the community. But in our study, age of a project did not show any significant effect, linearly or non-linearly, on any of the studied dependent variables. The reason could be because a large number of OSS projects on SourceForge are in the early stages of development. This could be attributed to the ease with which new projects can be started. A new project takes only few minutes to start, and need no thoughts about the outcomes. Such projects become inactive over time and have almost zero contributions from the developer community. It could be argued that age could bring the legitimacy, reputation, and attention only if the project is active. Another indication of continued development, the development status of a project, which is an indication of the activity, was also studied and was found to have a significantly positive impact on the number of commits from new developers only. In OSS literature, development status has been shown to have a positive impact on project's popularity. Al Marzouq et al. (2005) argue that a project attracts more developers as the software becomes more stable. In turn, these new developers bring effort and contribution that improves the software. A growth cycle starts that feeds both the community and development of the software. This growth cycle creates a network effect that is associated with the size of the community.

Lakhani and Wolf (2005) showed that developers receiving money in any form spend more time working on OSS than their peers. Similar results were shown by our data set. We found that the projects that have any form of sponsorship have higher number of contributions from new developers. We also found that such projects had less number of bugs and took lesser time to fix the bugs. This clearly indicates that developers are receptive to external stimuli such as monetary reward. Henkel (2006) illustrated a similar impact of external sponsorship on the development of applications for Linux, one of the most successful OSS project. Henkel noticed that most contributors in the field of embedded Linux are salaried or contract developers working for commercial firms. Google Summer of Code is already implementing organizational sponsorship to attract new developers (Google, 2008). Developers joining this would want to advertise their skills to Google, a potential employer. However, the success of the program is yet to be determined.

Developers' skills and experience were tested using a surrogate measure. The change in team skills with the addition of new developers was found to have significant influence on the number of contributions from new developers and the time taken to fix bugs. However, both the control variables showed have impact in the direction opposite of what was expected. We believed that as the new developers increase, the number of contributions will increase and the time taken to fix bugs will decrease accordingly. The opposite directions of the relations indicate that with the increase in number of skills, the overall time to fix bugs increases and the total relative new contributions decrease. The logical reasoning would be that either developers are *just* joining the development team without actually contributing towards the project development or the amount of contributions are not proportionate to the number of skills they possess. This is consistent with commonly argued problem in OSS development that OSS development follows a Pareto law; i.e. a small amount of developers of around 20% is responsible for a huge amount of the work done (around 80%).

Contributions

The structural complexity can be measured for the current software release, and therefore serve as useful planning tool for the next release. In addition, it should be noted that the hypotheses regarding the structural complexity were supported by the models after having controlled for various factors. Therefore, the results found here cannot be seen as an artifact due to possible correlation with these other factors.

The most important contribution of the paper is the strong support for the relation between the structural complexity and the contributions from new developers. Even though the results accord with the intuition, this is the first study to test the relationship empirically. Our models indicate that, on an average, OSS development projects with high structural complexity are significantly associated with increased bugs and repair time and decreased contributions from new developers. This suggests that project administrators may wish to implement guidelines for upper bounds of complexity during development and could recommend that software releases at no stage should exceed these guidelines. However, no such standard guidelines could be implemented for all software development projects.

Developers and administrators interested in software development need to set their own standards like NSA standard, which is derived from the analysis of 25 million lines of software written for NSA.

The project administrators for OSS projects can learn the importance of controlling complexity. As recommended by Lehman (1985), the results suggest possible strategies for not only planning for controlling complexity, but actively attempting to reduce it. As software projects progress, they become more complex making it difficult to understand and manage (Fitzgerald, 2005). Project administrators also need to be careful about subsequent changes between different releases. Such changes can have strong impact on projects (Samoladas et al., 2004). If such changes are not well monitored, it could lead to the ripple effect. Ripple effects refer to the phenomena of changes made to one part of the software affecting other parts of the software. Lehman's operating system example clearly shows the ripple effect since the percentage of modules changed in Release 15 is 33% while the percentage of modules changed in Release 19 is 56%. The OSS development, which primarily thrives from voluntary contributions, must keep a close watch on the structural complexity of the program in order to attract contributions from new developers.

Another important contribution of this research is for the organizations involved in or interested in getting involved in OSS development. Our results indicate that, on the contrary to OSS ideological beliefs, by offering a monetary reward in a specific project, organizations may successfully attract increased contributions from the OSS community.

Limitations

Although sample size is by far large enough to guarantee statistical validity, the choice of sample population might have affected the outcomes of the study.

It could be argued that the change log only records the committer; whether the developer of the code is always acknowledged is uncertain.

Do all bugs get reported? It is possible that some of bugs are fixed and are never reported.

References

Adamov, R. and Richter, L. “A Proposal for Measuring the Structural Complexity of Programs,” *Journal of System Software* (12: 1), Apr. 1990, 55-70.

Al Marzouq, M., Zheng, L., Rong, G., and Grover, V. “Open Source: Concepts, Benefits, and Challenges,” *Communications of the Association of Information Systems* (16), Nov 2005.

Banker, R. D., Datar, S., Kemerer, C., & Zweig, D. “Software Complexity and Maintenance Costs,” *Communications of the ACM* (36: 11), Nov 1993.

Banker, R., Davis, G. and Slaughter, S. A. “Software Development Practices, Software Complexity, and Software Maintenance Effort: A Field Study,” *Management Science*, (44: 4), 1998.

Boehm, B. *Software Engineering Economics*, Prentice-Hall, New York, 1981.

Cavalier, F. J. “Some Implications of Bazaar Size” 1998. *Available at http://www.mibsoftware.com/bazdev/* accessed 8 May 2006.

Crowston, K., H. Annabi, J. Howison. “Defining Open Source Software Project Success,” in *Proceedings of International Conference on Information Systems*, Seattle, WA, (2003).

Curtis, B., Sheppard, S. B., Milliman, P., Borst, M. A., & Love, T. “Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics,” *IEEE Transactions On Software Engineering*, (5: 2), March 1979.

Darcy, D. P., Kemerer, C. F., Slaughter, S., & Tomayko, J. E. “The Structural Complexity of Software an Experimental Test,” *IEEE Transactions on Software Engineering* (31: 11), 2005.

Dymo, A. “Open Source Software Engineering,” in *II Open Source World Conference,* Málaga. 15-17 Feb 2006.

Feller, J. & Fitzgerald, B. *Understanding Open Source Software Development*, London: Addison-Wesley, 2001.

Fitzgerald, B. “Has Open Source Software a Future?,” in *Perspectives on Free and Open Source Software*, MIT Press, Eds Feller, J., Fitzgerald, B. & Hissam, S. A., 2005, 93-106.

Fitzsimmons, A. & Love, T. "A Review and Evaluation of Software Science". *ACM Computing Surveys*, (10: 1) Mar. 1978, 3-18.

Fjeldstad, R. & Hamlen, W. "Application Program Maintenance-Report to Our Respondents," *Tutorial on Software Maintenance*, (13: 27), IEEE Computer Society, 1983.

Funami, Y., and Halstead, M.H. "A Software Physics Analysis of Akiyama's Debugging Data". CSD-TR-144, Purdue University, Lafayette, Ind., May 1975.

Gelman, A., & Hill, J. *Data Analysis Using Regression and Multilevel/Hierarchical Models*, Cambridge University Press, 2007.

Gill, G. & Kemerer, C. "Cyclomatic Complexity Density and Software Maintenance Productivity," *Transactions on Software Engineering*, (17: 12), 1991, 1284–1288.

Godfrey, M.W., and Tu, Q. "Evolution in Open Source Software: A Case Study," in *Proceedings of International Conference Software Maintenance*, San Jose, California, 2000, 131-142.

Godfrey, M.W., and Tu, Q. "Growth, Evolution, and Structural Change in Open Source Software," in *Proceedings of International Conference on Software Engineering*, 4th International workshop on Principles of Software Evolution, 2002, ACM Press 103-106.

Gonzalez-Barahona, J.M. Perez, M.A.O., Quiros, P.d.l.H., Gonzalez, J.C., and Olivera, V.M. "Counting Potatoes: The Size of Debian 2.2," *Upgrade Magazine*, (II: 6), 2001.

Google. *Google Summer of Code*, 2008, http://code.google.com/soc/2008

Gorla, N., & Ramakrishnan, R. "Effect of Software Structure Attributes Software Development Productivity," *Journal of Systems and Software*, (36: 2), 1997, 191-199.

Greene, W. *Econometric Analysis*, 2nd ed., MacMillan Publishing Company, New York, 1993, pp 791.

Grewal, R., G.L. Lilien, G. Mallapragada. "Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems," *Management Science* (52: 7), 2006, 1043– 1056.

Hair, Jr., J. F., Anderson, R. E., Tatham, R. L. & Black, W. C. *Multivariate Data Analysis*, Macmillan, NY, 1992.

Halstead, M.H. *Elements of Software Science*. Elsevier North Holland, 1977.

Hartmanis, J., "On Computational Complexity and the Nature of Computer Science," *Communications of the ACM*, (37: 10), October 1994.

Henkel, J. "Selective Revealing in Open Innovation Processes: The Case of Embedded Linux," *Research Policy*, (35: 7). 2006, 953-969.

Henry, S., Kafura, D. & Harris, K. "On the Relationship Among Three Software Metrics," *SIGMETRICS: Performance Evaluation Review*, (10), Spring 1981, 81-88.

Jones, C. Programming Productivity. McGraw Hill, New York, 1986.

Kearney, J. K., Sedlmeyer, R. L., Thompson, W. B., Gray, M. A., & Adler, M. A. "Software Complexity Measurement," *Communications of ACM*, (29: 11), Nov. 1986, 1044-1050.

Kemerer, C. F. &. Slaughter, S. A. "Determinants of Software Maintenance Profiles: An Empirical Investigation. Software Maintenance," *Research and Practice*, (9), 1997, 235–251.

Kemerer, C. F. "Software Complexity and Software Maintenance: A Survey of Empirical Research," *Annals of Software Engineering* (1), 1995, 1-22.

Kim, S., Whitehead, E. J., & Bevan, J. "Analysis of Signature Change Patterns," in *Proceedings of the 2005 international workshop on Mining software repositories*, St. Louis, MO, May 17-17, 2005, 1-5.

Klemola, T. "A Cognitive Model for Complexity Metrics," in *Proceedings of the 4th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, CRIM, 2000.

Krishnamurthy, S. "Cave or Community? An Empirical Examination of 100 Mature Open Source Projects," *First Monday*, 2002.

Lakhani, K. R., & Wolf, B. "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects," *Perspectives on Free and Open Source Software*. MIT Press, 2005.

Lehman, M. M., & Belady, L. A. *Program Evolution, Processes of Software Change*, Academic Press Inc. Orlando Florida 1985.

Littman, D. C., Pinto, J., Letovsky, S. and Soloway, E. "Mental Models and Software Maintenance", *The Journal of Systems and Software*, (7), 1987, 341-355.

Loch, C., Mihm, J., & Huchzermeier, A. "Concurrent Engineering and Design Oscillations in Complex Engineering Projects," *Concurrent Engineering*, (11: 3), 2003, 187-199.

Maddala, G. S. *Introduction to Econometrics*, Macmillan Publishing Company; NY, 1988.

Madey, G., Freeh, V., & Tynan, R. "The Open Source Software Development Phenomenon: An Analysis Based on Social Network Theory," in *Proceedings of the 8th AMCIS*, 2002.

McCabe, T. J. "A Complexity Measure", *IEEE Transactions on Software Engineering*, (2: 4), December 1976.

Nichols, D. M and Twidale, M. B. 'The Usability of Open Source Software," *First Monday*, (8: 1), Jan 2003.

Opensource.org. "The Open Source Definition (Version 1.9)", 2002 at http://www.opensource.org/docs/definition.html, accessed 5 May 2006.

Ramanujan, S. and Cooper, R. "A Human Information Processing Approach to Software Maintenance," *Omega*, (22: 2), 1994, 185–203.

Rana, Z.A., Khan, M.J., and Shamail, S. "A Comparative Study of Spatial Complexity Metrics and Their Impact on Maintenance Effort," *IEEE—ICET 2006, 2nd International Conference on Emerging Technologies*, 2006.

Raymond, E. S. "The Cathedral and the Bazaar", 1999, at *http://tuxedo.org/~esr/writings/cathedral-bazaar/*

Rilling, J. & Klemola, T. "Identifying Comprehension Bottlenecks Using Program Slicing and Cognitive Complexity Metrics," in *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, 2003.

Samoladas, I., Stamelos, I., Angelis, L., and Oikonomou , A. "Open Source Software Development Should Strive For Even Greater Code Maintainability," *Communications of the ACM*, (47: 10), 2004.

Schröter, A., Zimmermann, T., Premraj, R., & Zeller, A. "If Your Bug Database Could Talk... ", *ISESE* 2006.

Shen, V.Y., Yu, T., Thebaut, S.M., & Paulsen, L.R. "Identifying Error-Prone Software – An Empirical Study," *IEEE Transactions on Software Engineering*, (11: 4), 1985, 317-323.

Siedlok, F. *Characteristics and applicability of Open Source-based Product Development Model in Other than Software Industries*, M.S. Thesis, University of Durham Business School, 2002.

Smith, N., Capiluppi, A., & Ramil, J. F. "Agent-based Simulation of Open Source Evolution," *Software Process Improvement and Practice*, (11), 2006, 423-434.

Stamelos, I., Angelis, L., Oikonomou, A., & Bleris, G. L. "Code Quality Analysis in Open Source Software Development," *Information Systems Journal*, (12), 2002, 43-60.

Stewart, K., Ammeter, A., Maruping, L. "A Preliminary Analysis of the Influences of Licensing and Organizational Sponsorship on Success in Open Source Projects," in *Proceedings of the 38th Hawaii International Conference on System Sciences*¸ 2005.

Studenmund, A. H. *Using Econometrics: A Practical Guide*, Harper Collins, New York, 1992.

Thaik, R. W., Lek, N., and Kang, S-M. "A Router Using Zero-One Integer Linear Programming Techniques for Sea-of-Gates and Custom Logic Arrays," *IEEE Transaction on Computer Aided Design,* (11: 12), Dec 1992.

von Hippel, E., G. von Krogh. "Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science," *Organization Science*, (14: 2), 2003, 209–225.

von Krogh, G. Spaeth, S., and Lakhani, K. R. "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study," *Research Policy*, (32), 2003, 1217–1241.

Weyuker, E. J. "Evaluating Software Complexity Measures," *IEEE Transactions on Software Engineering*, (14: 9), 1988, 1357-1365.

Wood, R.E. "Task Complexity: Definition of the Construct.," *Organizational Behavior and Human Decision Processes*, (37), 1986, 60-82.

Xu, J., Y. Gao, S. Christley, G. Madey. "A Topological Analysis of the Open Source Software Development Community," in *Proceedings of the 38th HICSS*, 2005.