By Ioannis Samoladas, Ioannis Stamelos, Lefteris Angelis, and Apostolos Oikonomou

Open Source Software Development Should Strive for EVEN GREATER CODE MAINTAINABILITY

A study of almost six million lines of code tracks how freely accessible source code holds up against time and multiple iterations.

Open source software (OSS), a term first coined in 1998 [3], has spurred many products in its short history, most notably the Linux operating system, the Apache server, and the so-called killer-apps such as BIND and Sendmail.

But what really defines software as OSS? The answer is not as simple as it may seem. Generally, OSS is a software product distributed by license, which conforms to the Open Source Definition [11], the best known of which are GNU General Public License (GPL) and Berkeley Software Distribution (BSD). Unlike the traditional closed source software (CSS), OSS can be freely used, modified, and redistributed. Its source code is also freely accessible. Today, it is common to find CSS projects evolve into OSS in order to obtain its benefits. Occasionally, OSS projects become CSS, but usually for different reasons. An extensive explanation and analysis of OSS development can be found in [3].

ILLUSTRATION BY PETER HOEY

An OSS project begins simply. A single developer or group of developers writes the first version of the software and make it freely available over the Net, inviting other developers to participate and contribute pieces of code. The evolution of the product being developed is normally coordinated by its creators. Among the responsibilities of the coordinators is configuration management, release scheduling, deciding which code contributions will be accepted, and other management activities. We should note that sometimes this group of coordinators has a great deal of power (strong control on code submission, issuing of clearly defined specifications), which makes the process look like a traditional one. It can be said that the vitality and the success of the projects depend strongly on the coordination of the process.

OSS development emphasizes the maintainability of the software released. Making software available on the Internet allows developers around the world to contribute code, adding new functionality (parallel development), improving the present one, and submitting bug fixes to the current release (parallel debugging). The coordinators of the project ultimately decide the contributions and bug fixes to accept, incorporating them into the main code and final product release. Indeed, the same process of code contribution and bug fixing is continued in this circular manner.

This type of development has both advantages and disadvantages. In various cases OSS seems to have solved many of the problems of traditional software engineering methods, since it has been possible to produce reliable, high quality, and low-cost software in a brief amount of time. The existence of a large pool of testers and developers facilitates debugging and the true peer review of the code results in better code. The availability of the source code allows someone to make modifications to meet his or her own needs, while the lack of black boxes is important for someone to inspect the code for its correctness and to assure dependability issues. However, very few empirical studies have been available to support or reject these claims [8, 10].

The disadvantages of OSS development include absence of complete documentation or technical support. Moreover, there is strong evidence that projects with clear and widely accepted specifications, such as operating systems and system applications, are well suited for the OSS development model. However, it is still questionable whether systems like ERP could be developed successfully as OSS projects. (For more on the advantages and disadvantages of OSS development, see [3, 5].)

A ready interpretation of the OSS development process is that of a perpetual maintenance task. Developing an OSS system implies a series of frequent maintenance efforts for debugging existing functionality and adding new ones to the system. These two forms of maintenance are known as corrective and perfective maintenance, respectively.

Since it is still difficult to monitor the open source process, it is reasonable to measure and assess the resulting product, that is, the delivered code. The purpose of this article is to report and discuss the results of a case study examining the maintainability of the source code delivered by open source development. To this end, we measured the source code of five OSS projects, which in several cases involved development in CSS fashion. We have assessed the results according to a maintainability index derived from analysis of industrial CSS systems. We contend maintainability is the core quality issue in OSS development.

Source Code Measurement

A well-known conjecture in modern software engineering is that external quality characteristics are correlated to internal quality characteristics. The measurement of source code provides useful information for the assessment of its quality, predicting to some extent the external system quality characteristics, such as maintainability, reliability, extensibility, and portability.

Measurements may be used to obtain a picture of the quality both of a single component and of an entire program. Typical software metrics are the size of the code (measured in lines of code, number of statements, and so on) and the code complexity (measured through complexity figures such as the cyclomatic complexity). Setting an acceptable range for each metric considered assesses code quality, in which every measured value should fall in. A set of such ranges along with a number of efficient program-writing rules defines a programming standard, that is, any comparison to these values can lead to a representative view of the quality of the tested programs. Software organizations interested in software development set their own standards. An example is the NSA standard [1], which is derived from the analysis of 25 million lines of software written for NSA.

In our analysis we used a set of measurement programs available in the Debian GNU/Linux release, and a number of Perl scripts for managing measurement results. The metrics considered are among the most widely reported and used in the literature and are listed here:

- Number of lines of code (LOC) measures the physical size of the program code, excluding blank lines and comments.
- Percentage of lines of comments with respect to the number of lines of code (PerCM) describes the self-descriptiveness of the code.
- Halstead Volume (V). Halstead [4] defined four metrics that can be measured from a program's source code: *n1* (the number of distinct operators), *n2* (the number of distinct operands), *N1* (the total number of operators) and *N2* (the total number of operands). Based on them, he defined program vocabulary *n* (given by $n = n1 + n2$) and program length *N* (given by $N = N1 + N2$). Finally, he defined *Volume*, a composite metric given by the formula $V = N * (LOG_2\, n)$. Halstead Volume provides an alternative measure for the size of a program.
- Cyclomatic Complexity *V(g)*. Proposed by McCabe [7], this metric counts the number of independent paths in the control flow graph of a program component. Its value depends on the number of branches caused by conditional statements (*if-then-else*). It measures the structural complexity of the component.

It would be quite difficult to reach a consensus about the metric ranges that OSS code should respect in order to define an OSS-specific programming standard. In this study, rather than comparing with a predefined standard, we preferred to directly compare maintainability measures of open source with those obtained for closed source software, or simply to observe the trend of these measures and derive conjectures about improvement or deterioration of code quality. Because it may be difficult to obtain a single picture about maintainability from many measures derived from many different metrics, we preferred to use a different approach in our study based on the composite metric, Maintainability Index (MI), chosen by SEI [6] as the most suitable tool for measuring the maintainability of systems with high-quality requirements. We chose MI because we believe that OSS should conform to such standards in order to compete with CSS. Moreover, various OSS projects, including some we studied, are often part of large scale, critical applications. MI following this formula:

$$MI = 171 - 5.2\ln(avgV) - 0.23\,avgV(g) - 16.2\ln(avgLOC) + 50\sin(\sqrt{2.4\,avgPerCM})$$

where *avgV* stands for "average Halstead Volume per module" and *avgV(g)*, *avgLOC*, and *avgPerCM* are defined in a similar way.

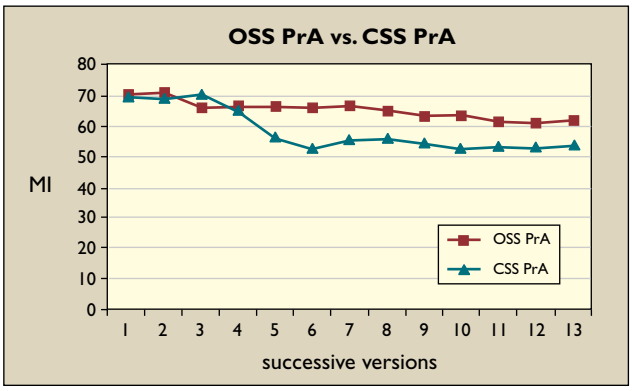

Figure 1. MI figures for project PrA.

MI measures maintainability by taking the size, the complexity, and the self-descriptiveness of the code into account. MI may change because of new code added to the existing source code due to bug fixing or other corrective actions. However, since MI is based on average values, it is relatively independent of the absolute size of these changes and may be used to compare systems of different size. The coefficients of the formula for MI have been calibrated by Oman [9] on various software systems maintained by Hewlett-Packard, and MI proponents verified this form of the MI equation generally fits other industrial-sized software systems [12]. High MI values indicate high maintainability.

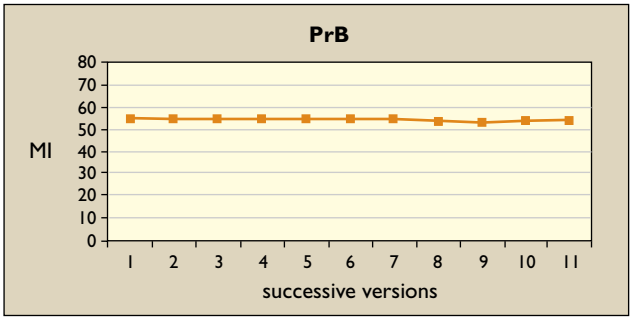

Figure 2. MI figures for project PrB.

Measuring and Assessing Open Source Code

We examined five active and popular OSS projects for this study; for a total of 5,856,873 LOCs. Table 1 provides typical project information (application type, total release code size) and a brief verbal description of the project's evolution. We do not provide the actual names of the projects in order to adhere to standard software engineering ethics [2], therefore a mnemonic code is assigned to each project.

For each project, we measured a number of major releases, obtaining a history of the evolution of the source-code quality. We present here only the results regarding maintainability index (Figures 1, 2, and 3),

since they allow fast analysis of the projects.

Project PrA was produced originally as an OSS product. Eventually, the steering group decided to exploit the system commercially and initiated a project of the close source type (denoted as CSSPrA). However, the original OSS system (OSSPrA) continued to evolve. The two projects developed essentially the same functionality in the period we performed our study. This fact produced an interesting situation that allows us to make a direct comparison between an OSS and a CSS project developing the same object. Figure 1 provides very interesting measurement results: it seems that while maintainability deteriorates in both projects, the OSS version does significantly better than its CSS counterpart.

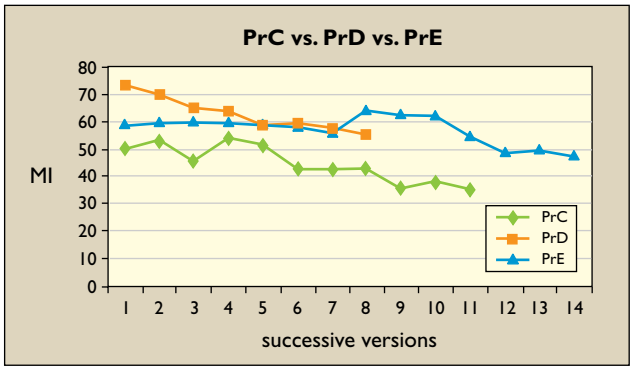

Figure 3. MI figures for projects PrC, PrD, PrE.

Project PrB was originally a CSS project. Eventually the company developing the system rendered its source code free, giving birth to an OSS project. Project PrB provides a situation in which a CSS object evolves in OSS fashion. Figure 2 gives an idea of what happened to the maintainability of the system across the subsequent initial releases. As a result of the project's fairly early development stage, maintainability of new versions does not show significant differences with respect to the first release, which is, in fact, the CSS release. Further analysis and monitoring of this application is needed in order to keep track of its maintainability status. Nonetheless, it can be said this case showed that switching from CSS to OSS did not deteriorate maintainability.

Project Mnemonic Code	Application Type	Total Code Size (KLOCs)	No. of releases measured	Project Evolution Path
OSSPrA	Operating system application	343	13	OSS project that gave birth to a CSS project while still evolving as OSS
CSSPrA	Operating system application	994	13	CSS project initiated from an OSS project and evolved as a commercial counterpart of OSSPrA
PrB	Operating system application	860	10	CSS project that opened its code and was transformed to an OSS project
PrC	Programming language	1050	11	Pure OSS project
PrD	Database management system	1411	8	Academia project that gave birth to an OSS project
PrE	Internet application	1198	14	Pure OSS project

Table 1. The top five open source projects in the study and their salient characteristics

Projects PrC and PrE are typical OSS projects: they were initiated as OSS projects and continued to evolve as such. PrD was initially an academia project that evolved to an OSS project (only the OSS releases were considered). Figure 3 provides the maintainability history of these projects in a single diagram: maintainability drops gradually in all three cases. Only PrE managed to improve maintainability for a limited number of releases. It seems these projects behave as would any CSS project.

Code Quality Needs Monitoring and Improvement

Although only provisionary conclusions may be drawn from such results, it is our opinion the results of this study may be interpreted as follows:

1. Using tools such as MI derived for measuring CSS quality, OSS code quality appears to be at least equal and sometimes better than the quality of CSS code implementing the same functionality. This may be due to the motivation of skilled OSS programmers to compete with skilled CSS programmers. High motivation may be considered an important advantage of OSS compared to CSS.

2. OSS projects may need careful individual analysis because of abrupt changes between subsequent releases due to decisions made by OSS project coordinators. Such changes have a strong impact on OSS software configuration. Structural code analysis may also provide very interesting feedback at the individual module or component level: according to Pareto's Law,[1] a small percentage of the software will be responsible for the majority of the problems, so it is reasonable to expect that 20% of components will produce about 80% of the maintainability problems. The risk-prone components may be identified through the analysis tools we described here.

3. OSS code quality seems to suffer from the very same problems that have been observed in CSS projects. Maintainability deterioration over time is a typical phe-

[1]Pareto's Law, frequently used in software engineering, states that 80% of the benefit can often be obtained with 20% of the work; however the remaining 20% benefit takes 80% of the work.

nomenon and produces legacy CSS systems. It is reasonable to expect similar behavior from the OSS projects as they age: the OSS approach will produce legacy systems in much the same way CSS has done. As a consequence, appropriate reengineering actions may be necessary for OSS systems too. In other words, preventive maintenance may be the third type of maintenance that must be taken into account by OSS proponents.

More empirical analysis is needed to consolidate the findings of this study. We will continue monitoring the quality of these projects and we will extend our analysis to other OSS projects that present interesting characteristics and allow comparison with CSS development. However, it is important to integrate the structural view of OSS quality with the view of OSS system users' perceived quality. C

References

1. Drake, T. Measuring software quality: A case study. *IEEE Computer 29,* 11 (1996), 78–87.
2. El-Emam. Ethics and open source. *Empirical Software Engineering 4,* 6 (2001), 291–292.
3. Feller, J., and Fitzgerald, B. *Understanding Open Source Software Development.* Addison Wesley, Reading, PA, 2002.
4. Halstead, M. H. *Elements of Software Science.* Elsevier, North-Holland, 1975.
5. *IEEE Software 16,* 1 (Jan/Feb 1999).
6. Maintainability index technique for measuring program maintainability. Software Technology Review, SEI; www.sei.cmu.edu/str/descriptions/mitmpm_body.html.
7. McCabe, T. A complexity measure. *IEEE Trans. on Software Engineering 2,* 4 (1976), 308–320.
8. Mockus A., Fielding R., and Herbsleb J. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Software Eng. and Meth. 11*, 3, (2002), 309–346.
9. Oman, P. and Hagemeister, J. Constructing and testing of polynomials predicting software maintainability. *J. Systems and Software 24,* 3 (1994), 251–266.
10. Schach S.R., Jin B., Wright D.R., Heller D.Z., and Offutt A.J. Maintainability of the Linux kernel. In *IEE Proceedings—Software Engineering 149,* 1 (2002), 18–24.
11. The Open Source Initiative. Open source definition, version 1.9; www.opensource.org/docs/definition.php.
12. Welker, Kurt D. and Oman, P.W. Software maintainability metrics models in practice. *Crosstalk, Journal of Defense Software Eng. 8,* 11 (1995), 19–23.

Ioannis Samoladas (ioansam@csd.auth.gr) is a Ph.D. candidate in the Department of Informatics at Aristotle University of Thessaloniki, Thessaloniki, Greece.
Ioannis Stamelos (stamelos@csd.auth.gr) is an assistant professor in the Department of Informatics at Aristotle University of Thessaloniki, Thessaloniki, Greece.
Lefteris Angelis (lef@csd.auth.gr) is a lecturer in the Department of Informatics at Aristotle University of Thessaloniki, Thessaloniki, Greece.
Apostolos Oikonomou is pursuing a Masters of Science degree in information systems at the Imperial University, London, U.K.