Investigating Quality in Large-Scale Open Source Software

Uzma Raja, Evelyn Barry
Texas A&M University
College Station, TX
1-979-845-6995
{uraja,ebarry}@mays.tamu.edu

ABSTRACT
Open Source Software (OSS) development and use has increased significantly over recent years. Therefore, there is a need to analyze and understand these projects. Software quality is an important characteristic effecting overall system lifecycle cost, performance and useful life. The existing models for software quality are based on empirical analysis of propriety source software (PSS), and need to be verified in OSS. Research on PSS has revealed that software quality declines, as it ages. Part of this decline is associated with the lifecycle maintenance activities that introduce change in the size and complexity of the system, while introducing software errors into modified system. Lifecycle maintenance activities in OSS systems are processed under a very different paradigm. We are interested in investigating the effects of maintenance activities on OSS project outcomes. Linux is one of the most popular and complex OSS project available. In our research, we investigate the characteristics of Linux source code. In this position paper we present some preliminary results of the effects of various types of maintenance activities on quality of Linux software.

Keywords
Software Quality, Software Maintenance, Open Source, Linux Operating System

1. INTRODUCTION
Over 30 years ago Fred Brooks wrote about the need for a silver bullet to help software engineers and IS professionals speed the process of creating and maintaining software systems. Now, even in the 21st century, software systems continue to be expensive to build and challenging to maintain. In recent years there has been a growth in the re-use of source code, and the use of off-the-shelf software. In many of these cases the source code is unavailable for the user to analyze. If the software does not perform as expected the IS manager and project leaders are at the mercy of the owner of the source code as to whether or not any changes can, or will, be made.

The OSS movement is changing the way software is developed, maintained and updated [1-3]. OSS systems are commonly developed as free ware and are available at little or no cost. Recently, OSS have been used in more and more software systems. For example, Apache, an OSS project, has 66.04 % of the web server market share (Netcraft 2004). The number of people using Linux Operating system is estimated at 150,000 based on registered Linux users. The reason for the increase in OSS use is not only the low cost and easy access to the code, but also consistently high software quality [2]

Current research on software quality is based on empirical data from software systems developed through traditional methods. The most frequently cited software quality measures are counts of system failure, i.e. counts of abnormal terminations (abends), and counts of nonconformance to user-defined requirements [4]. With OSS, we need to re-examine the metrics used to assess software system quality.

Linux is one of the most successful OSS projects and is being used with many commercial applications [3, 9]. In this research we explore some important software characteristics that contribute to consistent software quality in Linux. In the following sections, we present some important quality characteristics and maintenance activities in OSS. We then develop a model for quality in OSS and empirically test it, using historical data for the Linux modifications and the corresponding lifecycle maintenance change logs.

2. QUALITY CHARACTERISTICS IN OSS
Software quality is one of the most important metrics for the success of a software project. Barry Boehm defines software quality as " achieving high levels of user satisfaction, portability, maintainability, robustness and fitness for use" [10]. Jones refers to quality as " the absence of defects that would make software either stop completely or produce unacceptable results" [11]. These definitions of software quality cannot be applied directly to OSS. Unlike CSS, user requirements are not formally available in OSS. Existing quality models provide a list of quality carrying characteristics that are responsible for high quality (or otherwise) of software. We can divide OSS into two major categories: Type-1: Projects that are developed to replicate and replace existing CSS software; and Type-2: Projects initiated to create new software that has no existing equivalent CSS software. Linux is an example of Type-1 software, which was originally developed as a replacement for UNIX. Protégé, ontology development software is an example of Type-2 software. We identify some important quality carrying characteristics in OSS.

2.1 Reliability
Reliability refers to the persistence of the output provided by the system. The reliability factor is concerned with the behavior of the software. It is the extent to which it performs its intended functions with required precision. The software should behave as expected in all possible states of the environment. Although OSS is available free of cost, yet such software needs to have a minimum operational reliability to make it useful for any application. Reliability has a significant effect on software quality, since the user acceptability of a product depends upon its ability to function correctly and reliably [12].

2.2 Functionality

Functionality refers to providing minimum functions as required by the user. For Type-1 OSS there are no formal functionality requirements, yet there will be a certain level of expectations in terms of its functionality compared to an existing CSS. New users will adopt Type-1 software, if it provides the basic functionality of its CSS equivalent. In case of Type-2 OSS, there is no existing software to derive functional requirements from, thus new users will be defining such requirements according to their own needs. A Type-1 OSS will be considered of a high quality if it provides basic functionality of its CSS equivalent. On the other hand Type-2 OSS will be considered of a high quality if it provides the functional requirements of its active users at a steady pace.

2.3 Availability

Availability means that the software should be available to the user at minimum required time. This attribute is becoming very critical especially in the area of e-services. Depending upon the context, availability will have a different meaning, for example in mission critical space software, availability will be in terms of processing power availability for computation of complex algorithms, while in the field of e-services, it will be the availability of software system to support hardware for 24/7 service (hacker attacks will reduce availability of system). In open source, free availability of code may create a conflicting situation. The software source code is available globally. This means that users can identify potential vulnerabilities, but it also implies that hackers can also exploit these vulnerabilities easily. Hence the quality will depend upon how vigilant the active users are in detecting vulnerabilities and protecting the software from malicious hacking activities. Projects that fail to provide secure software will experience decline in quality and users.

2.4 Maintainability

Maintainability in general refers to the ability to maintain the system over a period of time. This will include ease of detecting, isolating and removing defects. Additionally, factors such as ease of addition of new functionality, interface to new components, programmers ability to understand existing code and test team's ability to test the system (because of option like test instructions and test points) will enhance the maintainability of a system. Maintenance is a huge cost driver in software projects. OSS is downloaded and used by a global community of users. There are no face-to-face interactions among the maintainers of the software. They have to rely upon the documentation with in the source code and on communication through message boards. Therefore OSS is required to be highly maintainable. Lack of proper interface definition, structural complexity and insufficient documentation in an existing version of OSS can discourage new contributions. Since participation is voluntary, low maintainability will generate minimum participation of active users and hence will have a negative effect on quality.

2.5 Reusability

In CSS there is use of existing modules for multiple projects. Development costs are a major factor affecting this quality characteristic in CSS domain. In OSS, there is no development or maintenance cost. OSS communities encourage development and use of reusable modules that can be shared and implemented easily. OSS that employs reusable modules will attract more contributions and maintain a high quality.

3. SOFTWARE MAINTENANCE IN OSS

Software maintenance is the modification of a software product after completion of development, to correct faults, improve performance, or to adapt to a changed environment (ANSI/IEEE, 1983). According to Lehman et al. "e-type programs[1] will be perceived as of declining quality unless rigorously maintained and adapted to a changing operational environment;" hence system quality is at constant decay [14-16]. One of the major contributors to this decline is lifecycle maintenance activity [16]. In software developed and maintained through conventional methodologies, the effort spent on maintenance represents a majority of the costs incurred during the useful life of a system. Researchers studying lifecycle software costs have shown that software maintenance activities account for as much as 90% of the lifecycle cost of a software system [17]. Extensive research has been done on how maintenance effort increases as a system ages [14]. As more and more organizations are adapting OSS at various levels, it is critical to investigate the factors that affect the maintenance activities in OSS domain. Unlike PSS, there are no contractual obligations for maintenance; hence maintenance costs could be significant in the form of lost business and non-availability of functions, if the project fails to grow. Maintenance activities can be divided into the following four categories.

3.1 Corrective maintenance

Corrective Maintenance is performed to remove a defect. It is performed once a defect has occurred. It is performed at unpredictable time, since there is no prior knowledge of the presence of defect.

3.2 Adaptive maintenance

Adaptive maintenance is the change in the software to accommodate changes to the environment in which it operates (e.g. new hardware platforms or new business rules).

3.3 Perfective Maintenance

Perfective maintenance is the addition of new functionality It involves making changes to improve some aspect of the system, even when the changes are not suggested by faults.

3.4 Preventive Maintenance

Preventive Maintenance involves changing some aspect of the system to prevent failures. Preventive maintenance usually results when a programmer finds an actual or potential fault that has not yet become a failure and takes action to correct the fault before damage is done. This type of activities will reduce he complexity of the software and improve quality.

4. RESEARCH MODEL

OSS projects exhibit high quality despite absence of defined user, requirements, costs or schedules. OSS is characterized by frequent

[1] E-type programs are programs that continually change, updated and evolved [13]

voluntary contributions from active users all across the globe. The development and maintenance methods in OSS and PSS are inherently different; hence we need new models that can explain the factors affecting various software characteristics in OSS systems. The usual metrics for software quality cannot be collected for OSS systems due to the difference between the software processes for OSS and PSS systems.

We believe that during different life cycle phases, importance of the quality characteristics for active user will change. Initially, users will start using the system, if it meets their functional requirements. Hence functionality will be a critical characteristic. Since the software is free, the users may be relaxed on reliability expectations of the software. Most of the OSS maintains production and experimental versions in parallel, so users have the option for using a more stable version if they are concerned with reliability or a more functional version that might not be very reliable and stable at moment.

As the usage of OSS progresses and users start reporting errors, other factors will also become important. From the point of view of contributors to the maintainability will be important as correction of errors or enhancement of the existing code is a significant task and the more maintainable the code is, the easier it will be for the maintainers to make changes. If the software lacks in maintainability, the maintenance team might loose interest and hence result in decline in quality.

For the original code contributors, reusability will become important as the software grows in size and functionality. If the users find the components developed in a project reusable, they will be more likely to continue using the software.

The characteristic of Availability becomes significant when the user has established the use of the software and is running applications that are supported by the software e.g. an OSS operating system. In such a case availability of the system and security aspects will also add towards the overall user perception of quality.

For any software, most of the lifecycle cost and effort is expended in the detection and elimination of errors or for functionality enhancements during maintenance [16, 18]. Addition of new functionality can make the maintenance task more difficult. The addition of new modules is usually accompanied by new errors thus making the maintenance task more complex [19].

We model the effects of corrective, adaptive and preventive maintenance on software quality. We combined perfective and adaptive because changes made to accommodate environmental requirements and user requirements will originate through the same process in OSS.

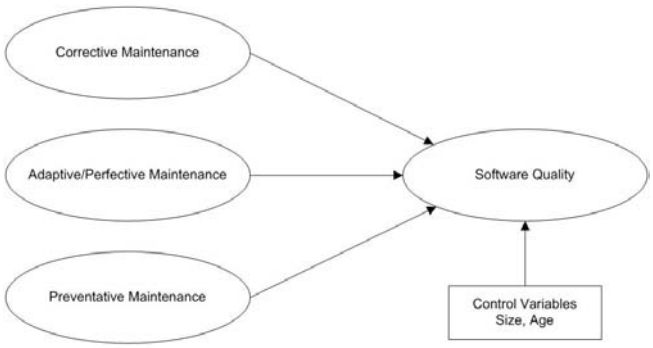

Figure 1: Research Model for OSS Quality

In OSS communities, user of the software detects defects and reports them (user can be a maintainer, author or an end user). Individual projects have their own structure to remove defects. Addition of new code to remove the defect will increase the complexity of the software. The more complex the software is, the harder it will be to make additions to it in future. If the code is not well documented, maintainers will be reluctant to remove existing lines of code. Corrective maintenance will reduce the reliability and maintainability of the software. We expect a negative relationship between corrective maintenance and quality.

Adaptive maintenance can present a complex scenario in case of corporate use of OSS. If OSS is used in organizations which have existing PSS applications, adaptive maintenance will be required to accommodate interfaces to PSS. Perfective maintenance in OSS will be very similar to Adaptive maintenance; therefore we group the two together in our model. The effect of this category of maintenance will be domain dependent. If the operational environment of OSS is very dynamic then there will be a frequent need for such maintenance. If on the other hand the operational environment is stable and the users are not demanding frequent addition of new functionality, there will less need for such maintenance.

Preventative maintenance is proactive approach to maintenance. We believe that OSS projects sustain a high level of quality through significant preventative maintenance. We expect a positive effect of such maintenance on software quality.

5. DATA ANALYSIS AND RESULTS

We are examining the research model using empirical data from our analysis of the Linux source code, as it evolved through version 2.4.0 to 2.4.20, a total of 21 releases. The period of release was 2001 to 2003. We measure the size of individual modules in each release and then use an aggregate measure for the entire release. The total number of modules analyzed increase from 5571 in version 2.4.0 to 11340 in version 2.4.20. Software size is measured in Source Lines of code (SLOC). We measure the raw size from the tar ball of the kernel and the SLOC using Linux commands for each module and for the complete system. We also measure size in terms of number of C modules. All results are

validated and verified against test files. Changes to a Linux version are provided in the form of a new patch. The user can simply download the new patch and the older version is updated to the current version. We analyze the patch files to obtain the same detailed metrics on each patch. We developed a tool to count the number and type of modules being added, deleted or updated in a newer version. We validated our tool against files of known modules and changes. We also developed a tool to count the number of corrective, adaptive or perfective maintenance from the change logs. So far a total of 29580 patches has been analyzed in the 21 releases.

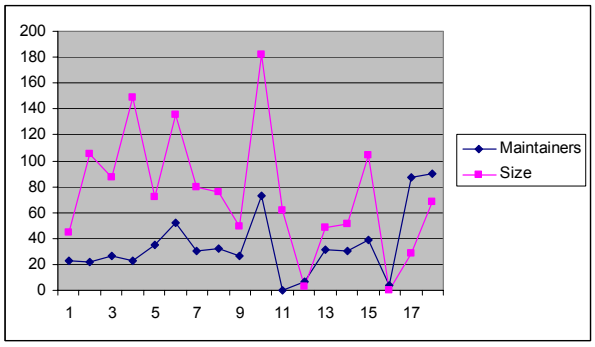

Figure 2: Growth of size and maintainers in Linux

Prior research has shown that Linux exhibits super linear growth [5]. Our analysis indicated that the increase in size is accompanied by a proportional increase in the number of maintainers, as shown in figure 1. Thus the quality is maintained because an increase in size of the software is accompanied by an increase in the effort expended in maintenance.

As explained earlier, we believe that the importance of the quality carrying characteristics will be dynamic in OSS. Since there is no means to capture conformance to user requirements or user satisfaction, for Linux we operationalize software quality in terms of Software Maturity Index (SMI), as defined by IEEE std.982.1-1988. SMI provides an indication of the stability of s software product. As SMI begins to approach 1.0, software product begins to stabilize. Preliminary results on testing of our model will be ready for presentation at the workshop.

6. REFERENCES

[1] M.-W. Wu and Y.-D. Ling, "Open source software development : An overview," *IEEE Computing Practices*, June 2001.

[2] E. S. Raymond, http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/," 2003.

[3] T. O'Reilly, "Lessons from Open-source software development," *Communications of the ACM*, vol. 42, 1999.

[4] N. E. Fenton and S. Pfleeger, *Software Metrics: A Rigorous Approach*. New York: Chapman & Hall, 1991.

[5] M. Godfrey and Q. Tu, "Growth, Evolution and Structural Change in Open Source Software," presented at IWPSE, Vienna Austria, 2001.

[6] D. A. Wheeler, "More than a Gigabuck: Estimating GNU/Linux's size," 2003.

[7] J. Paulson, G. Succi, and A. Eberlein, "An Empirical Study of Open Source and Closed-Source Software Products," *IEEE Transactions of Software Engineering*, vol. 30, April 2004.

[8] T. O'Reilly, "Lessons from open source software developmet," *Communications of the ACM*, vol. 42, pp. 32-37, 1999.

[9] A. Maccormack, "Red Hat and the Linux Revolution," in *Harvard Business School Case*: 9-600-009, 2002.

[10] B. Boehm, "Software Engineering Economics," *IEEE Transactions on Software Engineering*, vol. 10, pp. 4-21, 1984.

[11] C. L. Jones, "A Process-Integrated Approach to Defect Prevention," *IBM Systems Journal*, vol. 24, pp. 150-167, 1985.

[12] L. J. Arthur, *Measuring Programmer Productivity and Software Quality*. New York: Wiley, 1984.

[13] M. M. Lehman and J. F. Ramil, "Rules and Tools for Software Evolution Planning and Management," *Annals of Software Engineering*, vol. 11, pp. 15-44, 2001.

[14] S. G. Eick, T. L. Graves, A. K. Karr, J. S. Marron, and A. Mockus, "Does Code Decay? Assessing he Evidence from Change Management Data," *IEEE Transactions on Software Engineering*, vol. 27, pp. 1-12, 2001.

[15] C. F. Kemerer, "Software Complexity and Software Maintenance," *Annals of Software Engineering*, vol. 1, pp. 1-22, 1995.

[16] M. M. Lehman and J. F. Ramil, "Software Evolution and Software Processes," *Annals of Software Engineering*, vol. 14, pp. 275-309, 2002.

[17] K. H. Bennett, C. Knight, M. Munro, and J. Xu, "Centres of Excellence: Research Institute in Software Evolution, University of Durham," *Computing and Control Engineering Journal*, pp. 179-186, 2000.

[18] E. B. Swanson and E. Dans, "System Life Expectency and the maintanance Effort: Exploring their Equilibrium," *MIS Quarterly*, vol. 24, pp. 277-297, 2000.

[19] F. P. Brooks, *The Mythical Man-Month*: Addison Wesley, 1995.