Quality Practices in Open Source Software Development Affecting Quality Dimensions

Sheikh Umar Farooq[*]
S. M. K. Quadri[**]

Abstract

Purpose: *The quality of open source software has been a matter of debate for a long time now since there is a little concrete evidence to justify it. The main concern is that many quality attributes such as reliability, efficiency, maintainability and security need to be carefully checked, and that fixing software defects pertaining to such quality attributes in OSDM (Open Source Development Model) can never be guaranteed fully. In order to diminish such concerns, we need to look at the practices which affect these quality characteristics in OSS (Open Source Software) negatively. This paper presents an exploratory study of the quality dimensions and quality practices and problems in OSDM. An insight of these problems can serve as a start point for improvements in quality assurance of open source software.*

Design/Methodology/Approach: *A survey was administered based on existing literature. On the basis of this survey those practices in OSDM are described which affect quality attributes negatively in OSS.*

Findings: *The quality characteristics which should be taken into consideration to select or evaluate OSS are presented. Furthermore, quality practices in OSDM which affect the quality of OSS in a negative manner have also been highlighted.*

Research Implications: *Further research is suggested to identify other quality problems not found in this paper and to evaluate the impact of different practices on project quality.*

Originality/Value: *As a first step in the development of practices and processes to assure and further improve quality in open software projects, in addition to quality attributes, existing quality practices and quality problems have to be clearly identified. This paper can serve as a start point for improvements in quality assurance of open source software's.*

Keywords: *Open Source Software; Software Quality; Quality Practices; Quality Problems.*

Paper Type: *Survey Paper*

Introduction

There are more than hundred thousand open source software of varying quality. The OSS model has not only led to the creation of significant software, but many of these software show levels of quality comparable to or exceeding that of software developed in a closed and proprietary manner **(Halloran & Scherlis, 2002; Schmidt & Porter, 2001).** However, open source software also face certain

[*] Research Scholar. P. G. Department of Computer Sciences, University of Kashmir, Jammu and Kashmir. 190 006. India. email: shiekh.umar.farooq@gmail.com

[**] Head. P. G. Department of Computer Sciences, University of Kashmir, Jammu and Kashmir. 190 006. India. E-Mail: quadrismk@hotmail.com

challenges that are unique to this model. For example, due to the voluntary nature of open source software projects, it is impossible to fully rely on project participants **(Michlmayr & Hill, 2003)**. This issue is further complicated by the distributed nature because it makes it difficult to identify volunteers who are neglecting their duties, and to decide where more resources are needed **(Michlmayr, 2004)**. While most research on open source has focused and hyped popular and successful projects such as Apache **(Mockus, Fielding & Herbsleb, 2002)** and GNOME **(Koch & Schneider, 2002)**, there is an increasing awareness that not all open source software projects are of high quality. *SourceForge*, which is currently the most popular hosting site for free software and open source projects with over 95,000 projects, is not only a good resource to find well maintained free software applications – there are also a large number of abandoned projects and software with low quality **(Howison & Crowston, 2004)**. Some of these low quality and abandoned projects may be explained in terms of a selection process given that more interesting projects with a higher potential will probably attract a larger number of volunteers, but it has also been suggested that project failures might be related to the lack of project management skills **(Senyard & Michlmayr, 2004)**. Nevertheless, large and successful projects also face important problems related to quality **(Michlmayr & Hill, 2003; Michlmayr, 2004; Villa, 2003)**. In order to ensure that open source software remains a feasible model for the creation of mature and high quality software suitable for corporate and mission-critical use, open software quality assurance has to take these challenges and other quality problems into account and find solutions to them. As a first step in the development of practices and processes to assure and further improve quality in open software projects, existing quality practices and quality problems have to be clearly identified. To date however, only a few surveys on quality related activities in open software projects *(that too mostly in successful OSS)* have been conducted **(Zhao & Elbaum, 2000; Zhao, 2003)**. This paper presents an exploratory study of the quality dimensions and quality practices and problems in open source software based on existing literature.

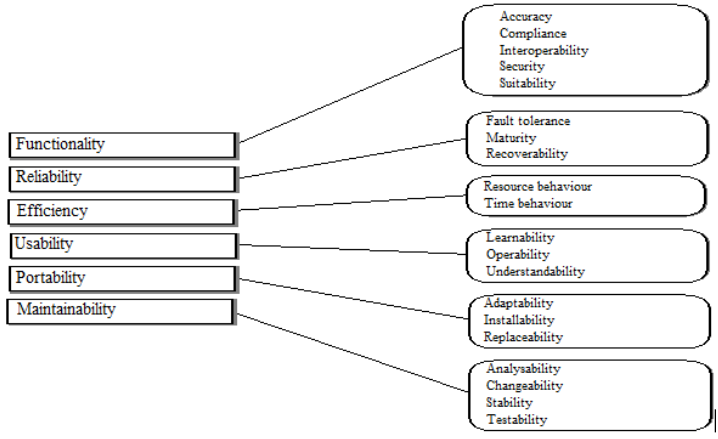
I Software Quality and its Characteristics

Software quality is imperative for the success of a software project. **Boehm (1984)** defines software quality as "achieving high levels of user satisfaction, portability, maintainability, robustness and fitness for use". **Jones (1985)** refers to quality as "the absence of defects that would make software either stop completely or produces unacceptable results". These definitions of software quality cannot be applied directly to OSS. Unlike CSS, user requirements are not formally available in OSS. We can

evaluate the project and its program on a number of important attributes. Important attributes include functionality, reliability, usability, efficiency, maintainability, and portability. The benefits, drawbacks, and risks of using a program can be determined from examining these attributes. The attributes are same as with proprietary software, of course, but the way we should evaluate them with OSS is often different. In particular, because the project and code is completely exposed to the world, we can (and should!) take advantage of this information during evaluation. 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.

Existing quality models provide a list of quality carrying characteristics that are responsible for high quality (or otherwise) of software. Software quality is an abstract concept that is perceived and interpreted differently based on one's personal views and interests. To dissolve this ambiguity, ISO/IEC 9126 provides a framework for the evaluation of software quality. ISO/IEC 9126 is the standard of quality model to evaluate a single piece of software **(Software Engineering-Product Quality-Part 1, 2001; Software Engineering-Product Quality-Part 2, 2001)**. ISO/IEC 9126 defines *six software quality attributes*, often referred to as quality characteristics along with various sub-characteristics to evaluate the quality of software as shown in **Fig. 1**.

Fig. 1: ISO 9126 Software Quality Model



➢ **Functionality**

Functionality refers to the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions. Functionality means the number of functions must be available in software that fulfils the minimum usage criteria of the user **(Raja & Barry, 2005)**. ISO 9126 Model describe the functionality attribute as "a set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs". This set of attributes characterizes what the software does to fulfil needs, whereas the other sets mainly characterizes when and how it does so **(International Organization for Standardization, 1991)**. It is fundamental characteristic of the software development and it is close to the property of the correctness **(Fenton, 1993)**. The specific functions that we need obviously depend on the kind of program and our specific needs. However, there are also some general functional issues that apply to all programs. In particular, we should consider how well it integrates and is compatible with existing components we have. If there are relevant standards, does the program support them? If we exchange data with others using them, how well does it do so? For example, *MOXIE*: Microsoft Office – Linux Interoperability Experiment downloaded a set of representative files in Microsoft Office format, and then compare how well different programs handle them **(Venkatesh et al, 2011)**. For Type-1 OSS there are no formal functionality requirements, yet there will be a certain level of expectation in terms of its functionality compared to an existing CSS. Type-1 OSS will be considered of a high quality and 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. The sub characteristics of functionality attribute specified by **Punter, Solingen & Trienekens (1997)** are as:

- **Accuracy**

This refers to the correctness of the functions i.e. to provide the right or agreed results or effects with the needed degree of precision. e.g. an ATM may provide a cash dispensing function but is the amount correct?

- **Compliance**

Where appropriate certain industry (or government) laws and guidelines need to be complied with, i.e. SOX. This sub characteristic addresses the compliant capability of software.

- **Interoperability**

A given software component or system does not typically function in isolation. This sub characteristic concerns the ability of a software component to interact with other components or systems.

- **Security**

This sub characteristic relates to unauthorized access to the software functions (programs)/data.

- **Suitability**

This characteristic refers to the appropriateness (to specification) of the functions of the software.

➢ **Reliability**

Reliability refers to the capability of the software product to maintain its level of performance under stated conditions for a stated period of time. 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 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. Many of the open source projects do not have resources to dedicate to accurate testing or inspection so that the reliability of their products must rely on community's reports of failures. The reports stored in the so-called bug tracking systems, are uploaded by the community, and moderated by internal members of the open source project. Reports are archived with various pieces of information including the date of upload and the description regarding the failure. What information can be collected from these repositories and how to mine them for reliability analysis is still an open issue **(Li, Herbsleb & Shaw, 2005; Godfrey & Whitehead, 2009)**. Problem reports are not necessarily a sign of poor reliability - people often complain about highly reliable programs, because their high reliability often leads both customers and engineers to extremely high expectations. Indeed, the best way to measure reliability is to try it on a "real" work load. Reliability has a significant effect on software quality, since the user acceptability of a product depends upon its ability to function correctly and reliably **(Samoladas & Stamelos, n.d)**. ISO 9126 defines reliability as “a set of attributes that bear on the capability of software to maintain its performance level under stated conditions for a stated period of time” **(International Organization for Standardization, 1991)**. Further, sub characteristics of reliability attribute stated by **Punter, Solingen & Trienekens (1997)** are as:

- **Fault Tolerance**

The ability of software to withstand and maintain a specified level of performance in case of software failure.

- **Maturity**

The Capability of the software product to avoid failures, as a result of faults in the software. It is refined into an attribute Mean Time to Failure (MTTF).

- **Recoverability**

Ability to bring back a failed system to full operation, including data and network connections.

- **Efficiency**

Efficiency refers to the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions. According to the ISO Model, efficiency is "a set of attributes that bear on the relationship between the software's performance and the amount of resources used under stated conditions" ***(International Organization for Standardization, 1991).*** *Efficiency describes that the response of the software should be faster in the form of any input. The sub characteristics of efficiency attribute are as* ***(Punter, Solingen & Trienekens, 1997)****:*

- **Resource Behavior**

Amount and type of resources used and the duration of such use in performing its function. It involves the attribute complexity that is computed by a metric involving size (space for the resources used and time spent using the resources).

- **Time Behavior**

The capability of the software product to provide appropriate response time, processing time and throughput rates when performing its function under stated conditions. It is an attribute that can be measured for each functionality of the system.

➢ **Usability**

Usability refers to the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions (the effort needed for use). ISO 9126 describe the usability attribute as "a set of attributes that bear on the effort needed for use and on the individual assessment of such use by a stated or implied set of users" **(International Organization for Standardization, 1991)**. The usability of open source software is often regarded as one reason for this limited distribution. The usability problem in most OSS is because of the following reasons:

- Developers are not users so they usually do not take user perception into consideration.
- Usability experts do not get involved in OSS projects
- The incentives in OSS work better for improvement of functionality than usability
- Usability problems are harder to specify and distribute than functionality problems
- Design for usability really ought to take place in advance of any coding

- Open source projects lack the resources to undertake high quality usability work
- OSS development is inclined to promote power over simplicity

It's important to note that to improve usability many OSS programs are intentionally designed into at least two parts: an *"engine"* that does the work and a *"GUI"* that lets users control the engine through a familiar point and click interface *(fragmentation)*. This division into two parts is considered an excellent design approach; it generally improves reliability, and generally makes it easier to enhance one part. Sometimes these parts are even divided into separate projects: The "engine" creators may provide a simple command line interface, but most users are supposed to use one of the available GUIs available from another project. Thus, it can be misleading if you are looking at an OSS project that only creates the engine - be sure to include the project that manages the GUI, if that happens to be a separate sister project. In many cases an OSS user interface is implemented using a web browser. This actually has a number of advantages: usually the user can use nearly any operating system or web browser, users don't need to spend time installing the application, and users will already be familiar with how their web browser works (simplifying training). However, web interfaces can be good or bad, so it's still necessary to evaluate the interface's usability. The sub characteristics of the usability attribute are as **(Punter, Solingen & Trienekens, 1997)**:

- **Learn ability**

Learning effort for different users, i.e. novice, expert, casual etc.

- **Operability**

Ability of the software to be easily operated by a given user in a given environment.

- **Understandability**

Determines the ease of which the systems functions can be understood, relates to user mental models in Human Computer Interaction methods.

- **Portability**

Portability refers to the capability of the software product to be transferred from one environment to another. The environment may include organizational, hardware or software environment. ISO 9126 Model defines the portability attribute as "A set of attributes that bear on the ability of software to be transferred from one environment to another (including the organizational, hardware, or software environment)" **(International Organization for Standardization, 1991).** Portability is also a main issue of today and with respect to it, Open Source Software could run and give better results on different platforms **(Loannis & Stamelos, 2011)**. From its early days, portability has been a central issue in OSS development. Various OSS systems have as first

priority the ability of their software to be used on platforms with different architectures. Here, we have to stress on important fact, which originates from the nature of OSS, and helps portability, namely the availability of the source code of the destination software. If the source code is available, then it is possible for the potential developer to port an existing OSS application to a different platform than the one it was originally designed for. Perhaps the most famous OSS, the Linux kernel, has been ported to various CPU architectures other than its original one, the x86. In the end, evaluating usability requires hands-on testing. The sub characteristics of portability attribute are as **(Punter, Solingen & Trienekens, 1997)**:

- **Adaptability**

Characterizes the ability of the system to change to new specifications or operating environments.

- **Install ability**

Characterizes the effort required to install the software in a specified environment.

- **Replaceability**

The capability of the software product to be used in place of another specified software product for the same purpose in the same environment.

➢ **Maintainability**

Maintainability refers to the capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the software to changes in the environment and in the requirements and functional specifications (the effort needed to be modified).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. ISO 9126 defines it as "A set of attributes that bear on the effort needed to make specified modifications (which may include corrections, improvements, or adoptions of software to environmental changes and changes in the requirements and functional specifications)" **(International Organization for Standardization, 1991)**. Maintainability of OSS projects is a factor that was one of the first to be investigated by the OSS literature. This was done mainly because OSS development emphasizes on the maintainability of the software released. Making software source code available over the Internet allows developers from all over the world to contribute code, adding new functionality (parallel development) or improving present one and

submitting bug fixes to the present release (parallel debugging). A part of these contributions are incorporated into the next release and the loop of release, code submission/bug fixing, incorporation of the submitted code into the current and new release is continued. This circular manner of OSS development implies essentially a series of frequent maintenance efforts for debugging existing functionality and adding new one to the system. These two forms of maintenance are known as corrective and perfective maintenance respectively.

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 within 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. The sub characteristics of the maintainability are as **(Punter, Solingen & Trienekens, 1997):**

- **Changeability**

It refers to the capability of the software product to enable a specified modification to be implemented. It also characterizes the amount of effort to change a system.

- **Stability**

The capability of the software product to avoid unexpected effects from modifications of the software (the risk of unexpected effect of modifications)

- **Testability**

Characterizes the effort needed to verify (test) a system change.

- **Analyzability**

It characterizes the ability to identify the root cause of a failure within the software.

Different users have different expectations of the same software and user's expectations of software evolve with time. For instance, some users may view performance and reliability as the key features of software, while others may consider ease of installation and maintenance as key features of the same software. Therefore, software applications today must do more than just meet technical specifications; they must be flexible enough to meet the varying needs of a diverse user base and provide reasonable expectations of future enhancements. The last five characteristics are not related to the task performed by the software and therefore are regarded as non-functional attributes. In many cases though software requirements and testing methodologies are mostly

focused on functionality and pay little if any attention to non-functional requirements. Since nonfunctional requirements affect the perceived quality of software (quality in use), failure to meet them often leads to late changes and increased costs in the development process. For example Reliability is a non-functional requirement that needs to be addressed in every software project. Therefore badly-written software may be functional, but not a reliable one.

II Quality Problems under Open Source Model

Although many high profile cases of successful OSSD projects exist (e.g., Apache, OpenOffice, PHP), the harsh reality is that the majority of OSS projects are of low quality. No doubt open source practices have been remarkable success as can be seen in some successful OSS, we believe there are several areas where there are opportunities for improvement. A commonly cited reason for the failure of OSS projects to reach a maturity level is in coordination of developers and project management, leading to some duplication of efforts by multiple developers, inefficient allocation of time and resources, and lack of attention to software attributes such as ease of use, documentation, and support, all of which impact conformance to specifications. Only few projects have explicit documentation describing ways of contributing to and joining a project. One more critical problem due to voluntary nature of open source is that, reliance on project participants can never be guaranteed **(Michlmayr & Hill, 2003)**. Regarding to its distributed nature issues like to identify who gets what to be done or to decide where more resources to break bottleneck need to be examined **(Michlmayr, 2004)**. Following issues usually lead to low quality software's under OSDM:

- **Missing or Incomplete Documentation**

Documentation is necessary for every project. Programmers and users have always criticized projects which lacks documentation regarding development practices **(Michlmayr, Hunt & Probert, 2005)**. A study in QA reveals that over 84% of the respondents prepare a ''TODO'' list including list of pending features and open bugs. 62% build installation and building guidelines, 32% projects have design documents, and 20% have documents to plan releases including date and content **(Zhao, 2003)**. Most of the open source projects / software's have little or no documentation. However, some projects with a large number of contributors have good documentation about coding styles and code commit **(Michlmayr, Hunt & Probert, 2005)**. Lack of documentation reduces the motivation of new users and programmers, because they always confront the difficulty to understand the project, whatever in order to make usage or improvement. New developers, who would like to

participate into a project potentially, have to understand a part of the project well enough **(Ankolekar, Herbsleb & Sycara, 2003)**. Volunteers may like to contribute in an area but they might not know how to start and where to start without proper documentation. The lack of developer documentation also implies that there is no assurance that everyone follows the same techniques and procedures. At the very beginning of Mozilla project, the community has faced problem to attract new developers, the situation did slow down the proceeding of project. After more well-formed documentations and tutorials were provided, the number of participants significantly raised **(Mockus, Fielding & Herbsleb, 2002)**. Due to the nature of the open source less attraction to users and developers may leads to low quality product or even abend of project **(Zhao, 2003)**. A survey, which explored QA activities in open source, concluded in that OS project starts regularly without a planning **(Zhao & Elbaum, 2000)**. While there is no specific definition of program, the program varies regularly during the development process. Worse off, those changes are most poorly recorded in documentation. Undocumented planning and program changes make the measure and validation of end product impossible.

- **Problems in Collaboration**

Software development is an interactive behaviour, often with tight integration and interdependencies between modules, and therefore requires a substantial amount of coordination and communication between developers if they are to collaborate on features **(Ankolekar, Herbsleb & Sycara, 2003)**. Strong user involvement and participation throughout a project is a central view of OSSD. In some projects, there are problems with coordination and communication which can have a negative impact on project quality. It is more difficult to achieve coordination and agreeing to goals in OSS development than in closed source software development. Sometimes it is not clear who is responsible for a particular area and therefore things cannot be communicated properly. There may also be duplication of effort and a lack of coordination related to the removal of critical bugs. Some features may for example be duplicated under open-source development because there is some chance that developers with the same needs will not meet – or will not agree on their objectives and methods when they meet and will end-up developing the same types of features independently (forking). In traditional development team, developers can work effective together, as long as the team members understand with each other. Due to convenient communications possibility those team tends to advance efficiently **(Thayer & McGetrick, 1993)**.Since the team members may cooperate on module or single one feature, to be aware of the activities

of cooperating members is important **(Ankolekar, Herbsleb & Sycara, 2003)**. Individuals and small teams take the advantages of convenient communication and simpler decision method. In any case, the potential for collaborative and group maintenance in successfully resolving a serious quality assurance issue is obvious and its importance and prominence in successful projects, in one form or another, seems like a good possibility **(Michlmayr & Hill, 2003)**.

- **Lack of global view of system constraints**

Large-scale open-source projects often have a large number of contributors from the user community (i.e., the periphery). When these users encounter problems, they may examine the source code, propose/apply fixes locally, and then submit the results back to the core team for possible integration into the source base. Often these users in the periphery have much less knowledge of the entire architecture of an open-source software system than the core developers. As a result, they may lack a global view of broader system constraints that can be affected by any given change, so their suggested fixed may be inappropriate.

- **Dependence on Participants**

No participants in OSS can be held responsible; the strong reliance on individual developers comes to be a consideration of quality assurance. It's a conflict that a project expects predictability and reliability from participants, who claims to be irresponsible for the project **(Raymond, 1999)**. A large user group is usually the fundamental of open source project **(Zhao, 2003)**. Without new volunteers the project seemed hard to proceed, because when project begins, it also starts losing participants. No member is obligated to contribute until the end of project **(Raymond, 1999)**, developers are free to decide, if stay with project or just leave. For open source project, regular demand on new developers keep itself proceeding steadily. A problem some projects face, especially those that are not very popular, is attracting volunteers. A study has confirmed that unlike big and mature projects, small projects may not receive much feedback from developers and co-users **(Mockus, Fielding & Herbsleb, 2002)**. There are usually many ways of contributing to a project, such as coding, testing or triaging bugs. However, many projects only find prospective members who are interested in developing new source code. As a result, developers have to use a large portion of their time for tasks other people could easily handle. Few contributors are interested in helping with testing, documentation and other activities. These are vital activities, particularly as projects mature and need to be maintained and updated by new cohorts of developers. Good documentation, tutorials, development tools, and a reward and recognition culture facilitate the creation of a sustainable community.

- **Unsupported Code**

One of the unsolved problems is how to handle code that has previously been contributed but which is now unmaintained. A contributor might submit source code to implement a specific feature or a port to obscure hardware architecture. As changes are made by other developers, this particular feature or port has to be updated so that it will continue to work. Unfortunately, some of the original contributors may disappear and the code is left unmaintained and unsupported. Lead developers face the difficult decision of how to handle this situation.

- **Release Problems**

Release management is one of the most important controller to ensure the quality of open source software. The state of release management guidelines remains remarkably informal since the beginning of open source development **(Erenkrantz, 2003).** Carefully defined criteria are needed to regulate the release management. Oftentimes, release manager are adopted in decentralized open source model to fit the rapidly scaled project dimensions **(Zhao, 2003)**. Under open source, it's recommended to release often and release early **(Raymond, 1999)**. The argument behind this principle is that, users will take the responsibility to find the bugs. It has been confirmed that a good part of debugging tasks are shifted to users **(Zhao, 2003)**. But as new versions are frequently released with poorly tested by core team, users burden the most tasks of debugging. The activities of testing increase, the quality of program gets worse **(Hendrickson, 2001)**. Though software quality investments can reduce overall software cycle costs by minimizing rework later on, many software manufacturers sacrifice quality in favor of other objectives such as shorter development cycles and meeting time constraints. As one of the manager said, "I would rather have it wrong than have it late" **(Paulk, Weber, Curtis & Chrissis, 1994)**. In contrast traditional conception of software quality is centred on a product-centric, conformance view of quality **(Prahalad & Krishnan, 1999)**. Absence of static testing on developer side delivers much more bugs as usually can be caught by a number of users. Often it turns out to be impossible for developers to keep up with a mass of bug reports. Release may be frequently performed, when every claimed stable version fulfills the settled release qualifications. Otherwise, it must be labeled as unstable version. It can be hard, however, to ensure consistent quality of open-source software due to the short feedback loops between users and core developers, which typically result in frequent “beta” releases, e.g., several times a month. Although this schedule satisfies end-users who want quick patches for bugs they found in earlier betas, it can be frustrating to other end-users who want more stable, less frequent software releases. In addition to our

own experiences, Gamma describes how the length of the release cycles in the Eclipse frame-work affected user participation and eventually the quality of the software **(Gamma, 2005)**.

- **Version Authorization**

The many different commercial versions of Linux already pose a substantial problem for software providers developing for the Linux platform, as they have to write and test applications developed for these various versions. The availability of source code often encourages an increase in the number of options for configuring and sub setting the software at compile and runtime. Although this flexibility enhances the software's applicability for a broad range of use cases, it can also exacerbate QA costs due to a combinatory increase in the QA space. Moreover, since open-source projects often run on a limited QA budget due to their minimal/non-existent licensing fees, it can be hard for core developers to validate and support large numbers of versions and variants simultaneously, particularly when regression tests and benchmarks are written and run manually. Smith reports an exchange with an IT manager in a large Silicon Valley firm who lamented, "Right now, developing Linux software is a nightmare, because of testing and QA—how can you test for 30 different versions of Linux?" **(Feller, et al, 2005)**.

- **Testing and Bug Reporting**

The study of 200 OSS projects discovered that

- fewer than 20 percent of OSS developers use test plans;
- only 40 percent of projects use testing tools, although this increases when testing tool support is widely available for a language, such as Java; and
- less than 50 percent of OSS systems use code coverage concepts or tools.
- Larger projects do not spend more time in testing than smaller projects.

OSS development clearly doesn't follow structured testing methods. The methodology an OSS project adopts will depend largely on the available expertise, resources, and sponsorship. Formal testing techniques and test automation are expensive and require sponsorship. Some high-profile open source projects can achieve this, but most don't, so the user base is often the only choice **(Aberdour, 2007)**. As more users with few technical skills use free software, developers see an increase in useless or incomplete bug reports. In many cases, users do not include enough information in a bug report or they file duplicate bug reports. Such reports take unnecessary time away from actual development work. Some projects have tried to write better documentation about reporting

bugs but they found that users often do not read the instructions before reporting a bug. Many popular open-source projects (such as GNU GCC, CPAN, Mozilla, the Visualization Toolkit, and ACE+TAO) distribute regression test suites that end users can run to evaluate the success of an installation on a user's platform. Users can – but frequently do not – return the test results to project developers. Even when results are returned to core developers, however, the testing process is often undocumented and unsystematic, e.g., core developers have no record of what configurations were tested, how they was tested, or what the results were, which loses crucial QA-related information. Moreover, many QA configurations are executed redundantly by thousands of users (e.g., on popular versions of Linux or Windows), whereas others are never executed at all (e.g., on less widely used operating systems).

- **Configuration Management**

Many free software and open source projects offer a high level of customization. While this gives users much flexibility, it also creates testing problems. It is very difficult or impossible for the lead developer to test all combinations so only the most popular configurations tend to be tested. It is quite common that, when a new release is made, users report that the new version broke their configuration. Well-written open-source software (e.g., based on GNU autoconf) can be ported easily to a variety of OS and compiler platforms. In addition, since the source is avail-able, end-users can modify and adapt their source base readily to fix bugs quickly or to respond to new market opportunities with greater agility. Support for platform-independence, however, can yield the daunting task of keeping an open-source source software base operational despite continuous changes to the underlying platforms. In particular, since developers in the core may only have access to a limited number of OS/compiler configurations, they may release code that has not been tested thoroughly on all platform configurations on which users want to run the software.

Although in some cases OSS seems to do better than closed source software, there are many things that need to be to be improved and further expanded, so that we avoid typical problems that arise from practices usually employed in OSS. To achieve the maturity level and to produce high quality open source software's one should also employ proved practices and methods usually employed in closed source software development in beneficial manner. **Aberdour (2007)** compares quality management practices in open source and closed source software development as shown in **Table 1**. We should strive to employ these proven practices in all types of projects whether small or large to achieve high quality and matured Open Source Software.

Table 1: Quality Management in Open Source & closed Source

Closed Source	Open Source
Well-defined developed methodology	Development methodology often not defined or documented
Extensive project documentation	Little project documentation
Formal, structured testing and quality assurance methodology	Unstructured and informal testing and quality assurance methodology
Analysts define requirements	Programmer define requirements
Formal Risk assessment process – monitored and managed throughout project	No formal risk assessment process
Measurable goals used throughout project	Few measurable goals
Defect discovery from black-box testing as early as possible	Defect discovery from black-box testing late in the process
Empirical evidence regarding quality routinely to aid decision making	Empirical evidence regarding quality isn't collected
Team members are assigned work	Team members choose work
Formal design phase is carried out and signed off before programming starts	Projects often go straight to programming
Much effort put into project planning and scheduling	Little project planning or scheduling

Conclusion and Future Work

OSS quality is an open issue and it should continue striving for even better quality levels if it has to outperform traditional, closed source development and target corporate and safety critical systems. The quality of selected software and the standards of evaluating the quality of OSS are often wrongly defined. Therefore, in this paper the quality characteristics which should be taken into consideration to select or evaluate OSS are also presented. The paper also presents insights into quality practices of open source software projects which affects the quality of OSS in a negative manner. Avoiding such practices and using proven quality management practices can result in high quality OSS. Further research is suggested to identify other quality problems not found in this paper and to evaluate the impact of different practices on project quality.

References

Aberdour, M. (2007). Achieving Quality in Open Source Software. *IEEE Computer Society*. 24 (1), 58-64. doi: 10.1109/MS.2007.2

Ankolekar, A., Herbsleb, J.D., & Sycara, K. (2003). Addressing Challenges to Open Source Collaboration with the Semantic Web. Retrieved from http://www.cs.cmu.edu/~anupriya/papers/icse2003.pdf

Boehm, B. W. (1984). Software Engineering Economics. *IEEE Transactions on Software Engineering*. 10 (1), 4-21. doi: 10.1109/TSE.1984.5010193

Erenkrantz, J.R. (2003). Release Management within Open Source Projects. Retrieved from *http://www.erenkrantz.com/Geeks/Research/Publications/ReleaseManagement.pdf*

Feller, J., et al (Eds.)(2005). *Perspectives on free and open source software*. Cambridge, Mass: MIT.

Fenton, N. E. (1993). *Software Metrics: A Rigorous Approach*. London: Chapman and Hall.

Gamma, E. (2005). Agile, open source, distributed, and on-time: inside the eclipse development process. Retrieved from http://www.inf.fu-berlin.de/inst/ag-se/teaching/S-BSE/034_Eclipse-process.pdf

Godfrey, M. W., & Whitehead, J. (2009). Proceedings of the 2009 *6th IEEE International Working Conference on Mining Software Repositories*, Vancouver Canada, May 16-17.

Halloran, T. J., & Scherlis, W. L. (2002). High quality and open source software practices. Retrieved from http://flosshub.org/system/files/HalloranScherlis.pdf

Hendrickson, E. (2001). Better Testing – Worse Quality? In *International Conference on software Management & Applications of Software Measurement,* February 12-16, 2001 San Diego, CA, USA

Howison, J., & Crowston, K. (2004). The perils and pitfalls of mining SourceForge. Retrieved from http://msr.uwaterloo.ca/papers/Howison.pdf

International Organization for Standardization. (1991). Information technology-Software product evaluation: Quality characteristics and guidelines for their use. Berlin: Beuth-Verlag: ISO/IEC.

Jones, C. L. (1985). A Process-Integrated Approach to Defect Prevention. *IBM Systems Journal*. 24 (2), 150-167. doi:10.1147/sj.242.0150

Koch, S., & Schneider, G. (2002). Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal*. 12 (1), 27–42. doi: 10.1046/j.1365-2575.2002.00110.x

Li, P.L., Herbsleb, J., & Shaw, M. (2005). Forecasting field defect rates using a combined time-based and metrics-based approach: a case study of OpenBSD. *16th IEEE International Symposium on Software Reliability Engineering (ISSRE) (pp. 193-202).* Washington, DC, USA: IEEE Computer Society. doi: 10.1109/ISSRE.2005.19

Michlmayr, M. (2004). Managing volunteer activity in free software projects. *In Proceedings of the 2004 USENIX Annual Technical Conference (pp. 39-33), FREENIX Track*, Boston, MA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=1247415.1247454

Michlmayr, M., & Hill, B. M. (2003). Quality and the reliance on individuals in free software projects. *In Proceedings of the 3rd Workshop on Open Source Software Engineering (pp. 105–109)*. Portland, OR, USA: ICSE.

Michlmayr, M., Hunt, F., & Probert, D. (2005). Quality Practices and Problems in Free Software Projects. *In Proceedings of the First International Conference on Open Source Systems Geneva, 11th-15th July (pp. 24-28)*

Mockus, A. R., Fielding, T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*. 11 (3), 309–346. doi:10.1145/567793.567795

Paulk, M. C, Weber, C., Curtis, W., & Chrissis, M. (1994). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, Mass: Addison-Wesley.

Prahalad, C. K., & Krishnan, M. S. (1999 September). The New Meaning of Quality in the Information Age. *Harvard Business Review*. 77 (5), 109-118. Retrieved from http://hbr.org/1999/09/the-new-meaning-of-quality-in-the-information-age/ar/1

Punter, T., Solingen,R.V., & Trienekens, J. (1997). Software Product Evaluation. *4th Conference on Evaluation of Information Technology (30-31 Oct. 1997)*. MB Eindhoven Netherland.

Raja, U., & Barry, E. (2005). *Investing Quality in Large –Scale Open Source Software*. U.S.A: Texas A&M University.

Raymond, E. S. (1999). The Cathedral and the Bazaar. Sebastopol, CA: O'Reilly & Associates.

Samoladas, I., & Stamelos, I. (n.d). *Assessing Free/Open Source Software Quality*. Retrieved from http://ifipwg213.org/system/files/samoladasstamelos.pdf

Schmidt, D. C., & Porter, A. (2001). Leveraging open-source communities to improve the quality & performance of open-source software. *In Proceedings of the 1st Workshop on Open Source Software Engineering*. Toronto, Canada: ICSE.

Senyard, A., & Michlmayr, M. (2004). How to have a successful free software project. *In Proceedings of the 11th Asia-Pacific Software Engineering Conference (pp. 84-91)*. Busan, Korea: IEEE Computer Society.

Software Engineering-Product Quality-Part 1. (2001, June). Software Engineering-Product Quality-Part 1: Quality Model. *ISO/IEC 9126-1*.

Software Engineering-Product Quality-Part 1. (2001, June). Software Engineering-Product Quality-Part 1: Quality Model. *ISO/IEC 9126-2*.

Thayer, R.H., & McGettrick, A.D. (1993). (Eds.), Software Engineering: A European Perspective. *IEEE Computer Society Press*, Los Alamitos, CA.

Venkatesh, C., et al. (2011). Quality Prediction of Open Source Software for e-Governance Project. Retrieved from www.csi-sigegove.org/emerging_pdf/16_142-151.pdf

Villa, L. (2003). Large free software projects and Bugzilla. *In Proceedings of the Linux Symposium (July 23-26, 2003)* Ottawa, Canada, *pp. 447-456.*

Zhao, L. (2003). Quality assurance under the open source development model. *Journal of Systems and Software.* 66 (1), 65–75. doi:10.1016/S0164-1212(02)00064-X

Zhao, L., & Elbaum, S. (2000). A survey on quality related activities in open source. *SIGSOFT Software Engineering Notes*, 25 (3), 54–57. doi: 10.1145/505863.505878