Electronic Communications of the EASST
Volume 20 (2009)

Proceedings of the Third International Workshop on Foundations and Techniques for Open Source Software Certification (OpenCert 2009)

Towards a metric for Open Source Software Quality

Siraj A. Shaikh and Antonio Cerone

11 pages

Guest Editors: Luis Barbosa, Antonio Cerone, Siraj Shaikh
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Towards a metric for Open Source Software Quality

Siraj A. Shaikh[1] **and Antonio Cerone**[2]

[1] s.shaikh@coventry.ac.uk
Department of Computing and the Digital Environment
Faculty of Engineering and Computing, Coventry University, Coventry, UK

[2] antonio@iist.unu.edu, http://www.iist.unu.edu/~antonio
International Institute for Software Technology
United Nations University, Macau SAR China

Abstract: Software quality is more than just conformance to a set of requirements and represents many attributes related to each other that make up a piece of software. An important part of this measure depends on the underlying processes and methodologies used in the engineering of software. We present an early exposition towards a quality model for open source software (OSS). We describe some basic notions of quality for OSS and present a basic model, where quality notions consist of various factors that influence such quality. The purpose of this effort is ultimately to develop a quantitative metric for software quality.

Keywords: Open source; software quality; quality model; software development

1 Introduction

Open source software (OSS) development has proved to be successful over the years and has delivered some high-quality system and application software. This includes some popular operating systems, network services and high-end applications. Linux and BSD distributions, Apache and MySQL serve to be shining examples, and are a testimony to the success and resilience of open source developers' effort. A recent report commissioned by the United States Department of Homeland Security into open source software finds that defect density has fallen by 16% over the last two years to the extent that such software on average contains one defect for every 4000 lines of code [Cov08]. The reputation that such products have built up over the years gives the industry leaders confidence that the open source community could deliver much more. They are therefore increasingly looking to the open source community for critical software for use in critical domains. Spiralling costs of traditional development, innovative developmental approaches, effective user testing and feedback, and better project durability and sustainability are other motivating factors for the industry leaders to look at open source development.

Systems designed for privacy and access control, real-time scheduling and industrial process control in industries such as public transport, automobile, avionics, health and defence undergo thorough modelling and design where each and every functional and operational property receives rigorous treatment. A standard approach to reduce the risks in deploying such critical systems is to establish an independent certification process. Nowadays, this is increasingly enforced by laws, on the basis of international standards which define the quality of software in

terms of the level of technical rigour used in the design, developing, verification, validation and documentation processes as well as the human processes involving project management, training and expertise of project teams, and the users community [Muf06]. However, today we lack standards and methods to assess the quality of open source software. In fact, the lack of central management in open source software projects [MHP05, Mic05] makes it difficult to define a standard that could suggest indicators of the technical rigour used by a distributed community of volunteers and identify the human processes involved in the project.

While open source software has proved to be popular and is often found to be of *better quality* than propertiary software, it is still not clear what factors contribute, either directly or by proxy, to this quality. For use in critical domains aforementioned, factors such as completeness, consistency, maintanability, testability and reliability become necessary more than ever. Moreover, a clearer understanding of the underlying factors is important if appropriate standards and methods to assess quality are to be developed. A lack of central management in most open source software projects [Mic05, MHP05] makes it particularly pertinent to examine available indicators of technical rigour used by a distributed community of volunteers and identify human processes involved in a typical project.

The development of software is very often as much a human process as a structured technical process [Muf06]. Software developers and engineers organise their code using various means, including programming styles and language-specific conventions, and software design and documentation. While a computer may not acknowledge the subjective notions of a well-written code or software design, such factors contribute to software quality, which is more than just conformance to a set of requirements or expectations. Such a notion represents many attributes, some constant some variable, related to each other and often entangled that come together to make a piece of software. An important part of this measure also depends on the underlying processes and methodologies used in the engineering of software. Programming styles, language-specific conventions, software design and documentation are among the factors that directly contribute towards the development of quality attributes of software but their precise role is not always understood or appreciated. The role of such factors then becomes even more blurred when we consider that the software is open sourced.

The rest of this paper is organised as follows. Section 2 introduces different quality metrics to represent the various types of factors that contribute to software quality. Section 3 shows how such factors are related to each other. Section 4 discusses related work and Section 5 concludes the paper.

2 Software Quality Metrics

Software quality metrics are measures of the quality of software based on a number of factors such as software correctness and recognised and industry-standard attributes. Example of such attributes are the ones suggested by the ISO 9126 model: functionality, reliability, usability, maintainability, portability and efficiency, which are assessed and judged through acceptable means. Software correctness is an important factor, which is aimed to be directly achievable by formal design and construction using refinement techniques, or verified using formal methods, along with validation using manual and automated testing.

When dealing with OSS, however, some factors that are unique to the OSS development process must be included within the characterisation of such metrics. In particular OSS has a wider and unrestricted availability to an open community of peer developers, code reviewers, contributors and end-users, who freely engage and employ a diverse range of development practices and methodologies. These factors are unique to OSS and come together to facilitate code release early and often.

We describe three main notions of quality in the context of open source software development, namely, *quality by access*, *quality by development* and *quality by design*.

2.1 Quality by Access

Open access to source code is fundamental to the OSS development process. For peer developers and code reviewers to contribute to the development process, and therefore quality, it is necessary that access to software is as open as it possibly could be. The openness of access here implies availability of source code (and files) from an easily accessible medium such as the Internet. In particular, availability is unrestricted in a way that

- it requires no authorisation and is least exclusive in terms of membership of mailing lists, working groups or a formal organisation;
- the source code is in a format that is deemed suitable and workable for the purposes of review and development, and appropriate for free distribution.

Our notion of *quality by access* is essentially a measure of these provisions, which allow the very basis for further development and maintenance of software in an open source environment.

2.2 Quality by Development

While open access lays it all down, it is the actual execution and review of the source code that allows bug detection, and hence reporting and fixing, new feature and functionality development, and eventually evolution. The idea of *quality by development* is an attempt to measure the efficiency of such processes and the interaction between them. We break this down into five identifiable factors,

- precise and explicit understanding of software goals and requirements;
- choice of methodologies for testing, debugging and error and bug reporting;
- choice of programming languages and development environments;
- tools to provide effective communication, coordination and overall management of the project;
- facilitation of rapid frequency of beta releases.

While all these factors work together, the overall management of the project plays a more central role, with communication and coordination being the two key aspects of this. It helps developers understand the requirements of the software and allows them to choose appropriate development and debugging techniques. Moreover, it helps coordinate all the activities throughout the development phase of the software, which in turn eventually helps in frequent releases of the software. We demonstrate this relationship explicitly in Figure 3.

2.3 Quality by Design

The availability and development factors provide only the right environment. The end quality is inevitably judged by the design and implementation of the actual software and the code that underlies it. Our idea of *quality by design* corresponds to the more traditional notion of software quality. Two definitions somewhat indicative of this traditional notion include the definition of quality by IEEE [IEE99]:

> "the degree to which a system, component, or process meets (1) specified requirements, meets (2) customer or user needs or expectations,"

and another more recent by Pressman [Pre00]:

> "conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software."

For the purpose of OSS, we define *quality by design* as a specific measure of

- the use of recognised software design notations, such as UML, formal notations, such as logics and process calculi/algebras, and engineering techniques, such as object-orientation, component-based design and reuse, to provide correctness with respect to explicitly desired safety, security and non-functional properties, such as reliability and fault-tolerance, and
- the production and frequent update of appropriate and explicit documentation that helps both the users and future developers.

Design and engineering techniques should present a high degree of formality in order to facilitate verification using established formal methods and techniques such as theorem-proving, model-checking and model-based testing. Other contributions to this measure include attributes of quality, such as those specified by ISO 9126.

3 Relationships among Quality Factors for OSS

In this section we delve further into the three notions of quality by highlighting the factors that contribute to the overall quality of the OSS and by defining relationships between such factors. The overall quality of software due to each factor may be enhanced or undermined due to another related factor. We illustrate this relationship by using a **box** that represents quality provided by a quality factor and an **arrow** to denote that quality provided by a factor is dependent on the quality provided by another factor. So, for example, Figure 1 illustrates that quality provided by Factor 2 is dependent on quality provided by Factor 1. This *dependency* relation may be defined in both directions to indicate two quality factors complementing each other.

The main factors that contribute to *quality by access* are

- **suitable format** for the purposes of review, development and free distribution;
- **accessible medium** such as the Internet;

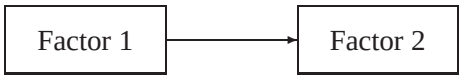

Figure 1: Dependency relation between factors

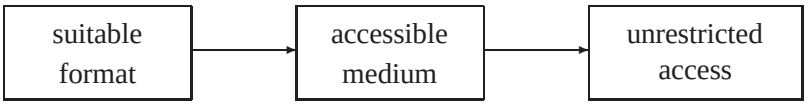

Figure 2: Quality by access

- **unrestricted access** to existing software code and documentation.

The relationships between these factors are shown in Figure 2. The use of a suitable format helps the quality provided by the use of an accessible medium, shown hereby with the help of an arrow between these two factors. Similarly, the use of an accessible medium helps with the accessability due to an absence of authorisation.

The five main factors that contribute to *quality by development*, as already mentioned in Section 2.2, are shown in Figure 3. The various relations centre around **effective communication, coordination and management**. This factor helps the quality provided by all other factors, and in particular, facilitates **frequent beta releases**. How much does effective communication and coordination helps the quality of the software is critically depedent on the **understanding of goals and requirements**, either directly, at the higher abstraction levels in the development process, or through a **choice of methodologies for testing and debugging** and a **choice of programming languages and environment** at the lower abstraction levels.

Note that coordination and communication lie at the heart of any open source software development, regardless of the hierarchy [GA04] of a project, as it serves to bring together and manage the various developmental activites in what is usually a widely dispersed community.

The main factors that contribute to *quality by design* are

- **use of recognised design and engineering techniques**;
- **correctness**;
- **formal analysis and verification**;
- **frequently updated documentation**.

As shown in Figure 4, software quality in terms of correctness of the code is helped by all other factors present here. The practice of recognised design and engineering techniques goes hand in hand with formal analysis and verification. Formal techniques allow more rigorous and automated support to assure safety and dependability attributes amongst others. Requirements engineering, software design and development, revision, maintenance and documentation are all to benefit from the use of formal methods. Suitability of some such methods is already demonstrated for open source software development [BP08]. Good updated documentation is also an important factor that only serves to help with the correctness of code, especially during

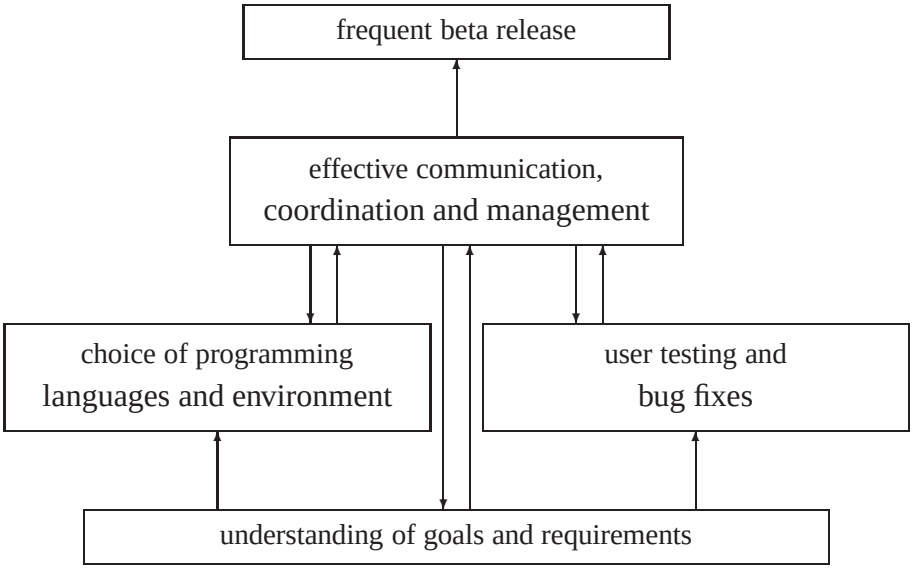

Figure 3: Quality by development

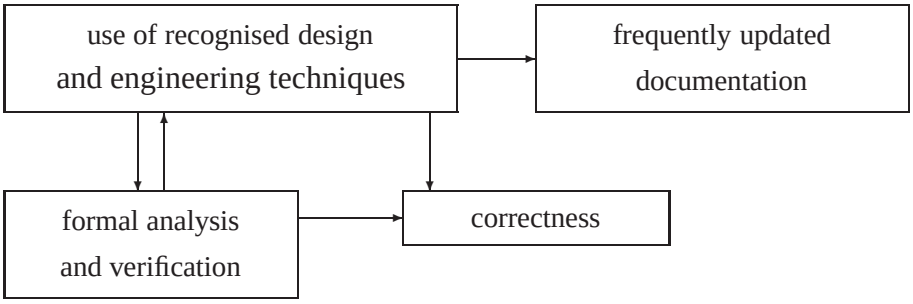

Figure 4: Quality by design

code debugging and updating.

We extend our dependency relation from Figure 1 to express this dependency between each of the three quality measures. *Quality by development* and *quality by design* both play an essential role in providing software quality and complement each other for this purpose. Underlying both measures is *quality by access* representing availability and openness of source code development, a principle that lies at the heart of open-source philosophy. These relationships are shown in Figure 5.

4 Related Work

The literature on quality for open source software is limited but varied. In this section we review some of this literature and discuss issues in relation to our model. Most of the work undertaken so far broadly divides into comparative case studies that focus on quality issues in particular open

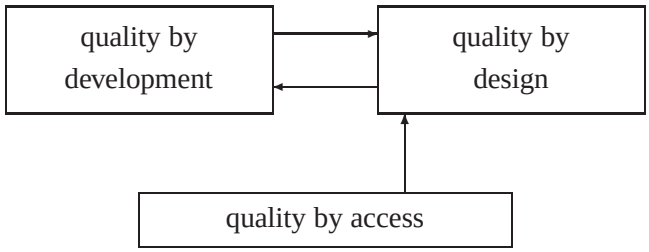

Figure 5: Relationships between quality metrics

source projects [MFH02], surveys of general quality assurance practices [ZE00, ZE03, HS02, YM06, Mic05] and particular aspects and issues of quality of open source software [YM06, Mic05, McC99, MHP05].

Zhao and Elbaum [ZE00] undertake a small survey to examine the factors underlying quality assurance methods of open source developers. Their work characterises the general attitude and practices of the open source community towards quality, realising quality assurance practices are somewhat different to those prevalent in traditional software development. The survey highlights the choice of programming languages and the availability of corresponding debugging tools as one of the factors in this respect. C, Perl and C++ are the three most commonly used languages (among the sample surveyed), with the GDB [SPS02], Perl Debugger [Fol04] and Electric Fence [EF0] as three most commonly used debugging tools. Interestingly, the survey also reveals wider tool support for longer a language has existed (with Java being an exception), where the availability corresponds to its widespread usage.

McConnell [McC99] acknowledges the efficiency of extensive field testing and peer review, along with an emphasis on the need for a comprehensive methodology for open source development. This is important if open source development is to be used for producing high quality complex software for use in critical domains.

Halloran and Scherlis [HS02] review a number of notable quality practices on some popular open source projects, of which good project communication and management is highlighted. This includes software bug and issue tracking, composition and configuration control, build management and computer and tool-assisted mediation. They also introduce the notion of a project wall that surrounds the project server, with the purpose of maximising outgoing information flow and controlling incoming information flow. The strength of this wall is then critical to the quality of the software produced.

Effective communication and management, the choice of programming languages and the choice of testing strategy emerge as the three most prominent factors affecting quality. These three factors form the core of the *quality by development* measure (see Figure 3) in our model. The three factors combine to become essential to the quality of the actual software code and the resulting product. Hedberg *et al* [HIRH07] emphasise the role good communication and management play in the production of good quality open source software. They portray open source development as essentially *developer-centric*, where the task of managing the project requires *good social and communication skills*. Such skills allow core developers to

- manage and motivate volunteers to review code and deliver patches,
- encourage users to test the software, with the aim to find and report bugs, and
- ensure that as a result of such "beta-testing" code is fixed.

Undoubtedly developers involved in open source projects do influence many critical aspects of the software that results, including most importantly the software architecture, as Arief *et al* [AGL01] highlight.

We model frequent beta release as a quality factor that is singularly dependent on effective communication and management. Software release, in an open source environment, requires effective control and coordination amongst the developer community. Michlmayr [Mic05] draws attention to this particular issue, and attributes the problems associated with release management to a lack of such skills in this community, who are mostly programmers and volunteers as opposed to dedicated managers.

Michlmayr [Mic05] goes further to note that the developer community may not always heed to user requirements for this purpose, resulting in software releases that do not meet their intended goals and purposes. Our model takes this into account and places the understanding of goals and requirements as a complementing factor to effective coordination and management.

While we acknowledge the link between frequent software release, enabling increased testing and feedback, and software quality, Schmidt and Porter [SP01] find it as a serious hindrance to quality assurance as it contributes to a serious rise in the cost of long-term maintenance and evolution.

De Groot *et al* [GKAG06] define a Software Quality Observatory to evaluate and quantify the quality of an open source project. The tool is based on a plugin-based SOA that supports the mixing and matching of metrics extraction suits, source code repositories and transformational filtres [GKS+07].

5 Discussion

Our effort ultimately aims to develop a quantitative metric for quality. Such a metric could be useful in formal assessment and comparison of open source development. While such a metric also raises many questions about how various factors are to be measured, we discuss some early ideas on our approach to deriving a quantitative metric for the moment, and leave measurement issues to be dealt with later.

We consider each quality factor, as introduced in Section 2, individually and assign it an index value within a limited range [0.1,1]. The index denotes the quality provided by the factor for a given development where the higher the value the higher the quality provided by a factor. We assume every factor does contribute towards the overall quality, at least to correspond to a minimum index of 0.1. For a factor F_1 we write the index as $I(F_1)$.

For two factors related to each other, as shown in Figure 6, we calculate the total metric of the two factors, F_1 and F_2, as a product of their index values. But since quality provided by F_2 is enhanced by the quality provided by F_1, we need to factor in this enhancement. This will allow us to model the understandably varying degrees of dependency that exist between factors. We calculate this by adding a proportion q of the quality provided by F_1 to $I(F_2)$, which then helps us to derive the total quality as $I(F_1) \times (I(F_2) + I(F_1) \times q)$.

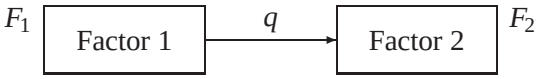

Figure 6: Metric for dependency relation between factors

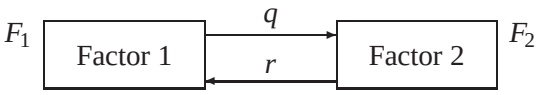

Figure 7: Metric for dependency relation between complementing factors

For complementing factors, where r is the proportion of quality provided by F_2 to F_1, we calculate the total as $(I(F_1) + I(F_2) \times r) \times (I(F_2) + I(F_1) \times q)$, as shown in Figure 7.

The calculations presented above allow us to sum up the total contribution of the various factors and represent it as a quality metric. Such a metric could be useful to judge how different factors influence the overall quality of software produced as a result of various strategies adopted in open source developments. Further work aims to determine ways to measure quality provided by factors and demonstrate our approach by applying it to an existing open source project.

The ideas presented in this paper are preliminary, and are shaped to stimulate discussion and promote further research in this area. Further work is underway to assess how effective is our approach to model quality with respect to established OSS projects. Our particular focus is on applying our model to existing projects and analyse the quality factor dependencies that emerge. Some are likely to correspond to the ones we have in our model, while some may not; it is the latter that are of interest to us. We hope to report back the results to the wider community. The challenge is to develop a comprehensive model which is both well-defined, in order for it to be precise and effective in judging the quality of OSS, and flexible to accommodate the yet developing and emerging notions of quality.

Bibliography

[AGL01] B. Arief, C. Gacek, T. Lawrie. Software Architectures and Open Source Software - Where can Research Leverage the Most? In *Proceedings of 1st Workshop on Open Source Software Engineering, Toronto, Canada*. Pp. 3–5. 2001.

[BP08] P. T. Breuer, S. Pickin. Approximate verification in an open source world. *Innovations in System and Software Engineering* 4(1):87–105, April 2008. Springer-Verlag.

[Cov08] Open Source Report 2008. 2008. http://www.coverity.com.

[EF0] Electric Fence debugger. Last Accessed on 7th December 2006. http://perens.com/FreeSoftware/ElectricFence/

[Fol04] R. Foley. *Perl Debugger Pocket Reference*. O'Reilly Press, January 2004.

[GA04] C. Gacek, B. Arief. The Many Meanings of Open Source. *IEEE Software* 21(1):34–40, January/February 2004.

[GKAG06] A. de Groot, S. Kügler, P. Adams, G. Gousios. Open source software quality observation. In *IFIP International Federation for Information Processing*. Volume 203, pp. 57–63. Springer, 2006.

[GKS+07] G. Gousios, V. Karakoidas, K. Stroggylos, P. Louridas, V. Vlachos, D. Spinellis. Software Quality Assessment of Open Source Software. In *Proceedings of the 11th Panhellenic Conference on Informatics*. 2007.

[HIRH07] H. Hedberg, N. Iivari, M. Rajanen, L. Harjumaa. Software Architectures and Open Source Software - Where can Research Leverage the Most? In *Proceedings of First Workshop on Emerging Trends in FLOSS Research and Development, Minneapolis, USA (FLOSS'07)*. Pp. 1–5. 2007.

[HS02] T. J. Halloran, W. L. Scherlis. High Quality and Open Source Software Practices. In *2nd Workshop on Open Source Software Engineering*. May 2002.

[IEE99] IEEE Std 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology. February 1999.

[McC99] S. McConnell. Open Source Methodology: Ready for Prime Time? *IEEE Software* 16(4):6–8, july/august 1999. IEEE Computer Society.

[MFH02] A. Mockus, R. T. Fielding, J. D. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology* 11(3):309–346, july 2002. ACM Press.

[MHP05] M. Michlmayr, F. Hunt, D. Probert. Quality Practices and Problems in Free Software Projects. In Scotto and Succi (eds.), *Proceedings of the First International Conference on Open Source Systems*. Pp. 24–28. Genova, Italy, 2005.

[Mic05] M. Michlmayr. Quality Improvement in Volunteer Free Software Projects: Exploring the Impact of Release Management. In Scotto and Succi (eds.), *Proceedings of the First International Conference on Open Source Systems*. Pp. 309–310. Genova, Italy, 2005.

[Muf06] M. Muffatto. *Open Source — A Multidisciplinary Approach*. Imperial College Press, 2006.

[Pre00] S. R. Pressman. *Software Engineering - A Practitioner's Approach*. McGraw-Hill International, London, 2000.

[SP01] D. C. Schmidt, A. Porter. Leveraging Open-Source Communities to improve the Quality and Performance of Open-Source Software. In *Proceedings of 1st Workshop on Open Source Software Engineering, Toronto, Canada*. 2001.

[SPS02] R. M. Stallman, R. Pesch, S. Shebs. *Debugging with GDB:The GNU Source-Level Debugger*. Free Software Foundation, January 2002.

[YM06] C. Yilmaz, A. Memon. Techniques and Processes for Improving the Quality and Performance of Open-Source Software. *Software Process: Improvement and Practice* 11(2):163–176, May 2006. John Wiley & Sons.

[ZE00] L. Zhao, S. Elbaum. A survey on quality related activities in open source. *ACM SIGSOFT Software Engineering Notes* 25(3):53–57, May 2000. ACM Press New York, NY, USA.

[ZE03] L. Zhao, S. Elbaum. Quality assurance under the open source development model. *Journal of Systems and Software* 66(1):65–75, April 2003. Elsevier Science.

Acknowledgements:
Early ideas on quantitative metrics have benefitted from discussions with Dr. Vasos Pavlika.