The QualOSS Open Source Assessment Model
Measuring the Performance of Open Source Communities

Martín Soto and Marcus Ciolkowski
Fraunhofer Institute for Experimental Software Engineering (IESE)
Kaiserslautern, Germany
{soto, ciolkows}@iese.fraunhofer.de

Abstract

Open Source Software (OSS) has an increasing importance for the software industry. Similar to traditional (closed) software acquisition, OSS acquisition requires an assessment of whether its quality is sufficient for the intended purpose, and whether the chances of being maintained and supported in the future, as well as of keeping certain quality standards over time, are sufficiently high. In this paper, we present an approach toward a comprehensive measurement framework for OSS projects, developed in the EU project QualOSS. This approach takes into account product quality as well as process maturity and sustainability of the underlying OSS community.

1. Introduction

Free and Open Source Software (F/OSS) plays and increasingly important role in companies and organizations of all types and sizes. Clearly, the potentially large benefits offered by F/OSS are appealing to decision makers in charge of software acquisition. In many situations, F/OSS offers valuable functionality with zero licensing costs, while simultaneously allowing for independence from particular software vendors.

The fact that F/OSS software is free of licensing costs, however, does not make its adoption completely cost-free. Indeed, the implementation of F/OSS systems in an organization, as well as their use as components in larger systems, are accompanied by all manner of risks and uncertainties. A wrong decision regarding the choice of F/OSS applications or components for a particular purpose may have serious negative consequences for the organization involved.

For this reason, the EU Project QualOSS has worked on developing a model framework for evaluating F/OSS projects in order to support procurement decisions. Given the variety of potential use cases for F/OSS in organizations, supporting this type of decisions is not an easy task. On the one hand, the decision to use a particular F/OSS application or component may be related to product characteristics, such as its particular functionality, its resource consumption under particular types of load, or the size and structure of its code, among many others. On the other hand, acquisition decisions are often related to the chances a software product has of being maintained and supported in the future, as well as of keeping certain quality standards over time. The QualOSS Model is therefore intended to be comprehensive, covering both the *robustness* of the evaluated product as such, as well as its *evolvability; that is*, its ability to thrive and keep growing over time.

The measurement strategies corresponding to these two aspects are widely different from each other. For the most part, product aspects of F/OSS software can be measured using the same techniques available for other types of software. Indeed, F/OSS often offers an advantage in this respect, because the source code is always available and can be readily analyzed. In fact, in recent years, F/OSS has often been the target of quantitative code quality analysis for both research and industrial purposes. [2]

Quality aspects related to evolvability, on the other hand, cannot usually be analyzed using existing techniques. In fact, evolvability is .mainly associated with the community behind a F/OSS component, which can be seen as its supplier. In the case of commercial software, supplier companies are normally evaluated using process assessment methods such as those defined by the CMMI-DEV [3] or SPICE [4] standards, which, for a number of reasons, can hardly be applied in their original form to a F/OSS community. Fortunately, the open nature of F/OSS projects provides us with plenty of information sources, such as source code, mailing lists, bug tracking systems, and versioning systems. By analyzing these information repositories, it is possible to investigate many quality aspects related to a F/OSS community in a systematic and reliable way.

This paper concentrates on the subset of the QualOSS Model that contains specialized metrics for analyzing the performance of F/OSS communities. The main questions addressed by these metrics is whether a community is likely to survive in the long term (*sustainability*) and whether it will be able to consistently produce high quality software over time (*maturity*). In

978-1-4244-4841-8/09/$25.00 ©2009 IEEE

the following, we provide a general description of the QualOSS model, discuss its community-related aspects in some more detail, and outline our approach for evaluating these aspects.

2. Related Work: OSS Assessment

In recent years, Open Source Software has often been used as the target of quantitative analyses of code quality, mostly due to the fact that large code repositories are available for analysis. Many publications exist on (semi-)automatic analysis of code, mailing lists, bug tracking, and versioning systems. Contrary to what happens with code and repository analysis, few publications have addressed OSS processes so far. A paper by Michlmayr [2] is one notable exception, providing evidence of disciplined processes in OSS projects and relating it with project success.

As a reaction to the insight that software quality is not restricted to code aspects, assessment models for OSS projects have emerged, whose aim is to support potential OSS users in making decisions regarding the selection of OSS products. The most prominent examples are the *Qualification and Selection of Open Source Software* (QSOS) model [9], two different models called *Open Source Maturity Model* (OSMM) —one from CapGemini [7] and one from Navica [8]— and the *Open Business Readiness Rating* (OpenBRR) model [6]. Although these models take the OSS product into account (i.e., code, documentation), as well as the community that produces it, they only have a rudimentary process perspective, if any. This lack of coverage for the process perspective constitutes one of our main motivations for proposing the more comprehensive approach discussed here.

3. The QualOSS Model

As already mentioned, the QualOSS model was originally designed to support the quality evaluation of F/OSS projects, with a focus on evolvability and robustness. It is composed of three types of interrelated elements: quality characteristics, metrics, and indicators. Quality characteristics correspond to the attributes of a product or community that we consider relevant for evaluation. Metrics correspond to concrete aspects we can measure—on a product or on its associated community assets—which we expect to be correlated with our targeted quality characteristics. Finally, indicators interpret a set of measurement values related to one quality characteristic; that is, they define how to aggregate and evaluate the measurement values in order to obtain a consolidated value that can be readily used by decision makers when performing an evaluation.

The quality characteristics in the model are organized in a hierarchy of two levels that we call characteristics and subcharacteristics. The subcharacteristics are considered to contribute in some way or another to the main characteristic they belong to. In order to define our hierarchy of quality characteristics, we relied mainly on three sources: (1) Related work on F/OSS quality models, (2) general standards for software quality, such as ISO 9126 [5], and (3) expert opinion; that is, we conducted interviews among industry stakeholders to initially derive relevant criteria for the QualOSS model

Table 1: Product-related quality characteristics in QualOSS

Char.	Definition
Maintainability	The degree to which the software product can be modified. Modifications may include corrections, improvements, or adaptation of the software to changes in the environment, and in requirements and functional specifications.
Reliability	The degree to which the software product can maintain a specified level of performance when used under specified conditions.
Transferability (Portability)	The degree to which the software product can be transferred from one environment to another.
Operability	The degree to which the software product can be understood, learned, used and is attractive to the user, when used under specified conditions.
Performance	The degree to which the software product provides appropriate performance, relative to the amount of resources used, under stated conditions.
Functional Suitability	The degree to which the software product provides functions that meet stated and implied needs when the software is used under specified conditions.
Security	The ability of system items to protect themselves from accidental or malicious access, use, modification, destruction, or disclosure.
Compatibility	The ability of two or more systems or components to exchange information and/or to perform their required functions while sharing the same hardware or software environment.

Given our emphasis on covering not only F/OSS products but the communities behind them, we have grouped the quality characteristics into two groups: those that relate to the product, and those that relate to the community. On the product side, the QualOSS model covers the top-level quality characteristics listed in Table 1. The community side of the model, in turn, covers the characteristics listed in Table 2.

The remainder of this article provides more details about this second group of quality characteristics.

Table 2: Community-related quality charact. in QualOSS

Charact.	Definition
Maintenance capacity	The ability of a community to provide the resources necessary for maintaining its product(s) (e.g., implement changes, remove bugs, provide support) over a certain period of time
Sustainability	The likelihood that a F/OSS community remains able to maintain the product or products it develops over an extended period of time.
Process Maturity	The ability of a developer community to consistently achieve development-related goals

978-1-4244-4841-8/09/$25.00 ©2009 IEEE

Charact.	Definition
	(e.g., quality goals) by following established processes. Additionally, the level to which the processes followed by a development community are able to guarantee that certain desired product characteristics will be present in the product.

4. Community Measures

Contrary to what happens in the product domain, development communities are the main differencing aspect between commercial and F/OSS products. It is not only that development happens in a loose community of (often volunteer) peer developers with almost no hierarchy, but that many important assets of the community are also open for inspection. This way, mailing lists, discussion forums, version management repositories, bug tracking systems, and a number of other resources are available on the Internet for interested parties to study and contribute to.

Our approach to F/OSS community evaluation is based on looking at such open community assets in order to assess relevant community quality characteristics. Our base assumption regarding community quality is twofold. On the one hand, certain characteristics of the community strongly influence product quality, especially when observed over an extended period of time. On the other hand, the ability of a F/OSS community to remain active over time is obviously very important for product survival, and thus very relevant when considering sustainability. In the following, we discuss these characteristics in some more detail.

4.1. Maintenance Capacity

Two aspects are particularly relevant to maintenance capacity, namely, the number of contributors to a project, and the amount of time they are able and willing to contribute to the development effort. The QualOSS model attempts to create a general profile of the contributor community and its level of activity by aggregating data coming from the analysis of mailing list, forum, and bug tracking system archives, as well as from versioning system logs.

Versioning logs, for example, provide data about the number of code contributors, as well as about the size and frequency of their individual contributions. This data can be related to the measures such as the project code size and the number of open problem reports to evaluate the capacity of the community for maintaining the software. Similarly, the activity of mailing lists and discussion forums, as well as the frequency of the contributions to the bug report system, can be used to produce a similar picture of a product's user community.

4.2. Sustainability

Community sustainability is affected by factors such as the composition of a community, and its ability to grow or regenerate (i.e., engage new members to take the place of those leaving the community). For instance, if a community is mainly composed of employees of a particular company, there is a higher risk of the project becoming stalled if the company decides to cut its financial support. On the other hand, a community that is composed only of volunteers may be less likely to disappear suddenly, but may also have less resources available to keep the project running over time. Consequently, heterogeneity in a F/OSS community is expected to enhance sustainability. Heterogeneity can be evaluated with relatively simple methods, such as looking at the contributors' email or web addresses in order to guess their affiliations. Although this technique is not 100% reliable, it can provide a general idea of a community's composition.

With regard to grow and regeneration, a community's history in this respect can be a good predictor of its future behavior. Past regeneration can be observed, for example, by analyzing contributions to the version management system or to the mailing lists over time. For example, regeneration of the developer community can be observed by looking at the first and last contributions of each individual developer over the project's history. If the same core group of developers have been active along the project history, this does not reflect significant regeneration. On the other hand, if the number of developers remained stable or tended to grow despite of individual developers leaving the project at times—as evidenced by their lack of activity after a certain point in time—there is a clear tendency of community regeneration. A regenerating community can be considered more sustainable since it is more likely to survive the lost of important contributors over time.

4.3. Process Maturity

It is a common belief that F/OSS communities operate in an ad-hoc, chaotic way. Evidence shows, however, that this is far from being the case for many successful F/OSS communities. Indeed, there is evidence of good practices being applied in an established and disciplined fashion by a variety of F/OSS communities and with regard to different areas of the software development process. Examples of disciplined good practices can be observed in prominent F/OSS communities in areas such as Version and Configuration Management, Release Management, and Requirements Management. We are convinced that many of these practices correspond to the spirit, if not directly to the letter, of the practices and goals specified by well-

978-1-4244-4841-8/09/$25.00 ©2009 IEEE

known process assessment standards such as CMMI-DEV.

In its current form, our Open Source process evaluation framework covers a number of basic software development tasks. Each of these tasks is evaluated with respect to five main questions, which constitute a simplified form of the sort of assessment a standard maturity model would require:

1) *Is there a documented process for the task?*
2) *Is there an established process for the task?*
3) *If there is an established process, is it executed consistently?*
4) *If both an established, consistent process, and a documented process could be found, do they match?*
5) *Is the process adequate for its intended purpose?*

In order to address these questions for each of our selected tasks, we have already defined simple evaluation procedures. These procedures use data from public data sources such as mailing lists, bug/issue tracking systems and version management systems, among others. The basic processes currently covered by the QualOSS process assessment are listed in Table 3.

Table 3: Processes covered by QualOSS

Process	Description
Change submission and review	Submit changes (e.g., defect corrections, enhancements) to the project for potential inclusion. Also, review changes submitted by community members.
Peer review of changes	In some projects, changes proposed by developers with direct commit rights are also subject to review by other community members. This type of peer reviews can significantly contribute to code quality.
Propose significant enhancements	Some projects have disciplined processes that allow community members to formally propose enhancements for discussion by the community.
Report and handle issues with the product	For obvious reasons, this process is present in almost all Open Source projects in some form or another.
Test the program(s) produced by the project	Most projects doing repeatable testing do it by defining an automated test suite. If no test suite is available, there may be explicitly defined manual test cases, but this is much less likely to happen.
Plan releases	Either releases are done on a time-based fashion or based on a feature "road map".
Release new versions of the product	Release processes in Open Source often include the creation of a number of alpha, beta and release-candidate versions that are delivered by the developers in order to obtain feedback from the community (active users of an OSS system are often willing to test these versions and report about problems they may find). Release processes also often include running a test suite or performing other forms of formal testing.
Backport corrections in the current release to previous stable releases	When a stable and an unstable (development) branch of a project are maintained simultaneously, so-called *backports* are often necessary that move corrections or selected improvements made to the development branch into the stable branch.

5. Evaluation and Calibration

At the time of this writing (June 2009) we are conducting the final evaluation of the QualOSS model. The evaluation targets up to 20 F/OSS projects, which will be assessed by applying the complete QualOSS model to them. Projects selected for the evaluation include projects that, according to expert judgment, are considered successful, together with projects that are considered unsuccessful. By comparing results for these two project groups, we expect to be able to correlate measurement results with the overall quality of a project and, in particular, of the community behind it. We plan to utilize this comparison for adjusting the indicators and weights used for aggregating indicators, to make sure that the QualOSS model is able to discriminate between successful and unsuccessful projects.

Additional aspects of the evaluation include the effort required for conducting an assessment. Some of the community measures can be automatically computed, and their collection requires only setting up measurement tools. Other measures, for example the ones related to process maturity, require more manual effort. Current results indicate that performing the full QualOSS requires less than five person-days for large projects.

ACKNOWLEDGMENTS

This work was supported in part by the EU QualOSS project (grant number: 033547, IST-2005-2.5.5). We would like to thank Sonnhild Namingha, from Fraunhofer IESE, for proofreading this paper.

REFERENCES

[1] Marcus Ciolkowski and Martín Soto: Towards a Comprehensive Approach for Assessing Open Source Projects. MetriKon 2008, Munich, Germany.

[2] Martin Michlmayr. Software Process Maturity and the Success of Free Software Projects. In: Zieliński, K., Szmuc, T. (Eds.), Software Engineering: Evolution and Emerging Technologies.

[3] Software Engineering Institute (SEI): Capability Maturity Model Integration (CMMI) for Development, Version 1.2, 2006.

[4] ISO/IEC 15504-5:2006, Software Process Improvement and Capability Determination, Part 5.

[5] ISO/IEC 9126 International Standard, Software engineering – Product quality, Part 1: Quality model, 2001.

[6] Business Readiness Rating. Information available from http://www.openbrr.org/. Last checked 2009-03-09.

[7] Cap Gemini: OSS Partner Portal. Internet address: http://www.osspartner.com/. Last checked 2009-03-09.

[8] Navica Software Web Site. Internet address: http://www.navicasoft.com/. Last checked 2009-03-09.

[9] Qualification and Selection of Open Source software (QSOS) Web Site. Internet address: http://www.qsos.org/. Last checked 2009-03-09.

978-1-4244-4841-8/09/$25.00 ©2009 IEEE