The QualOSS Process Evaluation: Initial Experiences with Assessing Open Source Processes

Martín Soto and Marcus Ciolkowski

Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany
{soto,ciolkows}@iese.fraunhofer.de

Abstract. For traditional software development, process maturity models (CMMI, SPICE) have long been used to assess expected product quality and project predictability. For the case of OSS, however, these models are generally perceived as inadequate. In practice, though, many OSS communities are well-organized, and there is evidence of varying levels of process maturity in OSS projects. This paper presents work in progress—performed as part of the EU project QualOSS—on developing a process evaluation framework specifically aimed at OSS projects. We present a first version of our evaluation procedures, and discuss some lessons learned during its preliminary application to a small number of OSS projects.

Keywords: Software process, Open Source Software, OSS, process assessment, process evaluation, QualOSS, software quality.

1 Introduction

Since the introduction of the Capability Maturity Model (CMM) in the early 1980s, maturity-oriented process assessment models have become a fundamental tool for determining the extent to which an organization can deliver software on time and with an acceptable level of quality. Currently, the most prominent examples of such process assessment models are CMMI-DEV (Capability Maturity Model® Integration for Development [10]) and SPICE (Software Process Improvement and Capability dEtermination [4]).

The growing popularity of Open Source Software (OSS) constitutes a big challenge to software process assessment, since, at first sight, maturity-oriented models appear very difficult to apply to OSS development. On the one hand, they seem to expect an organizational structure that is not present in most OSS communities, and, on the other hand, it is a widespread belief that OSS communities operate in an essentially chaotic way, and that, for this reason, no systematic development processes can be taking place during OSS development. Consequently, most casual observers would regard traditional maturity models as completely inappropriate for OSS software.

We disagree with this vision. The main assumption underlying process assessment approaches is that mature processes consistently lead to higher-quality products, whereas for an organization with immature processes, the capacity to deliver high-quality products is unreliable and cannot be predicted. There is no reason to believe that this assumption is not valid for OSS. Concretely, we expect that a higher level of process maturity will lead to better products and more sustainable communities, and that successful OSS communities often owe a good portion of their success to the introduction of sound software processes.

Indeed, many OSS communities have been able to consistently produce software of adequate quality, making regular releases over the years. There is evidence that this consistency does not stem from some mysterious property of OSS development that makes it work against all odds, or from the sheer talent of individual developers, but that it could be the result of good software development practices being applied and enforced by OSS communities in a disciplined fashion [7]. For this reason, the EU project QualOSS—which is generally concerned with the overall quality of OSS products, as well as with the sustainability of the communities around them—decided to add a process evaluation framework to its quality model, which is aimed at determining the ability of an OSS community to consistently deliver adequate products over time.

In this paper, we describe the first version of this process evaluation framework, and discuss our preliminary experience with applying it to a small number of OSS projects. In order to provide some background to the reader, Section 2 briefly describes the overall quality model defined by the QualOSS project. After a short discussion of related work in Section 3, Section 4 presents the QualOSS process evaluation in detail. Our initial experience with the process evaluation is discussed in Section 5. We close with some general conclusions and a brief discussion of future work in Section 6.

2 The QualOSS Quality Model

The process evaluation framework we describe in this paper is one component of the comprehensive quality model developed for the *Quality of Open Source Software* (QualOSS) project. Since the process evaluation framework was designed from the ground up to contribute to the overall QualOSS model, we start by describing it briefly.

The QualOSS quality model (or, simply, "QualOSS model" for short) is intended to support the quality evaluation of OSS projects, with a focus on evolvability and robustness. One central, underlying assumption while defining the model has been that the quality of a software product is not only related to the product itself (code, documentation, etc.), but also to the way the product is developed and distributed. For this reason, and since the development of OSS products is the responsibility of an open community, the QualOSS model takes both product- and community-related issues into account on an equal basis, and as comprehensively as possible.

The QualOSS model is composed of three types of interrelated elements: quality characteristics, metrics, and indicators. Quality characteristics correspond to the concrete attributes of a product or community that we consider relevant for evaluation (see below for an explanation of how these characteristics were chosen). Metrics correspond to concrete aspects we can measure on a product or on its associated community assets that we expect to be correlated with our targeted quality characteristics. Finally, indicators define how to aggregate and evaluate the measurement values resulting from applying metrics to a product or community 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 for reasons of simplicity. The subcharacteristics are considered to contribute in one way or another to the main characteristic they belong to. For defining our hierarchy of quality characteristics, we relied mainly on three sources: (1) related work on OSS quality models, (2) general standards for software quality, such as ISO 9126 [6], and (3) expert opinion. For the third source, we conducted interviews among industry stakeholders to derive relevant criteria for the QualOSS model.

Given our emphasis on covering not only OSS products but also 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 following top-level quality characteristics:

– *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.

The community side of the model, in turn, covers the following characteristics:

– *Maintenance capacity:* The ability of a community to provide the resources necessary for maintaining its product(s) (e.g., implement changes, remove defects, provide support) over a certain period of time.
– *Sustainability:* The likelihood that an OSS community remains capable of maintaining 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 (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.

The QualOSS process evaluation framework is aimed at covering the last characteristic mentioned, namely, process maturity. In what follows, we describe this framework in more detail.

3 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 [7] 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 [2] and one from Navica [8]—and the *Open Business Readiness Rating* (OpenBRR) model [1]. 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. For example, QSOS considers two process criteria: quality assurance processes (with levels none, informal, supported by tools), and bug/feature request tools (none, standard tools, active use of tools), which, in our opinion, are far from covering the wide variety of quality-relevant processes typically observed in OSS development. This lack of coverage for the process perspective constitutes one of our main motivations for proposing the more comprehensive approach discussed here.

4 Towards a Process Maturity Model for OSS

As discussed in the introduction, the idea of assessing an OSS community in order to determine which good practices it follows, as well as how established these practices are, is perfectly reasonable. Still, it is true that existing process assessment models cannot generally be applied directly to OSS, as they include too many elements that are specific to companies and other conventional development organizations. In this section, we describe our process evaluation framework, which is directly aimed at OSS development. This model reuses a number of the ideas present in existing maturity models, but adapts them in order to make them more directly applicable in an OSS context.

4.1 Maturity Models as a Basis for Open Source Process Assessment

In order to create an assessment model for OSS process maturity, we started by reviewing existing maturity models with the purpose of extracting, and, where necessary, adapting some of their elements to the specifics of OSS. Concretely, we used the *Capability Maturity Model for Software Development* (CMMI-DEV) as a starting point. Released in 2006, the current CMMI-DEV model is the latest version in a series of maturity models started in the 1980s by Humphrey's Capability Maturity Model (CMM). CMMI-DEV covers 22 process areas, ranging from process improvement practices to specific development practices. Each process area is subdivided into a number of goals, which, in turn, are structured as sets of practices. Goals and practices are associated to process maturity levels (also called *capability levels* when they are related to a single process area). In order to be classified at a particular maturity level, an organization must have implemented all practices required by that level.

Given how comprehensive CMMI-DEV is, reaching its highest capability levels represents a serious challenge for any software development organization. Clearly, OSS communities are not an exception in this respect, and, in addition, the vast majority of them are not involved in any explicit process improvement efforts. Consequently, most, if not all, OSS communities are still quite far from reaching the levels of process discipline required by the higher levels of CMMI-DEV.

This last fact notwithstanding, there is evidence of good practices being applied in an established and disciplined fashion by a variety of OSS communities and with regard to different areas of the software development process. We think that many of these practices correspond to the spirit, if not directly to the letter, of the practices and goals specified by CMMI-DEV.

Some examples of such disciplined good practices, observed in prominent OSS communities, are:

– *Version/Configuration Management:* Many OSS projects rely on advanced versioning tools for managing their source code. In most cases, access to such systems will be carefully regulated, and the processes for creating new versions are well established and enforced.
– *Release Management:* The GNOME Desktop project, as well as the popular GNU/Linux distribution Ubuntu, both have strict 6-month release cycles that have been successfully operating for years. The complex coordination process required for each such cycle is well documented and carefully supervised and enforced by an established release board.
– *Requirements Management:* The community behind the Python programming language has a well-documented requirements elicitation and management process as represented by the so-called *Python Improvement Proposals* (PIPs). Proposals for language enhancements are presented by community members and thoroughly refined through feedback from the community until they are considered ready for implementation. The process is conducted in the open and actively enforced by the community.

Many other similar examples can be found by directly observing the dynamics of OSS communities. This led us to believe that, despite the inviability of applying a full-fledged process maturity model to OSS, a process evaluation model for OSS is not only viable, but potentially very useful in order to gauge the ability of OSS communities to consistently deliver software of appropriate quality. This belief constitutes the main motivation for the QualOSS process evaluation framework described here.

4.2 The Generic QualOSS Process Evaluation

In its current form, our Open Source process evaluation framework covers a number of basic software development tasks (described in more detail in the next subsection).

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 produce assessment results that allow for comparison of a project's performance in different areas, the answers to these questions are encoded in a predefined, normalized form. These basic results, in turn, are used to compute indicators that are integrated into the QualOSS model, and that, similar to other QualOSS metrics, are intended to contribute to an overall view of an OSS project's quality.

In order to address these questions for each of our selected tasks, we have already defined simple evaluation procedures. In the following, we outline these procedures.

Question 1 is concerned with process documentation. Although process documentation is seldom found under that name for Open Source projects, many projects have indeed documented procedures for a variety of development tasks. The reasons for providing documentation are often related with making it easier for external contributors to perform certain tasks (e.g., submit a problem report or a so-called *patch file* with a correction), as well as with making certain tasks more reliable (release processes are a typical case). Our procedure for finding documentation for a task is based on searching through the Internet resources made available by a given project for the relevant information as follows:

1. Check project resources for documentation regarding the task. Perform an Internet search if necessary. Acceptable documentation are explicit documents (Web/Wiki pages, archived mail/forum messages) that contain direct instructions about performing the task. In some cases, these are presented as templates, or as a set of examples.
2. If no explicit documentation was found, check if a tool is being used to support the task. If this is the case, check if the tool can be used in a self-explanatory manner. If this is the case, this can be accepted as documentation.
3. If 30 minutes of search do not yield any positive results, stop searching.

The final step confines the evaluation to a time box. This is important because, in fact, we can never be sure that there is no documentation about a task, only that it could not be found with reasonable search effort.

The second question is concerned with how established a process is. Notice that this question is, to a large extent, independent from the first one, because undocumented processes can nonetheless be well established, and documented processes may not be followed as prescribed. In order to check for established processes, standard maturity models use the fact that such processes leave a *paper trail* behind them that can be used to observe them in a very reliable manner. If such a trail cannot be found, the odds are very high that the process is not established, e.g., not followed at all, or not followed in a consistent manner. Strictly speaking, of course, a paper trail cannot be found for OSS processes, but a data trail is often seen when looking at the diverse data repositories that belong to a project, such as:

- Internetbased tools, if the process is supported by a tool. For example, such processes as defect reporting and issue management can be analyzed by looking at the discussions stored in a project's bug/issue tracking system.
- Mailing lists, forums, Wikis, etc, used by community members to collaborate while performing the process. These repositories are useful, for instance, to track decision-related processes such as release planning, or to follow the interaction between developers and testers in preparation for a release.
- Internet-based repositories used to publish the results of a process, such as versioning repositories or download servers.

The procedure used to evaluate how established a process is consists of identifying specific instances of process execution in the potential process trail:

1. Determine the period of time the process has been/was active, by looking at the dates for the identified instances.
2. Identify instances where the process was successfully completed.
3. Identify instances where the process was not successfully completed/was left unfinished.
4. Identify currently running instances.
5. Use the identified instances to classify the process (see below).
6. If the number of instances available is large, the analysis can be performed by randomly sampling a smaller number of them.

The outcome of this evaluation should be one of the following four possible results:

1. *No established process:* no data trail found, or too few instances to be representative.
2. *Dead process:* tried at some point, but no evidence of continued use, no instances currently active.
3. *Young/immature process:* introduced recently, few actual instances, but instances appear active.
4. *Established process:* many successful completed instances, significant number of active instances.

The third question, which is subordinated to the previous one, refers to the consistency with which a process is executed over time. Clearly, this question can also be answered by looking at the process trail in order to sample instances of the established process for consistency. The purpose of this inspection is to look for potential significant variations in the way individual instances are executed. The evaluation should result in one of the following values:

1. *Not applicable:* no established process.
2. *Low consistency:* instances vary strongly in the way they are executed.
3. *High consistency:* relatively few variations between instances.

The fourth question has to do with the degree of coincidence between the documented process and the process that is actually executed. It is the last question of those concerned with the process maturity in itself, and depends on the previous ones being answered in a positive way. The evaluation procedure, of course, consists of comparing a representative number of instances of the process with the identified process documentation. Possible results for this evaluation are:

1. *Not applicable:* no documented process, no consistent process.
2. *Low agreement:* low agreement between documentation and established practice.
3. *High agreement:* high agreement between documentation and established practice.

The fifth and final question is concerned with how adequate the process is for the task it is intended for. This is, of course, a difficult question, not only because it is specific to each particular task, but because experts often disagree regarding the practices that are appropriate for a certain task. Our approach to handling this problem is to provide a list of additional questions that address the specificities of every task. These questions are normally not comprehensive, but provide a minimum checklist that helps to make sure that essential aspects of the corresponding process are being taken into account. We see these questions only as complementary to the first four assessment questions, because, clearly, if a process is established in the sense defined above, it is probably adequate to a certain measure, given how pragmatic OSS communities usually are.

4.3 Process Areas Currently Covered by QualOSS

As already mentioned, the QualOSS process evaluation covers a number of software development related tasks that are usually important for the success of an OSS project. The following table lists the tasks that are currently covered (left column) and provides a brief description for each of them, together with some information about where their process data trail could be found (right column). This is just an initial selection of tasks, which we are likely to extend as we gain experience with the process evaluation.

Task	*Description and Evidence Sources*
Change submission	Submit changes (e.g., defect corrections, enhancements), typically in the form of so-called patch files, to the project for potential inclusion. This task is restricted to changes proposed by community members who do not have commit rights to the main project versioning repository, and thus cannot change the project's code directly. Common methods used to submit changes include sending them to a mailing list, putting them in an issue tracking system, or, more recently, publishing modified code using a distributed version control system. After identifying the method used by a project, individual change submission instances can be studied using the generic evaluation procedure.
Review changes submitted by the community	This task is complementary to the previous task, namely, changes submitted by community members must be reviewed and either rejected with an appropriate justification, or accepted and integrated into the project's main code repository. This task can be analyzed in a way similar to the previous task.

Task	***Description and Evidence Sources***
Promote actively contributing members of the community to committers	Community members who provide valuable contributions to the project over a period of time often receive rights to contribute directly to a project's code repository. Instances of this process can sometimes be seen on a project's development mailing lists.
Review changes by committers	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. This process can be evaluated by looking at the project's change log files or at the log messages written when committing changes to the code repository.
Propose significant enhancements	Some projects have disciplined processes that allow community members to formally propose enhancements for discussion by the community. Enhancement proposals may take many forms, including web pages, Wiki pages, and messages submitted to a mailing list or forum.
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. Except for very small projects, this task is normally supported by an issue tracking system, in which case process instances correspond to the reports in the system, as well as their accompanying discussions. Small projects may handle this through a mailing list, in which case instances are the messages reporting the problem and the discussions following it.
Test the program or programs 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. Test suites and defined test cases are normally part of the source code and can be found in the code repository. Instances of this process are test reports, either created automatically by running the test suite or manually.
Decide at which point in time a release will be made.	Either releases are done on a time-based fashion or based on a feature “road map”. Instances of any of these two documents can often be found as part of a project's web or Wiki pages, or, occasionally, as messages to a certain mailing list or forum.

Task	*Description and Evidence Sources*
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. This process can be followed by looking at release announcements for preliminary versions in a project's mailing lists or forums. Actual releases can be easily found in software download repositories.
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. Backports are often announced in project mailing lists or forums.

5 Initial Experience with the QualOSS Process Evaluation

To this date, our experience with the QualOSS process evaluation is still quite limited, since we have applied it to only a handful of projects so far. A larger number of full QualOSS OSS assessments—which include the process assessment—is planned for the final, evaluation phase of the QualOSS project. We expect this effort to result in significant adjustments to the process assessment framework, as we better understand its limitations and improve it accordingly.

Nonetheless, our current experience has already taught us some valuable lessons:

– In its current form, the QualOSS process evaluation can be applied to small to medium OSS projects in about six hours of work. This makes its costs reasonable for a number of purposes, including comparison when selecting between OSS alternatives. A caveat here is that, so far, evaluations have been conducted exclusively by an OSS and process expert. We still have to evaluate our approach when applied by other assessors who may lack this expertise. This includes, among other aspects, studying inter-rater reliability in this context.
– The time box limitation of 30 minutes of searching may lead to important information being missed. One alternative for handling the collection of information about a task would be to ask the community directly, for example, by writing to an appropriate mailing list. This would not only make this aspect of the process evaluation fairer, but would potentially create opportunities for the community to learn from the evaluation and improve based upon it.

- In some cases, the number of instances of a particular task is too high for manual inspection. For example, some projects have databases of reported issues that have been operating for years and contain thousands of reports. So far, we have analyzed such data repositories by manually choosing a small number of instances "at random", but this method is clearly unsatisfactory due to the high risk of introducing biases. Ideally, we should be able to guarantee that we did a fair, random sample, and that the number of instances observed is representative. We still have to do more research in appropriate methods for this purpose, and, potentially, provide software tools to assist this procedure.
- The importance of some of the tasks listed in the previous section may vary depending on the size of the evaluated project. For instance, many small OSS projects have a single maintainer who is the only person with access to the main versioning repository. Such projects will rarely, if ever, accept new permanent contributors, and thus having a defined process for this purpose would be simply unnecessary. On the other hand, large projects with tens or even hundreds of official developers definitely require an explicit process for accepting new members. For this reason, we are considering the idea of giving variable importance to different tasks depending on such characteristics of a project as its number of active contributors or its code size.

Future versions of the QualOSS process evaluation framework are likely to incorporate enhancements based on the previous observations.

6 Conclusions and Future Work

The purpose of the QualOSS project is to produce a comprehensive quality model for assessing OSS projects. In this paper, we have presented a small portion of this work, namely, a process evaluation framework aimed at OSS. We expect OSS process evaluation to provide a better foundation for judging a community's ability to deliver high-quality software, as well as its long-term sustainability ("will this project exist in 10 years?"). Indeed, sustainability of suppliers is critical to many stakeholders, and is also a problem with commercial software. For example, the European defense consortium EADS decided to turn a critical piece of software into OSS in order to become independent of specific suppliers [11].

Moreover, highly regulated industries, such as the automotive, medical, or pharmaceutical industries, have established standards for evaluating software, which include assessment of the supplier [3] [5]. These industries often find it problematic to use OSS, because there is little support for the assessments required by their quality standards. Consequently, we believe that OSS assessment models that include a process assessment may help to increase the adoption of OSS in these industries.

As mentioned in Section 5, our experience with applying the QualOSS process assessment is still very limited. The final, evaluation phase of the QualOSS project will provide us with a valuable opportunity to introduce some initial improvements—such as those suggested in Section 5—as well as to collect more experience with using the process evaluation framework. We expect this experience to allow us to produce a much more robust and reliable framework during the next few months.

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] Business Readiness Rating, `http://www.openbrr.org/` (last check March 9, 2009)

[2] Cap Gemini: OSS Partner Portal. Internet address, `http://www.osspartner.com/` (last check March 9, 2009)

[3] International Society for Pharmaceutical Engineering (ISPE): Good Automated Manufacturing Practice (GAMP-4) Supplier Guide for Validation of Automated Systems in Pharmaceutical Manufacture (1995)

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

[5] ISO/IEC 61508:1998, Functional safety of electrical/electronic/programmable electronic safety-related systems

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

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

[8] Navica Software Web Site, `http://www.navicasoft.com/` (last check March 9, 2009)

[9] Qualification and Selection of Open Source software (QSOS) Web Site, `http://www.qsos.org/` (last check March 9, 2009)

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

[11] TOPCASED: Toolkit in Open Source for Critical Applications & Systems Development, `http://www.topcased.org/` (last check March 13, 2009)