Measuring the Performance of Open Source Development Communities: The QualOSS Approach

Martín Soto[1], Daniel Izquierdo-Cortazar[2], Marcus Ciolkowski[1]

[1]Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany
[2]Universidad Rey Juan Carlos, Móstoles, Spain

soto@iese.fraunhofer.de, dizquierdo@libresoft.es, ciolkows@iese.fraunhofer.de

Abstract:

Free and Open Source Software (F/OSS) has an increasing importance for the software industry. Similar to traditional (closed) software acquisition, F/OSS acquisition requires an assessment of whether its quality is sufficient for the intended purpose, and of whether the chances of being maintained and supported in the future, as well as of keeping certain quality standards over time, are sufficiently high. The first one of these aspects is strictly product related, and can be assessed using techniques that are not specific to F/OSS. The last two aspects, however, are related to the community developing the software, and require novel approaches to be evaluated properly.

In this paper, we present an approach toward a comprehensive measurement framework for F/OSS projects, developed in the EU project QualOSS. Although this approach takes into account product quality as well as process maturity and sustainability of the underlying F/OSS community, we concentrate here on its community-related aspects. After describing our quality model and assessment techniques in some detail, we close with a description of our ongoing evaluation effort and a discussion of lessons learned.

Keywords

Open Source quality, F/OSS community, process assessment, community measures

1 Introduction

The strong, and ever increasing, influence of *Free and Open Source Software* (F/OSS) on industry can hardly be denied. The potentially large benefits offered by F/OSS, such as valuable functionality at relatively low cost and independence from a particular supplier, are undoubtedly appealing to decision makers in charge of industrial software acquisition. F/OSS adoption, however, does not come for free, even despite of the fact that it is free of licensing costs. 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. Making a wrong decision regarding the choice of F/OSS applications or components for a particular purpose may have serious negative consequences.

The EU Project QualOSS has worked on developing a model framework for evaluating F/OSS projects in order to support acquisition decisions. This framework was discussed in detail in a publication by two of the authors [1], presented at the MetriKon 2008 conference. The QualOSS Model is intended to be comprehensive, covering the evaluated product as well as the community that produces it. 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]

On the other hand, quality aspects related to the community that acts as supplier for a F/OSS software component cannot normally be evaluated using existing techniques. In the case of commercial software, supplier companies are typically evaluated using process assessments, such as those defined by the CMMI-DEV [3] or SPICE [4] standards. F/OSS communities, however, differ significantly from traditional companies, thus making it almost impossible to reasonably apply these standards to them. Fortunately, though, F/OSS projects usually maintain a wide variety of data repositories in the open, such as code releases, mailing lists, issue tracking systems, and versioning systems. By analyzing these repositories, it is possible to investigate many quality aspects of a F/OSS community in a systematic and reliable way.

The development and testing of appropriate tools and techniques for the analysis of F/OSS data repositories is currently a very active research subject. One of the main contributions of the QualOSS project is to put together a number of state-of-the-art analysis tools and techniques into a coherent framework that can be used to look at a F/OSS community from a variety of perspectives. For this reason, this article 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 (*evolvability*) and whether it will be able to consistently produce high quality software over time (*maturity*). In the following, we provide a general description of the QualOSS model, provide a more in-depth look into its community-related aspects, and discuss our ongoing effort for evaluating these aspects.

2 The QualOSS Model

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 concrete 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 assessment.

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.

Given our emphasis on covering not only F/OSS products but the communities behind them, we have organized the quality characteristics into two groups: those that relate to the product, and those that relate to the community. As explained above, the rest of this article provides more details about the community-related quality characteristics. For this reason, we provide here only a list of the product characteristics considered (Table 1).

Characteristic	**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 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.

Characteristic	Definition
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.

Table 1: Summary of QualOSS product-related quality characteristics

3 Community Measures in the QualOSS Model

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. We work on the basis of a twofold assumption regarding community quality. 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. These ideas lead to the quality characteristics summarized in Table 2.

Characteristic	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 (e.g., quality goals) by following established processes. Additionally, the level to which the

Characteristic	Definition
	processes followed by a development community are able to guarantee that certain desired product characteristics will be present in the product.

Table 2: Summary of the QualOSS community-related quality characteristics

The following subsections discuss these characteristics in more detail.

3.1 Maintenance Capacity and Sustainability

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. Also, a community requires members with the ability and willingness to perform all required maintenance tasks (e.g., programming, testing, documenting, bug triaging, managing releases, etc.) One aspect of maintenance capacity that is particularly relevant to potential users is product support. Oftentimes, companies trying to use a new F/OSS product are met by a general lack of responsiveness from the corresponding F/OSS community. Indeed, the capacity to react to user requests varies widely from one community to the next. This is often caused by the fact that many communities are composed mainly of volunteers, who have a limited amount of time to offer, and who may also restrict their participation to user requests that interest them personally.

Regarding sustainability, it is generally affected by factors such as the composition of the community, and its regeneration ability [6]. For example, 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 [7, 8] 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. With regard to regeneration ability, a community's past regeneration can be a good predictor of its future behavior in this respect. Past regeneration can be observed, for example, by analyzing contributions to the version management system or the mailing lists over time.

Data Sources

As mentioned above, F/OSS communities offer plenty of publicly available data sources which can be analyzed in order to better understand how they work and

evolve over time [9]. Source code management systems, mailing lists, and issue tracking systems are the most important of these data sources, but Wikis, forums and Web-logs ("blogs") can also provide useful information.

QualOSS focuses on the first three data sources mentioned:

- *Source code management systems.* These systems store information about each modification in the source code carried on by the software developers. It means that it is possible to find out when modifications took place, what was modified in each case, and by whom. This is key to understanding how the source code evolves, but also to studying how the community evolves and regenerates, to checking the status of junior and senior developers, and so on. The source code management systems most commonly used in F/OSS development are CVS, Subversion, and Git.
- *Mailing lists.* Mailing lists are central to the organization of F/OSS projects. In a distributed environment, it is absolutely necessary to have a common way of communication. Mailing lists and forums normally provide this infrastructure. QualOSS offers tools for analyzing lists based on the very common GNU Mailman mailing list administration software, which makes archives of the mailing list discussions available in the Internet RFC 822 format. This format provides detailed information about the time and origin of every message, and allows for determining the place each message occupies in the general discussion.
- *Issue tracking systems.* Also known as bug tracking systems (BTS), these systems provide an environment where developers can centralize the defect fixing process. They are also one of the main communication channels between the project users, who report about problems the may have encountered, and the main developers. In addition to allowing for reporting defects, the BTS offers the possibility of adding comments or attachments (such as a test case) to an existing report, and of subscribing to events for a given report. The best known BTS used in F/OSS development are Bugzilla, GNATS, Mantis, Jira, and the SourceForge tracker.

Metric Definition

Metric definition in QualOSS was based loosely on the the Goal-Question-Metric paradigm (GQM). Based on the goals of characterizing evolvability and robustness of a community, a number of specific questions were posed. The following is an excerpt of the complete question list found in QualOSS Deliverable 4.1 [10]. It refers to the quality characteristic "Community size and Regeneration Adequacy":

- Has the evolution of new community members reporting bugs remained stable or grown over the history of the FLOSS endeavor? (at least shown a stable or positive trend overall)

- Has the evolution of new code contributing members remained stable or grown over the history of the FLOSS endeavor? (at least shown a stable or positive trend overall)
- Has the evolution of new members contributing data other than code or bug report remained stable or grown over the history of the FLOSS endeavor? (at least shown a stable or positive trend overall)
- Has the evolution of new core contributing members remained stable or grown over the history of the FLOSS endeavor? (at least shown a stable or positive trend overall) (a core member is one with commit right who perform commits frequently for instance, more than once every three month period)
- What is the evolution of core members who stopped contributing for a significant period?
- Has the evolution of core members who stopped contributing for a significant period been compensated by the joining of new core members around the same time frame?
- What is the average longevity of committers to the FLOSS endeavor?

As indicated by the Goal-Question-Metric paradigm, each one of the listed questions was addressed by defining specific metrics. Most of the basic QualOSS metrics related to maintenance capacity and sustainability can be calculated automatically by retrieving data from the aforementioned data repositories. The usual procedure is to apply a set of tools which access the repositories in order to "mine" for relevant pieces of data. These data are later stored in a local database, using a representation that is amenable to our analysis procedures [9] .

In order to illustrate how metrics were defined in this context, we will use the second question above ("Has the evolution of new code contributing members remained stable or grown over the project history?") as an example. This question is addressed by looking at the logs stored by the code management system in order to determine when each project developer made his first code contribution (that is, when he committed code to the system for the first time). These values are them bucketed into appropriate time intervals (one, three or six months depending on the history time span) and a linear regression analysis is conducted on them in order to fit a straight line to the resulting data set.

The slope of this line provides a general idea of how the arrival of new community contributors has evolved over time. A negative slope indicates that the number of new committers has generally decreased during the observed period. A positive slope, on the other hand, indicates an overall tendency to grow.

Unfortunately, providing just the slope value as an answer is not enough for the purposes of the QualOSS quality model, because only experts would be able to determine what a given value means for the a particular project. For this reason, and in order to provide an easy-to-use indicator, it is necessary to determine the

value ranges that correspond to a generally "good" new committer evolution, as well as those that probably indicate that the community is having trouble acquiring new contributing members. In order to do this, we started with a naive approach that defined a positive slope as "Green", a negative slope as "Red", and a flat slope as "Yellow". This approach, however, is too simplistic to reflect the dynamics of actual F/OSS communities.

We are currently in the process of determining a more realistic set of thresholds for this and other similar metrics by using data from a large number of F/OSS projects compiled by the EU project FLOSSMetrics [11]. Since these data was collected for the most part using the same tools used by QualOSS, it makes it possible to evaluate the QualOSS metrics on a large number of projects in order to define thresholds based on a feasible statistical basis.

3.2 Process Maturity

As mentioned in the introduction, existing process assessment models cannot generally be applied directly to F/OSS development, as they include too many elements that are specific to companies and other conventional development organizations. Still, the idea of assessing a F/OSS community in order to determine which good practices it follows, as well as how established these practices are, is perfectly reasonable. In this subsection, we describe our process evaluation framework, which is directly aimed at F/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 a F/OSS context.

Maturity Models as a Basis for F/OSS Process Assessment

In order to create an assessment model for F/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 F/OSS. Concretely, we used the *Capability Maturity Model for Software Development* (CMMI-DEV) as a starting point. Given how comprehensive CMMI-DEV is, reaching its highest capability levels represents a serious challenge for any software development organization. Clearly, F/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, F/OSS communities are still quite far from reaching the levels of process discipline required by the highest 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 F/OSS communities, and with regard to different areas of the software development process. We think that many of these practices correspond to the spirit, although often not directly to the letter, of the practices and goals specified by CMMI-DEV.

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

- *Version/Configuration Management:* As already discussed, many F/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 management 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 F/OSS communities. This led us to believe that, despite the inviability of applying a full-fledged process maturity model to F/OSS, a process evaluation model for F/OSS is not only viable, but potentially very useful in order to gauge the ability of F/OSS communities to consistently deliver software of appropriate quality. This belief constitutes the main motivation for the QualOSS process evaluation framework.

The Generic QualOSS Process Evaluation

In its current form, our F/OSS 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 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 data repositories that belong to a project.

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 potentially 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 F/OSS communities usually are.

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 a F/OSS project. The following list presents the tasks that are currently covered by the QualOSS process evaluation. This is just an initial selection of tasks, which we are likely to extend as we gain experience with the process evaluation:

- Change submission.
- Review changes submitted by the community.
- Promote actively contributing members of the community to committers.
- Review changes by committers.
- Propose significant enhancements.
- Report and handle issues with the product.
- Test the program or programs produced by the project.
- Decide at which point in time a release will be made.
- Release new versions of the product.
- Backport corrections in the current release to previous stable releases.

4 Evaluation and Calibration of the QualOSS Model

At the time of this writing (September 2009) we are conducting the final evaluation of the QualOSS model. The evaluation targets up to 15 F/OSS projects, which will be assessed by applying the QualOSS model (partially or completely) to them. 10 of these assessments are already completed. Projects selected for the evaluation include projects that, according to expert judgement, 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. On the other hand, measures such as those related to process maturity, require a significant amount of manual effort. Current results indicate that performing the full QualOSS requires less than five person-days for medium-sized projects. It is important to note, however, that a complete assessment normally involves the participation of several assessors, because the various measurement procedures applied require expertise in widely different areas. The coordination overhead may significantly increase the overall effort, even if every single assessor only contributes a few hours of work.

Our experience with the QualOSS Model has already taught us several valuable lessons. In the following, we discuss some of the most prominent ones:

- It is often the case that, for a given project, some of the data sources are not available, or cannot be analyzed. Among other reasons, projects may have not archived certain types of data, or may have decided not to publish certain types of data because of, for example, privacy or security concerns. Also, data may be only available in formats, or through on-line interfaces that our tools cannot handle. We are currently trying to devise an improved way to present our results that makes it clear that certain metrics and indicators were not computed because of missing data, or that they were computed on the basis of incomplete data.
- In addition to the problems caused by missing data sources, it can also happen that certain project processes difficult data analysis. For example, in certain projects, a small group of code maintainers review code submissions made by external contributors, and commit them manually to the version management system. Since our analysis tools are not able to identify this case, the analysis results will show the direct maintainers as the only project contributors, each of them sporting an inordinately large productivity, while, in reality, they are just channeling the contributions of a much larger community. Unfortunately, limitations of these type can only be handled by providing specific tools or measurement procedures for the situation at hand.
- Some of our indicators yielded bad marks far all projects already evaluated. This probably means that our model entails too high expectations in some particular dimensions that, it at all, will only be fulfilled by a very small percentage of actual projects. We are currently considering to remove some of these

characteristics from the model, because they are not helpful to discriminate among F/OSS projects. In some cases, however, we have found projects outside our evaluation set that actually fare well regarding these quality aspects. In such cases, the characteristics are more likely to remain in the model.

- When evaluating processes, the number of instances of a particular task may be too high for manual inspection. For example, some projects have databases of reported issues that have been operating for several 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 process section may vary depending on the size of the evaluated project. For instance, many small F/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 these and other similar observations.

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 Gregorio Robles and Jesús María Gonzalez-Barahona: Contributor turnover in libre software projects. In Proceedings of the Second International Conference on Open Source Systems, Como, Italy, July, 2006.

7 M. Michlmayr: Managing volunteer activity in free software projects. In Proceedings of the USENIX 2004 Annual Technical Conference, FREENIX Track, pages 93-102, Bostson, USA, 2004.

8 Daniel Germán: The GNOME project: a case study of open source, global software development. Journal of Software Process: Improvements and Practice, 8(4):201-215, 2004.

9 Gregorio Robles, Jesús María Gonzalez-Barahona, Daniel Izquierdo-Cortazar and Israel Herraiz: Tools for the Study of the Usual Data Sources found in Libre Software Projects. International Journal on Open Source Software and Processes. 2008.

10 Jean-Christophe Deprez, Kirsten Haaland and Flora Kamseu: QualOSS Methodology and QUALOSS assessment methods. QualOSS project Deliverable 4.1.

11 FLOSSMetrics project database. Available from http://melquiades.flossmetrics.org (last checked 2009-09-21).