The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation

Ioannis Samoladas[1], Georgios Gousios[2],
Diomidis Spinellis[2], and Ioannis Stamelos[1]
[1] Department of Informatics, Aristotle University of Thessaloniki,
541 24, Thessaloniki, Greece
{ioansam, stamelos}@csd.auth.gr
[2] Department of Management Science and Technology,
Athens University of Economics and Business, 104 34, Athens, Greece
{gousiosg, dds}@aueb.gr

Abstract. Software quality evaluation has always been an important part of software business. The quality evaluation process is usually based on hierarchical quality models that measure various aspects of software quality and deduce a characterization of the product quality being evaluated. The particular nature of open source software has rendered existing models inappropriate for detailed quality evaluations. In this paper, we present a hierarchical quality model that evaluates source code and community processes, based on automatic calculation of metric values and their correlation to a set of predefined quality profiles.[1]

Keywords: Quality models, automated measurement, software metrics

1 Introduction

One of the main concerns of software engineering is the production of high quality software systems and thus software quality evaluation has always been a critical task for software professionals. IT managers often face the problem of evaluating software in order to decide whether it is suitable for their needs. Additionally, software houses perform evaluations on the software they develop to decide whether it has matured enough to be deployed. Evaluations are based on software models that define and measure software quality, usually by combining software metrics and experts' opinions. The advent of free, libre and open source software (OSS) has rendered the traditional quality evaluation models non applicable to some extent, as

[1] This work was partially supported by the European Community's Sixth Framework Programme under the contract IST-2005-033331 "Software Quality Observatory for Open Source Software (SQO-OSS)".

Please use the following format when citing this chapter:

Samoladas, I., Gousios, G., Spinellis, D. and Stamelos, I., 2008, in IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*; Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, Giancarlo Succi; (Boston: Springer), pp. 237–248.

they cannot be tuned to reflect OSS development practices and therefore cannot be used to evaluate both the software and the community as a whole.

In this paper, we present a novel software quality evaluation model, specifically targeted to OSS. The SQO-OSS model was constructed to support an automated software evaluation system; its variables are mainly metric-oriented while human intervention is minimal. Additionally, our model evaluates all aspects of OSS development, both the product (code) and the community. The evaluation weights and criteria can be tuned by the evaluator, while a set of predefined profiles that cover basic evaluation cases are offered.

The remainder of the paper is organized as follows: In Section 2, we present related work in the area of both traditional and open source software quality evaluation; in Section 3, we present the SQO-OSS quality model definition and the evaluation process. Section 4 presents an application of a part of the model on three example open source projects. The paper concludes with a description of the Alitheia system, the host system for software metric calculations, as well as our plans for future work.

2 Related Work

Since researchers started investigating the issue of quality in software systems, they employed specific models to express it. Models usually decompose quality into an hierarchy of criteria and attributes[2]. These hierarchical models lead to metrics at their lowest level. Metrics are directly measurable attributes of software and they are used to express certain aspects of the product that affect quality [1]. Examples of traditional software quality models are the McCall and Boehm's models [1], the more widely accepted ISO/IEC 9126 model [2], and its more recent implementation by SQuaRE, the ISO 25000:2005 [3].

The adoption of OSS in many organizations has raised the issue of OSS quality evaluation. Due to the nature of OSS development where standard practices include open access to the source code, shared artifact repositories, peer review of committed code, asynchronous global development and lack of formal support, traditional software quality models may not be sufficient. An array of quality models specifically targeted to OSS development can be found in the literature, but most of them are either purpose specific or require significant human intervention,.

The OSMM [4] model assumes that the quality of an open source project proportional to its maturity. The latter is decomposed into six constituents (*Product software, Support, Documentation, Training, Product integrations* and *Professional services*), each one having some weight. The evaluator assigns a score to each element and the final evaluation mark is the weighted sum of the scores. Although

[2] Throughout the paper the terms criterion and attribute (sub-criterion and sub-attribute) are used interchangeably.

OSMM is simple and thus easy to apply, it is often criticized for not taking into account some important software artifacts, such as the source code itself.

The Open Business Readiness Rating (OpenBRR) [5] defines a model and a process for evaluating OSS, with particular emphasis on attributes interesting to businesses. OpenBRR uses a variety of high-level criteria for evaluation, such as functionality, operational software characteristics, support and service and adoption and development process. The assessment process involves defining a reference application and through it identifying a set of characteristics (and weights for them) that are desirable in the evaluated applications. The evaluation result is extracted by assigning grades to each characteristic and averaging the results from the evaluators. While the OpenBRR method is a step forward from OSMM through the inclusion of the community process in the evaluation, the notion of the reference application has been criticized as a major drawback. Furthermore, the evaluation itself is highly subjective, while the overall process seems complicated, offering very little prospect for automation.

The Qualification and Selection of Open Source Software (QSOS) [6] is another open source evaluation model. The evaluation process is done in four iterative phases. Phase one is the definition of the evaluation factors. The second phase involves the collection of information from the open source community and the construction of an identity card for the evaluated software. The quality criteria are then scored in a range from zero to two according to specific guidelines provided by the methodology. Phase three is the definition of the selection criteria according to user's needs and constraints. The last phase is the identification of the software that fulfills user requirements and more generally compares software from the same family. Like OpenBRR, QSOS offers a tool that supports the evaluation process. Although QSOS scoring guidelines allow for objective results among users, the whole process is not flexible enough and difficult to handle.

The SQO-OSS quality model distinguishes from existing open source quality models in various ways:

1. The SQO-OSS model was constructed with a focus to automation, while the rest of the models require heavy user interference and lack automation of metrics collection.

2. The SQO-OSS model is the core of a continuous quality monitoring system and automatic metrics collection guarantee that assessments are made with relatively recent data.

3. The SQO-OSS model does not evaluate functionality. Functionality assessment requires the evaluator to play an important role in the assessment process and thus introduces subjectivity. The SQO-OSS model focuses on fundamental aspects of OSS quality, namely OSS project maintainability, reliability and security.

4. The SQO-OSS model focuses on source code. Source code is the single most important product of a software development project and its quality must play a significant role in determining the final assessment of the product.

5. The SQO-OSS model also considers the open source community. However, it takes into account only those community factors that can be measured automatically.

6. As the evaluation must necessarily take into account the evaluator's point of view, we allow the user to intervene in the measurement-based evaluation process by modifying category profiles.

3 The SQO-OSS Quality Model

We can generally assume that an evaluation process can be divided into two phases, the definition of the evaluation model and the definition of the measurement process. In our case, each phase includes two distinct steps:

Phase One: *Definition of the evaluation model*

1. Definition of the model criteria (attributes and sub-attributes).

2. Definition of metrics.

Phase Two: *Definition of the aggregation method*

1. Definition of the evaluation categories

2. Definition of the profiles of those categories

Phase one represents the work done in order to define the evaluation framework, applying our notion of quality, the definition of the quality criteria and the metrics that measure these criteria. The second phase represents the data collection part and the aggregation of the measurement results in order to reach an outcome for the quality of the artifact under evaluation. At this point we have to lay more emphasis on the fact that throughout this process, automation was the main concern. We wanted a model that can be applied automatically and on a fair amount of projects, fed continuously with data from the SQO-OSS observatory. Thus we tried to focus on quality attributes that can be measured with minimum human interference. For example usability involves extensive human interaction so it was not chosen as a quality attribute for our model. The same approach was followed for metrics selection.

3.1 Model definition

Prior to starting the construction of our model, we took into account that the quality and the health of an OSS project depends on the quality of its source code base and that of the community built around it. In order to measure these two aspects of quality and construct the SQO-OSS quality model we used a simplified version of the Goal-Question-Metric (GQM) process [7].

For the first part, we formulated our first goal, "*analyze the source code of an open source project*". This analysis has to be done in order to characterize code quality with respect to its maintainability, reliability and security. So, for this goal we formulated the question *"How is source code quality measured?"*. We then formulated the question as *"How is maintainability, reliability and security measures?"* In turn, each one of these aspects of source code quality is a small goal itself with its own questions. We kept on formulating questions iteratively until we reached a level where an attribute can be measured using straightforward metrics, i.e. without any usage of compound metrics.

For example, in order to measure *Maintainability*, we chose to follow its definition in the ISO/IEC 9126 quality model: *Analyzability*, *Changeability*, *Stability*, *Testability*. To measure each one of these sub-attributes we used direct source code metrics. In order to differentiate between structured and object oriented programming we used different definitions of *Maintainability* attribute for these two programming paradigms. Metrics selection was a difficult part, since software engineering metrics research is a very active topic and many researchers express their concerns on various metrics. For our own model we chose to select only widely acceptable metrics and metrics that have been validated extensively [8]. For an extensive review on metrics please refer to [9] and [1].

In a similar fashion, we constructed a hierarchical view of our quality model as a tree. The root represents the overall quality model, the next two nodes the source code and community quality, while the leaves represent the metrics. After constructing the initial model, we uploaded the model on the wiki page of our project asking from our consortium partners to review it and comment on it. Our partners come from the OSS community (developers, users), academia and companies specializing on OSS development and support. Additionally, we also asked them to change the model (both model leaves and respective metrics) and justify their opinion. The only thing stressed to the partners was the importance of automatic metric collection. The history facility of the wiki allowed us to review the changes, discuss them with the partners and finalize the model. A tree view of the model is presented in Figure 1, while the metrics selected for the evaluation of the selected criteria (the leaves on the tree view) can be seen in Table 1. At this point, we should mention that our system allows partial evaluation of a product, i.e. evaluation of a single attribute like *Testability*. Thus, we have used the same metrics in more than one attribute.

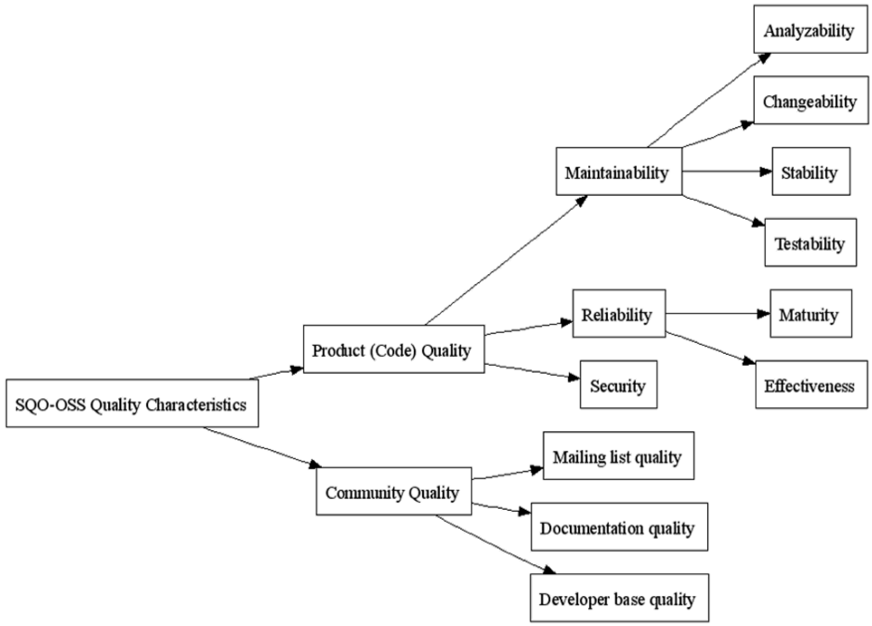

Fig. 1. The SQO-OSS quality model.

Table 1. Metrics for criteria of Product (Code) Quality and Community Quality

Attribute	**Metric**
Analyzability	Cyclomatic Number
	Number of statements
	Comments frequency
	Average size of statements
	Weighted methods per class (WMC)
	Number of base classes
	Class comments frequency
Changeability	Average size of statements
	Vocabulary frequency
	Number of unconditional jumps
	Number of nested levels
	Coupling between objects (CBO)
	Lack of cohesion (LCOM)
	Depth of inheritance tree (DIT)
Stability	Number of unconditional jumps
	Number of entry nodes
	Number of exit nodes

	Directly called components
	Number of children (NOC)
	Coupling between objects (CBO)
	Depth of inheritance tree (DIT)
Testability	Number of exits of conditional structs
	Cyclomatic number
	Number of nested levels
	Number of unconditional jumps
	Response for a class (RFC)
	Average cyclomatic complexity per method
	Number of children (NOC)
Maturity	Number of open critical bugs in the last 6 months
	Number of open bugs in the last six months
Effectiveness	Number of critical bugs fixed in the last 6
	Months
	Number of bugs fixed in the last 6 months
Security	Null dereferences
	Undefined values
Mailing list	Number of unique subscribers
	Number of messages in user/support list per month
	Number of messages in developers list per month
	Average thread depth
Documentation	Available documentation documents
	Update frequency
Developer base	Rate of developer intake
	Rate of developer turnover
	Growth in active developers

3.2 Evaluation Process

In order to evaluate the quality of an open source project, we have to combine all these measurements in one single view to obtain an evaluation result, i.e. we have to aggregate the measurements. For this, we used the profile-based evaluation method described in detail in reference [10]. Most evaluation methods presented in the literature use a weighted average sum function as their aggregation method. A drawback for applying this aggregation method on our model is that it uses measures with interval scales, while our goal was to provide results in an ordinal scale (such as *good*, *fair* or *poor*). The method we selected allows us to combine all kind of measurements, based on either ordinal or interval scales.

In order to apply an ordinal scaling aggregation method, we must first decide how many categories of evaluation ranking are required. In reference [10], Morisio et al. discuss that the ideal number of evaluation categories is between three and five. Based on that, for our model, we used four categories: **Excellent** (*E*), **Good** (*G*), **Fair** (*F*) and **Poor** (*P*) (or as an ordinal scale *E*>*G*>*F*>*P*). Having four categories, the aggregation method requires the definition of three profiles, each one for the first three categories (*E, G* and *F*). If an artifact cannot be fitted into one of these three categories, then it is automatically categorized as poor. The profiles represent the least measurement values required for each category and they are defined separately for each composed criterion of the model. Then each criterion is further decomposed into its sub-criteria (hierarchically, according to the quality model) and each decomposed criterion has its own profile. For the quality model leaves, which consist of an array metrics used to assess their parent criterion, we use a vector of numbers that is constructed by the application of each specific metric to the assessed artifact. We used the thresholds indicated by the literature [11, 12, 13] to correlate the measurement value vectors to the profiles we had developed (see Table 2).

Decomposing root attributes of our quality model into more fine-grained criteria and then down to metrics entails that profiles correspond to each decomposed criterion. Thus, in order to characterize the product quality of a product as "Excellent", Maintainability, Reliability and Security must be also characterized as "Excellent". Table 2 shows that a software component that scores "*Excellent*" in Maintainability must score *"Excellent"* in the Analyzability sub-attribute, too. This, in turn, means that the corresponding metrics applied to the evaluated component must return values equivalent to or higher than the vector [4,10,0.5,2]. Similar profiles correlating metric values to profiles can be applied to the rest of the criteria. Due to space limitations, Table 2 shows only the metrics values for Analyzability and Changeability; however, similar thresholds exist for the rest of criteria. Users of the model can modify the profiles according to their needs, e.g. a security aware user may define higher default values for each of the profiles. Additionally, the method allows usage of weights on the various metrics, but such practice is not recommended, as we assume that all metrics are of equal importance.

The aggregation process is done with the use of specific outranking relations iteratively with all the given profiles. The outranking relations express our decision of comparing the artifact with the profiles. Thus, an artifact *x* is considered to be *at least as good as* the *y* profile if and only if the "weighted" majority of the criteria agree so. If a set of tests agree, which represent the strength to be reached in order for an artifact to be categorized in a category *A* then *x* is assigned to *A*. The method also allows for two kinds of assignments in categories. The first is the pessimistic assignment representing the *at least as good as* relation (project *x* is *at least as good as* profile *y*). The second is the optimistic assignment, which identifies the profile which is surely worse than *x* and assigns *x* to the previous one (for example if *x* is strictly worse than *E* then it is assigned to *G*). If the two assignments coincide, then we are sure about our decision, otherwise it is the evaluator's decision which of the two assignments will be adopted. The mathematical foundations and the actual procedure of the aggregation process is presented in reference [10]. The method used here is different than Analytic Hierarchy Process, a widely used aggregation method, which requires ratio scales on all measures, a requirement that is not fulfilled in our case. Moreover, as already mentioned earlier, having a weighted average sum function as an aggregation method forces us to use measures with interval scales; our goal was to provide results on an ordinal scale, a feature that is provided with the method presented.

An example of the aggregation process is presented in the next section.

Table 2. Profiles for criteria Analyzability, Changeability, Stability and Testability

Composed Criterion	Criterion	Profile E	Profile G	Profile F	Scale
Analyzability	Cyclomatic number	4	6	8	Less is better
	Number of statements	10	25	50	Less is better
	Comments frequency	0.5	0.3	0.1	More is better
	Average size of statements	2	3	4	Less is better
Changeability	Average size of statements	2	3	4	Less is better
	Vocabulary frequency	4	7	10	Less is better
	Number of unconditional jumps	0	0	1	Less is better
	Number of nested levels	1	3	5	Less is better
Stability	Number of unconditional jumps	0	0	1	Less is better
	Number of entry nodes	1	2	3	Less is better
	Number of exit nodes	1	1	1	One is better
	Directly called components	2	5	7	Less is better
Testability	Number of exits of conditional structs	0	1	4	Less is better
	Cyclomatic Number	4	6	8	Less is better
	Number of nested levels	1	3	5	Less is better
	Number of unconditional jumps	0	0	1	Less is better

4 Evaluation Example

In this section, we present an example of application of the SQO-OSS quality model. For our example, we evaluated the source code of the CVS versioning system, the interpreter of the Perl programming language and the C files of the FreeBSD operating system. Table 5 shows the performance of these projects according to the various measurements. All measurements were performed using the metrics extracted from the application of the CScout refactoring browser [14] on the evaluated projects. According to the measurements, two out of our three projects scored well in the maintainability criterion. A direct interpretation of these results according to the proposed model is that CVS and FreeBSD are *at least as good as* the metrics thresholds for the **Good** profile, while Perl is *at least as good as* the thresholds for **Fair**.

A careful examination of the results reveals details of the performance of these projects in various sub-attributes of maintainability. All three projects achieve high marks in the changeability metrics and a good level of analyzability, perhaps due to their development model. Stability and testability are fair, but close to the metric thresholds we set for each respective criterion. Taking into consideration that the thresholds used in our example are relatively strict, the results are encouraging even for these two factors. Apart from providing predefined thresholds, our method has the benefit of presenting the results in a way that enables the developer to focus on measurements that are really interesting to him and also to receive both coarse and fine grained information.

Table 5. Example measurements of projects CVS, Perl and FreeBSD

Composed Criterion	**Project evaluation**		
	CVS	**Perl**	**FreeBSD**
Analyzability	[5.63, 37.8, 0.34, 1.89]	[3.0, 36.17, 0.08, 1.02]	[3.82, 29.99, 0.12, 1.99]
Evaluation	*Good*	*Fair*	*Excellent*
Changeability	[1.89, 2.91, 0.24, 1.13]	[1.02, 2.42, 0.28, 0.53]	[1.99, 2.87, 0.51, 0.86]
Evaluation	*Good*	*Excellent*	*Excellent*
Stability	[0.24, 3.28, 1.08, 4.49]	[0.08, 3.21, 0.78, 4.75]	[0.25, 3.99, 1.23, 4.10]
Evaluation	*Poor*	*Fair*	*Poor*
Testability	[0.57, 5.62, 1.13, 0.24]	[0.46, 3.0, 0.53, 0.08]	[0.55, 3.82, 0.86, 0.25]
Evaluation	*Good*	*Good*	*Good*
Maintainability	**Good**	**Fair**	**Good**

5 The Alitheia System

The quality model presented above serves as an automated decision support tool, integrated into the SQO-OSS system [15]. The SQO-OSS project aims to build a software quality observatory for OSS. For that purpose, we have developed Alitheia, a quality evaluation tool, and a web site with the results of the tool application on various OSS projects. The user is able to browse the product and process quality characteristics of the evaluated projects. The SQO-OSS quality model assists the user by incorporating the individual measurements in a comprehensive set of predefined quality profiles.

The Alitheia platform is an OSGi-based tool, targeted to the evaluation of software quality. It consists of a set of core services, such as accessors to project artifacts, job managers and relational data storage, and it is extensible through the use of plug-ins. Plug-ins can either implement basic software metrics or combine the results from other plug-ins arbitrarily. In fact, the quality model is a compound plug-in in Alitheia. The system allows full automation of the quality evaluation process after the initial project registration. The core communicates to the world through a web services interface. Clients being developed include the aforementioned web site and an Eclipse plug-in.

6 Conclusion and Future Work

In this paper we presented a new open source software quality evaluation model. The model was constructed for use in the Alitheia system, as a measurement-based decision support system, therefore automation was one of the first priorities while constructing the model. Previous models developed for OSS evaluation require a substantial effort from the user regarding the rating of the software under evaluation, while the model presented here asks for limited user interaction. Apart from the model itself, the evaluation process facilitates a profile based evaluation algorithm that is different from the traditional weighted aggregation that most of the models use. The profiles used for evaluation can be altered by the evaluator if he decides it is needed so.

Our immediate plans are to empirically validate our model. In order to test our model we are collecting measurements from an array of OSS projects. To meet our goal, we are going to perform a user based validation. Our project consortium includes members of a large OSS project, namely KDE Desktop Environment. Partners from the KDE project will evaluate software against our model and their opinions will be tested against the results or our evaluation process. In addition we want to evaluate its predictability and accuracy regarding its ability to classify software according to its quality. These tests will also allow us to calibrate the threshold values of our profiles. Moreover, we will work towards testing the relationships between metrics and categories and try to identify trends between aspects of quality and metrics.

References

1. Norman Fenton and Shari Lawrence Pfleeger. *Software Metrics - A Rigorous Approach*. International Thomson Publishing, London, 1997.
2. ISO. Software engineering - Product quality - Part 1: Quality model, ISO/IEC 9126-1:2001. ISO Geneva, 2001.
3. ISO. Software engineering Software product Quality Requirements and Evaluation (SQuaRE) Guide to SQuaRE. ISO Geneva, 2005.
4. Bernard Golden. *Making Open Source Ready for the Enterprise, The Open Source Maturity Model*. Extracted From Succeeding with Open Source, Addison-Wesley Publishing Company, 2005.
5. Business Readiness Rating. Business readiness rating for open source. http://www.openbrr.org.
6. QSOS. Method for qualification and selection of open source software (qsos) version 1.6. http://www.qsos.org.

7. R. Van Solingen. The goal/question/metric approach. *Encyclopedia of Software Engineering*, 2:578-583, 2002.
8. Tibor Gyimothy, Rudolf Ferenc, and Istvan Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897-910, 2005.
9. Stephen H. Kan. Metrics and Models in Software Quality Engineering, 2nd Edition. Addison-Wesley Publishing Company, 2003.
10. Maurizio Morisio, Ioannis Stamelos, and Alexis Tsoukias. Software product and process assessment through profile-based evaluation. International Journal of *Software Engineering and Knowledge Engineering*, 13(5):495-512, 2003.
11. NASA and SATC. Recommended thresholds for non-oo languages. http://satc.gsfc.nasa.gov/metrics/codemetrics/non_oo/thresholds/index.html.
12. NASA and SATC. Recommended thresholds for oo languages. http://satc.gsfc.nasa.gov/metrics/codemetrics/oo/thresholds/index.html.
13. Saida Benlarbi, Khaled El Emam, Nishith Goel, and Shesh Rai. Thresholds for object-oriented measures. In *Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, 2000.
14. Diomidis Spinellis. Global analysis and transformations in preprocessed languages. *IEEE Transactions on Software Engineering*, 29(11):1019-1030, November 2003.
15. Georgios Gousios, Vassilios Karakoidas, Konstantinos Stroggylos, Panagiotis Louridas, Vasileios Vlachos, and Diomidis Spinellis. Software quality assesment of open source software. In *Proceedings of the 11th Panhellenic Conference on Informatics*, May 2007.