

# Call for Quality: Open Source Software Quality Observation

Adriaan de Groot<sup>1</sup>, Sebastian Kügler<sup>1</sup>, Paul J. Adams<sup>2</sup>, and Giorgos Gousios<sup>3</sup>

<sup>1</sup> Quality Team, KDE e.V. {groot,sebas}@kde.org

<sup>2</sup> University of Lincoln padams@lincoln.ac.uk

<sup>3</sup> Athens University of Economics and Business gousiosg@aueb.gr

**Abstract.** This paper describes how a Software Quality Observatory works to evaluate and quantify the quality of an Open Source project. Such a quality measurement can be used by organizations intending to deploy an Open Source solution to pick one of the available projects for use. We offer a case description of how the Software Quality Observatory will be applied to the KDE project to document and evaluate its quality practices for outsiders.

## Keywords

Open Source software, software quality evaluation, static code analysis

## 1 Introduction

The software development process is well known as a contributor to software product quality, leading to application of software process improvement as a key technique for the overall improvement of the software product. This can be said for any form of software development. Within the Open Source paradigm, the leverage of software quality data can be as useful for the end users as it is for the developers.

From the perspective of a potential *user* of a piece of Open Source software (OSS), it can be very difficult to choose one of a myriad solutions to a given problem. There are often dozens of Open Source solutions which “compete” for users and development resources. They may differ in quality, features, requirements, etc. By making the quality aspects of a given project explicit, it becomes easier for the user to choose a solution based on the quality of the software. Here the Software Quality Observatory (SQO) can play a useful role in quantifying the quality of processes employed by a given OSS project.

With ever increasing numbers of projects and developers on SourceForge ([www.sourceforge.net](http://www.sourceforge.net)), it is clear that the OSS paradigm is of interest to those wishing to contribute to the creation of software. By using scientifically obtained software quality data, such as that which the Software Quality Observatory will produce, it may be possible to encourage similar growth within the OSS user community.

## 2 The Benefits of Software Quality Observation

As participation has grown in Open Source development over the past decade, so too has the user base of the software grown. Increasingly OSS is being viewed as a viable alternative to proprietary (closed source) software, not just by technically-aware developers, but also by non-developers. European research projects, such as COSPA ([www.cospa-project.org/](http://www.cospa-project.org/)) and CALIBRE ([www.calibre.ie](http://www.calibre.ie)), have raised awareness of OSS development through specific targeting of public administration bodies and industrial organisations, especially small and medium enterprises (SMEs).

As the OSS paradigm makes progress within these organisations any potential software procurer is tasked with some important questions which, currently, cannot be answered with any real assurance:

- Many OSS projects are very similar. How do we choose between them? Which is the most appropriate system for the company’s IT infrastructure?
- How can we distinguish the “good” and “bad” projects?
- How can we reason about the quality of a software product in order to trust its future development?

Unfortunately these organisations often have nothing more than word-of-mouth on which to base their judgments of OSS products. With 109,707<sup>1</sup> projects currently hosted on SourceForge it is understandable that products of excellent quality may be overlooked. It is possible to supplement the word-of-mouth tradition with some rudimentary data that is available from hosting sites: download numbers, project activity etc. Unfortunately this data is easily skewed and can present a product in an inaccurate manner.

Quality can be a very subjective measure of many aspects of a system in combination: suitability for purpose, reliability, aesthetic etc. Software quality is formally defined by the ISO/IEC 9126 standard as comprised of six characteristics, but no measurement techniques are defined. It has been suggested that the external quality characteristics of a software system are directly related to its internal quality characteristics. It is therefore possible to evaluate the quality of software through its source code and a of project by considering other data sources intimately related to the project’s code such as bug-fix databases or mailing lists.

In the long run it is crucial to OSS developers and their projects to know *quantitatively* what the quality of their product is. The volunteer nature of OSS makes “managing” such a project to include quality control a matter of motivating volunteers to behave in ways consistent with improving quality[2]. By fully understanding their software quality, OSS developers are able to promote and improve their products and process. It is also crucial in helping end-users making informed decisions about software procurement.

<sup>1</sup> Data from the FLOSSMole Project, 02/12/05.

### 3 Why SQO of Open Source Software differs from that on Closed Software

There are two aspects that play a role for quality assessment of software, the quality of the product itself and the quality of the product team. The main differences between quality assessment (QA) of Open Source software and QA of closed source software naturally relate to the availability of the source code and the transparency of the development process. Third party quality assessment is facilitated by the availability of the source code and the openness of the development process.

Quality assessment of OSS software is usually much more transparent than that of closed source software, at least to quality observers on the “outside” [2]. Most OSS projects use an Open Source tool-chain to create their software. Those tools, compilers for example, have considerable influence on the quality of the products and therefore need to be taken into account when assessing the quality of a piece of software. Furthermore, discussion about quality issues often happens in public, on mailing lists and message boards, which adds transparency. Third-party quality assessment of closed source software involves guessing in most cases.

The number of open bugs might give another impression of the quality of a product. This number is to be taken with a grain of salt since the number of bugs might indicate that there is a lot of testing, or that there are a lot of people reporting bugs. The type of bugs, response times and their frequency is important. Merely counting the number of bugs reveals more about the community behind the product than about the product itself.

The number of code check-ins gives a good idea of the activity level of the development of the product. Products that receive a lot of attention from developers are likely to be fixed faster than products that have been abandoned. A product can be very actively developed, but that might also indicate that it is unstable and many changes are being made which increase the amount of effort needed to assess and maintain a certain level of quality.

Assessing the product team is another aspect where quality assessment of OSS products differs from QA on closed source software. The term *Product Team* refers to all participants in the project, engineers, documentation team, translators, and of course QA people [3]. In closed software products, the number and skill level of developers is usually kept secret by the company, the number of participants in an OSS project can at least be estimated by educated guessing, based on commit logs and the source code itself.

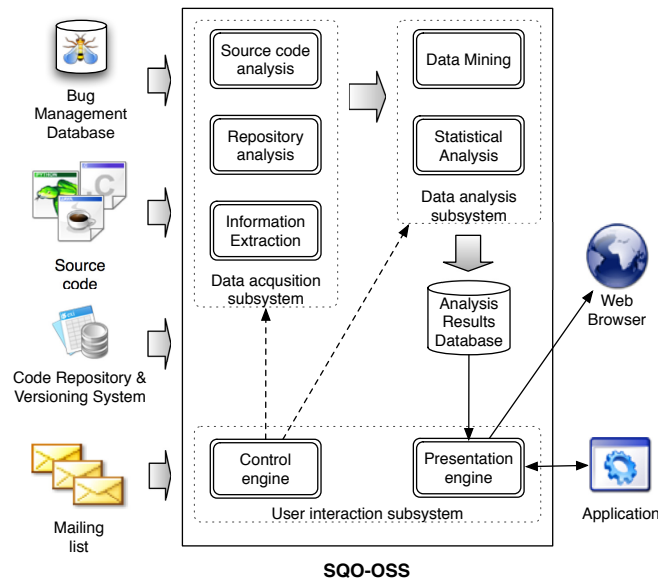
The size of the team is an important issue to examine the longevity of the product, and thus the chance to have the product supported in the future. The *Open Source Maturity Model* (OSMM) [2] uses team size explicitly as a numeric indicator of quality.

## 4 The Software Quality Observatory

The automated analysis of source code as a quality measurement is not a new concept. In recent years, the growth of OSS development has provided a wealth of code in which new techniques can be developed. Previous work in this area is often based in metric analysis: statement count, program depth, number of executable paths or McCabe’s cyclomatic complexity [5] for example. In their work using on metric-based analysis Stamelos et al. [7] observed good quality code within Open Source. Other techniques, such as neural networks [4] are not only capable of evaluating code, but also in predicting future code quality.

The Software Quality Observatory aims to provide a platform with a plug-gable architecture as outlined in figure 1 for software development organisations that will satisfy four objectives:

- Promote the use of OSS through scientific evidence of its perceived quality.
- Enhance software engineers’ ability to quantify software quality.
- Introduce information extraction, data mining and unsupervised learning to the software engineering discipline and exploit the possible synergies between the two domains using novel techniques and algorithms.
- Provide the basis for an integrated software quality management product.



**Fig. 1.** A schematic representation of the proposed system

SGO-OSS is based around three distinct processing subsystems that share a common data store. The data acquisition subsystem processes unstructured

project data and feeds the resultant structured data to the analysis stages. The user interaction subsystem presents analysis results to the user and accepts input to affect the analysis parameters. The components of the data acquisition subsystem are responsible for extracting useful data for analysis from the raw data that is available from the range of sources within software development projects. Metric analysis of source code is well-known and an important aspect of this system. Repository analysis will perform examine the commit behaviour of developers in response to user requests and security issues. The information extraction component will extract structured information from mailing lists and other textual source in order to feed higher-level analyses.

The data mining component will use structured information from project sources to predict the behaviour of the project with respect to quality characteristics and classify projects according to their general quality measurements. The statistical analysis component will apply statistical estimation models in order to predict events in the development life-cycle that can have an impact on the product's quality.

## 5 The SQO and KDE

The KDE project ([www.kde.org](http://www.kde.org)) is one of the largest desktop-oriented projects in the world. Its scope encompasses the entire desktop (i.e. end-user use of a computer, including web-surfing, email, office applications, and games). It is a confederation of smaller projects all of which use a single platform (the KDE libraries) for consistency. The project has some 1200 regular contributors and many hundreds more translators. The source code has grown to over 6 million lines of C++ in 10 years of “old-school” hacking.

KDE's quality control system has traditionally been one of “compile early, compile often.” By having hundreds of contributors poring over the code-base on a wide range of operating systems and architectures, bugs were usually found quickly. Certainly most glaring deficiencies are quickly found, but more subtle bugs may not be.

In terms of formalized quality control, there is a commit policy which states when something may be committed to the KDE repository [1], but this does not rise much above the level of “if it compiles, commit it.” Only recently has a concerted push been made for the adoption of unit tests within the KDE libraries. Adoption of the notion of *writing* unit tests has been enthusiastic, but there are questions of coverage and completeness. Automated regression testing is slowly being implemented, but here the lack of a standardized platform for running the tests hampers the adoption of those automated tests.

Documentation (user and API) quality has become an issue, and quality measurements are now done regularly. User interface guidelines have been formulated, but not enforced. Once again, there is an effort underway to measure (deviations from) the interface guidelines. This produces discouraging numbers, and has not yet been successfully automated in a large scale manner.

The KDE project expects the Software Quality Observatory to extend and enhance the quality measurements which it has begun to implement, in order to guide the actions of the KDE developers. Whether the availability of quality metrics for the code base has an effect on the “average” volunteer developer remains to be seen — experiences with the existing tools suggests that fixing bugs found by automatic techniques does not score high on the “fun” chart for developers. For the core KDE developers (of which there are perhaps 100) the existence of the quality metrics produced by the SQO may guide their efforts in bug fixing and yield more productive code freezes prior to release.

## 6 Conclusions

Software quality observation has long been performed as a crucial element in software process improvement. However, established methods of quality observation have mostly focused on source code and overlooked other available data sources e.g. mailing lists or bug fix data[6].

Many OSS projects, such as KDE, have established processes for the maintenance of software quality. However, these can only be of limited use when then actual quality of the product is still unknown. By scientifically evaluating the quality of a software *product* and not the *process*, software engineers can leverage this knowledge in many ways. By providing this quality evaluation the SQO-OSS system will allow engineers to make informed choices when addressing their development process and allow them to better maintain quality in the future. The developers and their supporting organisations can also use this evaluation to promote their product. This is especially crucial within the OSS world, where there is a wealth of choice.

Ultimately, the SQO-OSS system will aid OSS developers to write better software and enable potential users to make better informed choices.

## References

1. KDE Developer’s Corner. KDE commit policy. On <http://developer.kde.org/>.
2. Bernard Golden. *Succeeding with Open Source*. Addison-Wesley, 2005.
3. Lewis R. Ireland. *Quality Management for Products and Programs*. Project Management Institute, 1991.
4. R. Kumar, S. Rai, and J. L. Trahan. Neural-network techniques for software-quality evaluation. In *Proceedings of the Annual Reliability and Maintainability Symposium*, 1998.
5. T. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.
6. Diomidis Spinellis. *Code Quality: The Open Source Perspective*. Addison-Wesley, Boston, MA, 2006.
7. Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou, and Georgios L. Bleris. Code quality analysis in open source software development. *Information Systems Journal*, 12(1):43–60, January 2002.

