Certification of Technical Quality of Software Products

José Pedro Correia and Joost Visser

Software Improvement Group, The Netherlands
j.p.correia@sig.nl, j.visser@sig.nl

Summary. In this paper we propose a method for certification of technical quality of software products. The certification method employs a layered model of technical quality, based on the ISO/IEC 9126 international standard for software product quality. This model was developed in the context of software assessments conducted on commercial software systems over the course of several years. Using the layered quality model as a basis, we define a three-phase appraisal method that ends in certification of a software product at one of five possible levels. We illustrate the certification method by providing details of its application to twelve open source database management systems and five open source web servers.

1 Introduction

Previously, the Software Improvement Group (SIG) has developed a method for measuring the quality of software products in terms of the ISO/IEC 9126 standard [8]. It employs a small number of source code metrics, which are aggregated and mapped to quality characteristics in a manner that directly relates to the standard. The model was developed in the course of delivering IT management consultancy services [4, 13]. In particular, it has proven to be a pragmatic instrument in providing fact-based rapid estimates of technical quality of scores of large software systems.

In this paper, we explore the possibility of using the previously developed model for certification of software products. We focus on *product* certification, motivated by the current lack of standards and procedures for this purpose. We ask ourselves whether the model can be used to provide the necessary criteria and standards. And if yes, how exactly?

We can not disclose measurement data for the systems of our clients, so instead we have applied our method to a series of open source software (OSS) systems. These are listed in Table 1, ordered by their overall quality score. In the sequel, we use them to illustrate our certification approach, and we discuss in some detail the measurement data and intermediate scores obtained.

System	Version	Functionality	Main PLs	LOC	Rating
Axion	1.0-M2 (2003-07-11)	DBMS	Java	18921	⋆⋆⋆⋆
H2	1.0.69 (2008-03-29)	DBMS	Java	72102	⋆⋆⋆
CSQL	1.2 beta (2007-12-27)	DBMS	C/C++	14830	⋆⋆⋆
SmallSQL	0.19 (2007-07-31)	DBMS	Java	18402	⋆⋆⋆
JalistoPlus	2.0 M2 (2008-04-11)	DBMS	Java	21943	⋆⋆
Derby	10.3.2.1 (2007-12-10)	DBMS	Java	307367	⋆⋆
Jaminid	0.99 (2006-04-21)	Web Server	Java	1120	⋆⋆
HSQLDB	1.8.0.9 (2007-09-07)	DBMS	Java	65378	⋆⋆
SQLite	3.5.8 (2008-04-16)	DBMS	C	69302	⋆⋆
AOLServer	4.5 (2006-06-27)	Web Server	C	45344	⋆⋆
Apache	2.2.8 (2008-01-19)	Web Server	C	203829	⋆⋆
Canopy	0.2 (2007-09-12)	Web Server	C	15570	⋆⋆
Tomcat	6.0 (2008-01-17)	Web Server	Java	164199	⋆⋆
PostgreSQL	8.3.1 (2008-03-17)	DBMS	C	497400	⋆⋆
MySQL	5.0.51a (2008-01-11)	DBMS	C/C++	843175	⋆⋆
602SQL	2008-04-18	DBMS	C/C++	274938	⋆
Firebird	2.1.0 (2008-04-18)	DBMS	C/C++	357537	⋆

Table 1. OSS systems appraised, ordered by overall quality score.

The paper is structured as follows. In Section 2, we provide motivation and background to our work by discussing the position of software product quality with respect to other software quality perspectives. In Section 3, we discuss design choices and constraints for a possible certification method. In Section 4, we summarize our previously developed model and its relation to ISO/IEC 9126. In Section 5, we propose a particular method for certification in which this model is employed. We describe the method in terms of its actors, steps, and deliverables. In Section 6, we discuss the application of the model to the OSS systems mentioned in Table 1. In Section 8, we discuss related work. We conclude in Section 9 with a discussion of contributions and future work.

2 Motivation

To motivate our proposal for a certification method for software product quality, we provide a brief discussion of traditional perspectives on software quality and offer an explanation for the current lack of product certification methods.

2.1 Traditional perspectives on software quality

In IT, quality is mostly approached via three angles: process, people, and projects. These are what could be called the *environmental* characteristics of a given software product, that is, the factors that determine its development conditions and thus are expected to influence its final quality. For each of

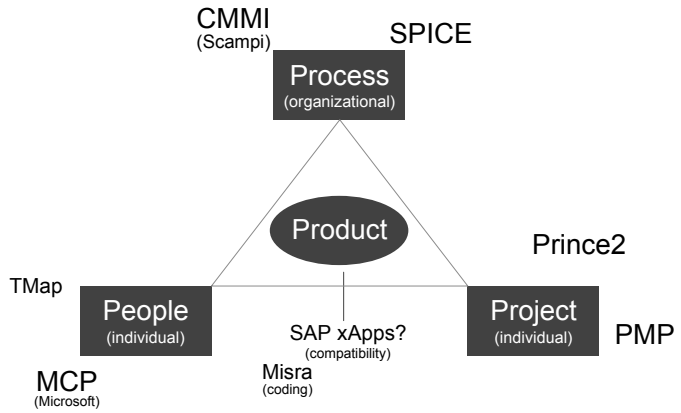

Fig. 1. Triangular representation of well-known standards and certifications in IT.

these perspectives, methods exist to improve and maintain quality, often with associated standards and certifications (Figure 1).

Regarding the software process, the most well-known certifications are CMMI (Capability Maturity Model of the Software Engineering Institute[1]) and SPICE (Software Process Improvement and Capability dEtermination, linked to the ISO/IEC 15504 standard on IT Process Assessment). These certifications are awarded at the level of entire organizations or their departments and deal with the quality of the software production process.

Certification of people concerns quality in the sense of skills and knowledge of the individuals that carry out the software production process. Such certificates are awarded after one concludes a training course. In some, but not all cases, an exam needs to be passed. Examples include the Microsoft Certified Professional, dealing with skills for particular technologies of the software giant, or TMap certification for testers.

With regard to the quality of the projects by which the software production process is structured, certifications can be obtained regarding project management methodologies, such as PRINCE2 (PRojects IN Controlled Environments of the UK Office of Government Commerce, OGC) or PMP (Project Management Professional of the Program Management Institute, PMI). Again, these certificates are awarded to individuals after appropriate training and exams.

Thus, certification is available for the software production process itself, the people that carry it out at the technical level, and the people that manage the projects by which the process is structured. But what about the result of the software production process? What certification possibilities exist for the software product?

[1] SEI uses the term "Class A Appraisals" instead of "certification"

2.2 State of the art in software product certification

Outside IT, product certification is ubiquitous. Some products receive certification on a piece-by-piece basis, such as industrial plants or the foundations of a house. Other products are certified on the basis of sampling, such as electrical plugs. The majority of the parts from which cars are assembled have been certified in some way.

Perhaps surprisingly, certification of software products is not a commonly accepted practise [5]. In fact, the possibilities of software product certification are very limited. The MISRA-C standard has been adopted in the automotive domain to enforce suppliers of embedded software to abide by certain rules for *coding style*. In the realm of implementation of Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) packages, the possibility exist of certifying the *compatibility* of custom code with the main product. This means for instance, that only SAP-endorsed technologies have been used for the ERP integration. Beyond such technology-specific or vendor-specific certificates with limited scope, software product certification is absent from current practise.

2.3 No certification without normative standard

Can we identify any fundamental reasons why software product certification is absent from current IT practises?

The very notion of certification presupposes the existence of a standard against which the object of certification can be measured and evaluated. At this moment, no widely accepted standard sets concrete, actionable norms for software product quality [5].

The closest thing is the ISO/IEC 9126 international standard for software product quality [10]. This standard answers the question of how the multi-faceted concept of software product quality can be defined. By breaking this concept down into 6 characteristics, which are further split into 27 sub-characteristics (see Figure 2), the ISO/IEC 9126 provides a terminological basis for thinking and communicating about software quality. In the technical reports that accompany the ISO/IEC 9126 standard, some metrics are suggested to measure each sub-characteristic. Unfortunately, no concrete thresholds are given for these suggested metrics, nor do their definitions clarify how to measure them in practical situations (for a more elaborate critique see [8]).

A related standard, the ISO/IEC 14598, defines an *evaluation process* of software product quality, including guidelines on how to plan and design the evaluation, as well as about its execution. This is useful as a basis to define a certification process, but does not solve the issue of how to measure software product quality or how to compare measurement values to a normative standard.

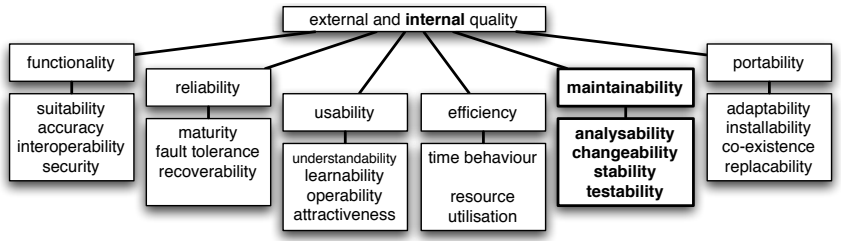

Fig. 2. Breakdown of the notions of internal and external software product quality into 6 main characteristics and 27 sub-characteristics. The 6 so-called compliance sub-characteristics of each of the 6 main characteristics have been suppressed in this picture. In addition, ISO/IEC 9126 provides a breakdown of the notion of 'quality in use' of a software product into four sub-characteristics.

3 Towards certification of software products

When proposing a method for software product certification, several design decisions need to be made. We first discuss what the precise target of certification would be. Then we discuss prerequisites on a certification method to make it viable for broad acceptance in the IT industry.

3.1 Possible targets of product certification

The target of certification is the *quality* of the software product. But which perspective on quality should be taken into account? On first consideration, several perspectives may seem likely candidates:

Functionality: Does the software product satisfy its functional requirements?
Performance: Does the product perform its required operations fast enough?
Usability: Is using the product easy and effective?

These aspects indeed represent valid views on software product quality. In fact, in the ISO/IEC 9126 quality model (see again Figure 2), these aspects can be recognized as the *suitability*, *efficiency* (time behaviour), and *usability* (operability) characteristics of internal and external quality, and as the *effectiveness* characteristics of quality-in-use.

Note, however, that all of the aforementioned aspects are tightly connected to a system's functionality and to the perspective of the end-user. This means that only systems that share certain functionality could be certified against a shared norm, and that a norm against which to certify would only be applicable to a limited set of systems.

In fact, functional quality aspects may be problematic as a target for certification for several reasons:

- Functional requirements may be unavailable, unclear, or instable.

- Functional requirements are rarely shared across systems.
- Functional requirements that *are* shared across systems typically concern only a subset of the functionality, such as particular interoperability features, GUI guidelines, or performance demands.

Due to these reasons, certification methods that target functional quality necessarily have restricted applicability.

In this paper, we choose *technical quality* as an alternative target of certification. The technical view on a software product's quality concerns how well-constructed it is. In terms of the ISO/IEC 9126, the focus of technical quality is primarily on the *maintainability* characteristic and its sub-characteristics. Other characteristics and sub-characteristics, such as *reliability* and *portability*, may also play a role.

Technical quality of a software product is interesting, not from an end-user's perspective, but from the perspective of those who maintain and modify the source code. The technical quality of a software product determines how difficult it is to keep the system evolving to remain aligned with the continuous changes in the functional requirements, whatever these are. In the longer term, technical quality may actually be the largest contributor to the total cost of ownership of a software system, because software with high technical quality can evolve with low cost and low risk to keep meeting functional and non-functional requirements.

3.2 Requirements for a certification method

Based on the preceding discussion, some prerequisites and desired traits can be laid out for a software product certification method, as we envision it.

- The method should target *technical quality*.
- The method should conform to accepted standards for software product qualty, such as ISO/IEC 9126 (quality model) and ISO/IEC 14598 (quality evaluation procedure).
- The method should be based on objective evaluation criteria and clear thresholds.
- The method should follow a transparent and repeatable procedure.
- The method should have wide applicability.

In the sequel we first describe the quality model we developed previously which conforms with ISO/IEC 9126 and sets clear criteria and thresholds for technical quality. Then we propose a certification process that conforms to ISO/IEC 14598 which is transparent and repeatable.

4 Summary of the quality model

The SIG has developed a layered model for measuring and rating the technical quality of a software system in terms of the quality characteristics of

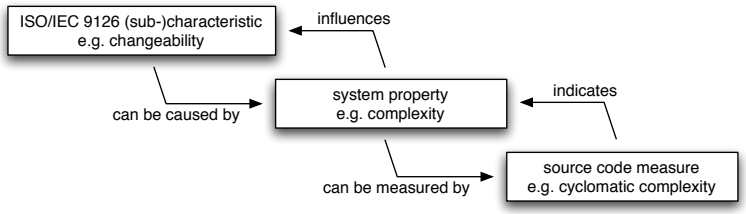

Fig. 3. The relation between source code metrics and system sub-characteristics of maintainability, as established by the SIG model.

ISO/IEC 9126 [8]. The layered structure of the model is illustrated in Figure 3. Source code metrics are used to collect facts about a software system. The measured values are combined and aggregated to provide information on properties at the level of the entire system, which are then mapped into higher level appraisals that directly relate to the ISO/IEC 9126 standard.

To aggregate metrics at the source code level to properties at the system level, we make use of so-called *quality profiles*. As an example, let's take a look at how unit complexity is calculated. First the McCabe complexity index is calculated for each code unit (where a unit is the smallest piece of code that can be executed and tested individually, for example a Java method or a C function). The values for individual units are then aggregated into four *risk categories*, as indicated in the following table (following a similar categorization of the Software Engineering Institute):

MCC	Risk evaluation
1-10	without much risk
11-20	moderate risk
21-50	high risk
> 50	very high risk

For each category, the relative volumes are computed by summing the lines of code of the units of that category and dividing by the total lines of code in all units. These percentages are finally ranked using the following thresholds:

	maximum relative LOC		
rank	moderate	high	very high
++	25%	0%	0%
+	30%	5%	0%
o	40%	10%	0%
-	50%	15%	5%
--	-	-	-

Other properties have similar evaluation schemes relying on different categorization and thresholds. Such quality profiles have as advantage over other kinds of aggregation, such as taking the average, that sufficient information is retained to make significant quality differences between systems detectable.

For the appraisal of test quality, the model does not rely on static source code analysis only. Instead we apply a structured review which takes into account the use of testing frameworks, estimates of the level of coverage achieved, and the amount of test documentation available.

The rating of system properties is first done separately for each different technology, and subsequently aggregated into a single technology-independent property rating by weighted average according to each technology's relative volume in the system.

Property ratings are mapped to ratings of sub-characteristics, following the relations summarised in the following table:

ISO 9126 maintainability	properties: duplication	test quality	unit complexity	unit size	volume
analysability	×			×	×
changeability	×		×		
stability		×			
testability		×	×	×	×

For example, duplication is mapped onto both analysability and changeability. For each sub-characteristic a straightforward averaging approach is taken. Finally, all sub-characteristic ratings are averaged to provide the top *maintainability* rating.

5 The certification method

On top of SIG's quality model, we now propose an appraisal method for carrying out software product certification. The method consists of three steps, namely *scoping*, *measurement* and *rating*. Depending on the appraisal results, the software product will be certified at one out of five possible levels of quality. These are represented using a five stars system, with the following interpretation:

rating	quality
⋆⋆⋆⋆⋆	excellent
⋆⋆⋆⋆	good
⋆⋆⋆	fair
⋆⋆	poor
⋆	very poor

We start by explaining the roles of various actors in the certification procedure, and follow by describing the procedure itself.

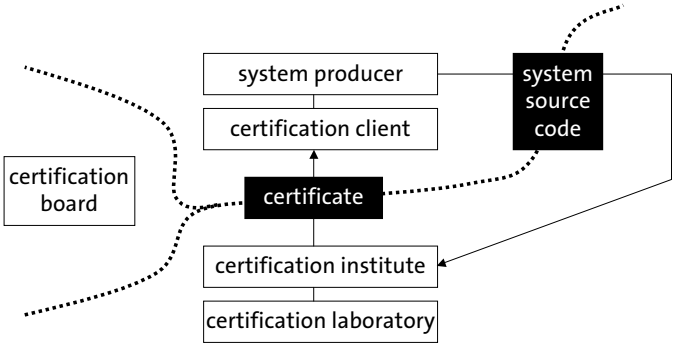

Fig. 4. Product certification roles.

5.1 Roles

Software product certification is carried out through a procedure where tasks and responsibilities are divided over various players. The roles of the various actors in the certification procedure are depicted in Figure 4.

The *system producer* is the organization that has built the software product for which certification is sought. This organization is responsible for making the source code and possibly other artefacts available to the certification institute. In case of open source software, this is the community of programmers that develops, maintains, and releases the software. For closed source software, this is the organization that ordered the system to be built and that holds property rights over the software.

The *certification client* is the organization that seeks certification of the software product. In the case of closed source software, this is typically the organization that has ordered the system to be built or that is considering to acquire the system. In case of open source software, this may be an organization that intends to adopt the software as application to support its business processes, as library in application development, or for other purposes. The certification client may work with the certification institute to determine the desired scope of the certificate.

The *certification institute* oversees the certification procedure and issues the certificates. The certification institute is responsible for brokering the relation between certification client, system producer, and certification laboratory, and for performing the scoping and rating steps.

The *certification laboratory* carries out the measurement step, in which measurement values are produced for all system units. The laboratory may also assist in defining the certification scope, if required.

The *certification board* provides oversight over the entire certification program and offers arbitration in case of disputes about application of the certification norms.

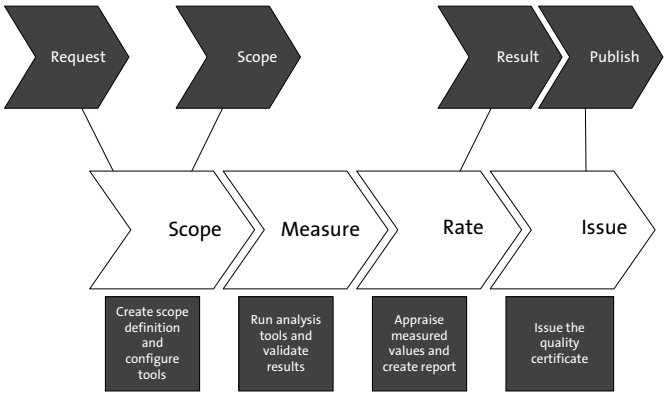

Fig. 5. Steps of the certification procedure.

5.2 Procedure

The certification procedure leads to attribution of a quality certificate in three steps, as illustrated in Figure 5.

Scope

In the first step, the scope of the certification study is determined. As a result of this step, an unambiguous description will be available of which software artefacts are covered by the evaluation.

This scope description includes the release version of the software system, a listing of file names, dates, and sizes, and a characterisation of the technology footprint of the system (which programming languages, database management system, persistence and GUI frameworks, configuration languages, communication protocols). When appropriate the description will also separate documentation from code, test code from production code, generated code from manually written code, and library from application code.

Measure

In the second step, a range of measurement values is determined for the software artefacts that fall within the evaluation scope. Each measure is determined automatically by processing the software artefacts with an appropriate algorithm.

Rate

In the third step, the raw metric values obtained in the measurement step are aggregated, weighted, combined, and subsequently compared against target

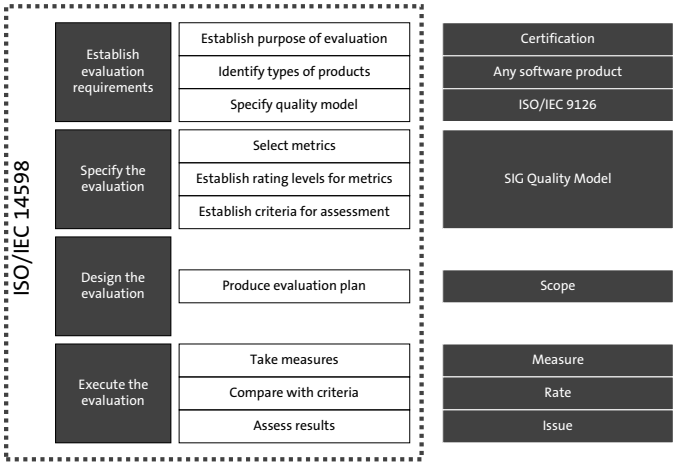

Fig. 6. Correspondance of our certification method to the ISO/IEC 14598 standard.

values in order to determine quality sub-scores and final score for the system under evaluation.

The result consists of a layered set of appraisals, namely the appraisal for the only characteristic now considered, maintainability, traceable to the sub-characteristics' appraisals which in turn are traceable to property appraisals and finally those can be related to the source code measurements that were considered.

Appraisals are represented in a 5 star system, directly correspondent to the ++,+,o,-,-- system used in the SIG quality model, being 1 star representative of very poor quality and so forth. Internally, a fractional value is used in order to maintain precision in the calculations.

Issuance

Finally, based on the rating results, a certificate is issued. The certificate is published in an online registry if the certification client so wishes.

5.3 Compliance with ISO/IEC 14598

The ISO/IEC 14598 international standard for software product evaluation [9] provides an overview of software product evaluation processes and is intended to give guidance and requirements for software evaluation.

Figure 6 depicts the overal structure of the software evaluation process according to this standard and highlights its correspondance to the certification procedure proposed in this paper. The left hand side of the picture shows the various evaluation phases and the constituent evaluation steps defined by the standard. The right hand side indicates how each step is made concrete by

	Unit testing		Functional testing			
System	Framework	Coverage	Framework	Coverage	Documentation	Appraisal
Axion	JUnit	81%	JUnit	unknown	almost none	⋆⋆⋆⋆
H2	custom	83%	custom	high	none	⋆⋆⋆⋆
CSQL	-	-	none	high	none	⋆⋆
SmallSQL	-	-	JUnit	79%	almost none	⋆⋆⋆
JalistoPlus	JUnit	low	JFunc	low	none	⋆⋆
Derby	JUnit	71%	custom	high	almost none	⋆⋆⋆
Jaminid	-	-	-	-	none	⋆
HSQLDB	JUnit	low	custom	unknown	yes	⋆⋆
SQLite	-	-	custom	very high	almost none	⋆⋆⋆
AOLServer	-	-	rudimentary	almost none	almost none	⋆
Apache	-	-	custom	high	good	⋆⋆
Canopy	-	-	-	-	none	⋆
Tomcat	JUnit	very low	-	-	almost none	⋆
PostgreSQL	-	-	custom	high	good	⋆⋆
MySQL	custom	very low	custom	high	extensive	⋆⋆⋆
602SQL	-	-	-	-	none	⋆
Firebird	-	-	TCS & QMTest	high	extensive	⋆⋆⋆

Table 2. Ratings of test quality based on structured review.

our certification method. For example, the concrete purpose of evaluation is *certification*. The metrics, rating levels, and criteria are chosen in accordance to the SIG quality model. The evaluation design is carried out in the scoping step of our procedure, and the execution of the evalation are covered by our measurement, rating, and issuance steps.

6 Application to open source software

To put the proposed certification method to the test, we conducted appraisals of some well-known open source systems. For now, 12 relational database systems, including MySQL and PostgreSQL have been examined, as well as 5 web servers, including Tomcat and Apache HTTP Server. The results are presented in this section, preceded by some remarks.

6.1 Systems under study

We were looking for systems with similar functionality, if possible of different volume and using different languages. We selected projects from two functionality groups, namely database systems and web server applications. Some systems were included for their popularity and the remaining by doing a search on SourceForge[2]. The latest *release* versions, at the time of analysis, were selected for each system. An overview was already presented in Table 1, showing systems ranging from very small to large and covering the programming languages of Java, C and C++.

[2] `http://sourceforge.net`

System	Maintainability	Analysability	Changeability	Stability	Testability	Duplication	Test quality	Complexity	Unit size	Volume
Axion	★★★★ (+0.54)	★★★	★★★★	★★★★	★★★	★★★	★★★★	★★★★	★★	★★★★★
H2	★★★ (+0.08)	★★★	★★★	★★★★	★★	★★★★	★★★★	★★	★	★★★★
CSQL	★★★ (−0.13)	★★★	★★★★	★★	★★★	★★★	★★	★★★★	★	★★★★★
SmallSQL	★★★ (−0.38)	★★★	★★	★★★	★★	★★★★	★★★	★	★	★★★★★
JalistoPlus	★★ (−0.54)	★★★	★★	★★	★★	★★★	★★	★★	★★	★★★★★
Derby	★★ (−0.54)	★★	★★	★★★	★★	★★★	★★★	★★	★	★★★
Jaminid	★★ (−0.63)	★★★★	★★★★	★	★	★★★★★	★	★★	★	★★★★★
HSQLDB	★★ (−0.70)	★★★	★★	★★	★	★★★★	★★	★	★	★★★★★
SQLite	★★ (−0.79)	★★★	★★	★★★	★★	★★	★★★	★	★	★★★★★
AOLServer	★★ (−0.83)	★★★★	★★★	★	★	★★★★★	★	★	★	★★★★★
Apache	★★ (−1.00)	★★★	★★	★★	★	★★★	★★	★	★	★★★★
Canopy	★★ (−1.04)	★★★	★★	★	★	★★★★	★	★	★	★★★★★
Tomcat	★★ (−1.08)	★★★	★★★	★	★	★★★	★	★★	★	★★★★
PostgreSQL	★★ (−1.08)	★★	★★	★★	★	★★★	★★	★	★	★★★
MySQL	★★ (−1.16)	★	★	★★★	★★	★	★★★	★	★	★★
602SQL	★ (−1.21)	★★★	★★	★	★	★★★★	★	★	★	★★★
Firebird	★ (−1.25)	★	★	★★★	★★	★	★★★	★	★	★★

Table 3. Appraisals of technical quality for the selected OSS systems, ordered by overall score.

6.2 Conducting the appraisals

The scoping of the mentioned systems was done by us, either by inspecting the available documentation, browsing through the source code or using some automatic heuristics (for example searching for specifc patterns in files to determine and exclude the generated code). This makes the process less accurate than in the ideal situation, when this is done jointly by the certification institute, the certification client and the system producer. On average we spent about 4 hours on the scoping step for each system.

Source code measurements were performed using SIG's software analysis toolkit. The scope definition served as the basis for the configuration of the toolkit. To conduct the rating step, the measurement results were fed into SIG's rating engine. This second tool implements a subset of the SIG quality model. The structured review of test quality is summarized in Table 2.

6.3 Results

The appraisal values can be found in Table 3 ordered by overall maintainability value. The sorting is done according to the fractional representation of quality appraisals used internally by the rating tool, thus systems with the same number of stars do have different internal values. These are shown for the maintainability appraisal and range from -2 (one star) to 2 (five stars).

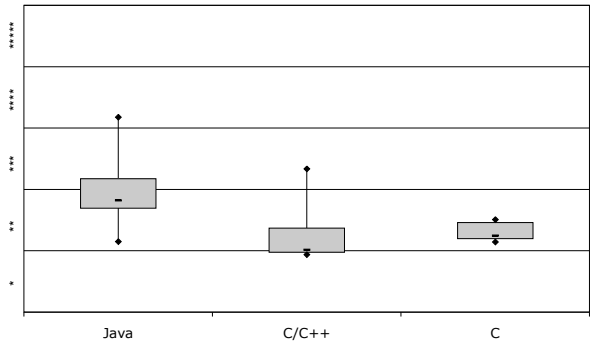

Fig. 7. Boxplot of the maintainability values per language. The Y axis is divided into 5 sections, corresponding to the star system.

6.4 Observations

The overall technical quality of the systems, as appraised by our method, is low. A rating of three stars is considered to represent medium quality. By inspecting the results one can observe that one system is rated as good (Axion), three as medium (H2, CSQL, SmallSQL), and most systems were appraised as having (very) poor quality.

We do not claim this sample to be statistically significant or representative of the population of open source systems, so it would be interesting to further investigate some trends observed here. For example, one of the most striking observations is that almost all systems rank very low regarding unit size and relatively low regarding complexity. This could be due to the model being too strict, but the definition of these particular properties, as with the others, was done based on our experience with proprietary systems, and for those a broader distribution of values is observed.

In Figure 7 one can observe the boxplots of the maintainability values per main programming language. The main conclusion one can take from this chart is that Java systems tend to rank higher. Although there are some outliers, the bulk of these systems rank around the border between two and three stars, whereas for the other languages the bulk ranks in the two stars area but closer to the border between one and two stars.

7 Relation to other quality instruments

Software product certification is a new instrument for software quality management that can be used in combination with other instruments.

Various kinds of *software testing*, such as unit testing, functional testing, integration testing, and acceptance testing are essential for software product

quality, both functional and technical. Certification does not replace testing, but operates on a higher level, is less labour-intensive, and can be performed independently. Note that certification also takes test quality into account.

Methodologies for *software process improvement* (SPI), such as the capability-maturity model (CMM) concern the production process, rather than the product. SPI works under the assumption that better processes lead to better products, which is true to a limited degree. Product certification concerns only the product, independently of its organizational context. These two instruments can thus be used independently or simultaneously.

A *software risk assessment* [4] is an in-depth, one-off investigation into the technical quality of a software system as well as into the strategic risks that it harbours. Each assessment is necessarily specific to the investigated software system as well as to its organizational context. By contrast, software product certification is not situational and enables objective comparison to common norms and standards applicable across systems.

Monitoring a software system or an entire software portfolio [12] is a continuous activity aimed at real-time progress tracking and detection of problems as they occur. Software monitoring provides increased control over the development and maintenance processes. Software product certification is a low-frequency activity providing control at the level of governance rather then at the tactical level.

Software benchmarking consists of comparing software products against each other according to desired characteristics [2]. This can be used to rank a system with respect to a group of peers, thus differing from certification where one is interested in establishing a *norm* to which to compare systems.

8 Related work

For a discussion of work related to the employed quality model we refer to Heitlager *et al* [8]. Here we briefly discuss work related to our certification method. A more elaborate discussion of various approaches to software certification is provided by Fabbrini *et al* [5].

Certification of functional quality

To our knowledge, no comprehensive proposals have been made so far for certification methods that target technical quality. However, some proposals have been made for software product certification targeting functional quality.

Heck *et al* [7] have developed a method for software certification where five levels of verification are distinguised. At the highest level, the software product is verified using formal methods where not only properties are proven about an abstract model, but about the software itself. To date, one certificate has been awarded, on the one-but-highest level.

ISO/IEC 9126 lists security as one of the subcharacteristics of functionality. The most well-known standard regarding security is the ISO/IEC 15408

standard on evaluation of IT security criteria [11] which has also been published under the title *Common Criteria for Information Technology Security Evaluation*. This standard does not specify concrete security requirements. Rather, it defines a framework for specification, implementation, and evaluation of security aspects.

Quality assessment methodologies

Various methodologies have been proposed for the assessment of open source products, including OSMM [6], QSOS [1] and OpenBRR [14]. A comparison of the latter two can be found in [3]. These methods mainly focus on the community contribution, activity and other "environmental" characteristics of the product. Although these approaches usually include assessment of technical quality, no concrete definition of measurements or norms are provided.

Open source quality research

Currently there are several research projects related to quality of open source software, e.g. FLOSSMetrics, QualOSS, or SQO-OSS [3]. Each of these three projects aims to provide a platform for gathering information regarding open source projects and possibly to provide some automated analysis of quality.

9 Concluding remarks

In this paper we motivated the need for certification for software products. We identified technical quality (how well the system is built) as the appropriate target, and formulated further requirements for a certification method.

We presented our certification method, based on a previously developed quality model which relates to the ISO/IEC 9126 standard for software product quality. The model provides objective evaluation criteria and is widely applicable because it is programming-language independent. The certification procedure itself conforms to the ISO/IEC 14598 standard.

We illustrated our certification method by applying it to a group of OSS products. The conclusion from applying certification to the OSS systems is that technical quality, especially length and complexity of units, tends to be low. Of course, this does not imply that functional quality of these systems, as experienced by end users, is low as well.

9.1 Future work

We are currently starting to apply the certification method to closed source systems in our consultancy practise. Also, we intend to apply the method to more open source systems, such as operating system kernels and office

[3] `http://flossmetrics.org`, `http://www.qualoss.eu`, `http://www.sqo-oss.org`

applications. We are collaborating with the Software Quality Observatory for Open Source Software (SQO-OSS) to make the certification results for OSS software available to the community.

The certification method introduced in this paper is light-weight, pragmatic, and broadly applicable. Over time, we expect to develop it into an increasingly sophisticated method to meet rising demands. We intend to validate and improve the method in various ways. For example, we are currently conducting benchmark studies and statistical analysis to provide further calibration of the underlying quality model [2]. We also intend to broaden the range of source code metrics and evaluated quality characteristics, including for example *reliability*.

References

1. Atos Origin. Method for qualification and selection of open source software (QSOS), version 1.6, April 2006.
2. J.P. Correia and J. Visser. Benchmarking technical quality of software products. To appear, 2008.
3. J.-C. Deprez and S. Alexandre. Comparing assessment methodologies for free/open source software: OpenBRR and QSOS. In *PROFES*, 2008.
4. A. van Deursen and T. Kuipers. Source-based software risk assessment. In *Proc. Int. Conf. Software Maintenance*, page 385. IEEE Computer Society, 2003.
5. F.Fabbrini, M.Fusani, and G.Lami. Basic concepts of software certification. In S.Gnesi, T. Maibaum, and A. Wassyng, editors, *Proc. First Int. Workshop on Software Certification (CertSoft 2006)*, Software Quality Research Laboratory. McMaster University, Canada, 2006. Report no. 37.
6. B. Golden. Making open source ready for the enterprise: The open source maturity model. Whitepaper available from `www.navicasoft.com`, 2005.
7. P.M. Heck and M.C.J.D. van Eekelen. The LaQuSo software product certification model: (LSPCM). Technical Report 08-03, Tech. Univ. Eindhoven, 2008.
8. I. Heitlager, T. Kuipers, and J. Visser. A practical model for measuring maintainability. In *6th Int. Conf. on the Quality of Information and Communications Technology (QUATIC 2007)*, pages 30–39. IEEE Computer Society, 2007.
9. International Organization for Standardization. ISO/IEC 14598-1: Information technology - software product evaluation - part 1: General overview, 2001.
10. International Organization for Standardization. ISO/IEC 9126-1: Software engineering - product quality - part 1: Quality model, 2001.
11. International Organization for Standardization. ISO/IEC 15408: Information technology - security techniques - evaluation criteria for IT security, 2005.
12. T. Kuipers and J. Visser. A tool-based methodology for software portfolio monitoring. In Mario Piattini et al., editors, *Proc. 1st Int. Workshop on Software Audit and Metrics, (SAM 2004)*, pages 118–128. INSTICC Press, 2004.
13. T. Kuipers, J. Visser, and G. de Vries. Monitoring the quality of outsourced software. In J. van Hillegersberg et al., editors, *Proc. Int. Workshop on Tools for Managing Globally Distributed Software Development (TOMAG 2007)*. Center for Telematics and Information Technology, Netherlands, 2007.
14. OpenBRR.org. Business readiness rating for open source, request for comment 1, 2005.