Standardized Code Quality Benchmarking for Improving Software Maintainability

Robert Baggen
TÜV Informationstechnik GmbH
Essen, Germany
Email: r.baggen@tuvit.de

Katrin Schill
TÜV Informationstechnik GmbH
Essen, Germany
Email: k.schill@tuvit.de

Joost Visser
Software Improvement Group
The Netherlands
Email: j.visser@sig.eu

***Abstract*—We provide an overview of the approach developed by the Software Improvement Group (SIG) for code analysis and quality consulting focused on software maintainability. The approach uses a standardized measurement model based on the ISO/IEC 9126 definition of maintainability and source code metrics. Procedural standardization in evaluation projects further enhances the comparability of results. Individual assessments are stored in a repository that allows any system at hand to be compared to the industry-wide state of the art in code quality and maintainability. When a minimum level of software maintainability is reached, the certification body of TÜV Informationstechnik GmbH (TÜViT) issues a Trusted Product Maintainability certificate for the software product.**

***Keywords*-Software product quality, benchmarking, certification, standardization.**

I. Introduction

The quality of source code is an important determinant for software maintainability. However, many projects fail to assess code quality and to control it the same way as the other classical project management KPIs for timeline or budget. This is often due to the fact that projects lack a standardized frame of reference when working with source code measurements. As a result, the quality of the product remains unknown until the (final) testing and problem fixing phase begins.

In this short paper, we describe an approach developed by the Software Improvement Group (SIG) for code analysis and quality consulting focused on software maintainability. The approach uses a standardized measurement procedure based on the ISO/IEC 9126 definition of maintainability and source code metrics. Due to measurement standardization, individual assessments can be stored in a repository. Based on the repository, any system at hand can be compared to the industry-wide state of the art in code quality and maintainability. Procedural standardization in evaluation projects further enhances the comparability of results. When a minimum level of software maintainability is reached, TÜV Informationstechnik GmbH (TÜViT) issues a Trusted Product Maintainability certificate for the software product. An illustration of the approach is provided in Figure 1.

The paper is organized as follows. In Section II we start with an explanation of the measurement model and its calibration against a benchmark database of measurement results. In Section III we describe the standardized evaluation procedure in which the model is used to arrive at quality judgements in an evaluator-independent manner. This evaluation procedure is used as part of the software product certification scheme presented in Section IV. The evaluation procedure and the certification also play a role in software quality consulting services, as explained in Section V. Section VI discusses related work. In Section VII we conclude with a summary of the approach.

II. Measuring software via code metrics

The application of objective metrics for the measurement and improvement of code quality has a tradition of more than 40 years. Today, code measurement is seen as pragmatic work – the goal is to find the right indicator for a given quality aspect and a given development environment. However, being too creative about the measurements may preclude helpful comparisons with other projects. Therefore, metrics have to be chosen with clear reference to an agreed standard – e.g. the ISO/IEC 9126 international standard for software product quality [1].

A. Software code metrics for maintainability

Conceiving maintainability as a function of code quality leads to a number of code metrics as candidates in maintainability assessments. SIG chose the code volume ("the larger a system, the more difficult it is to maintain"), McCabe's complexity ("simple systems are easier to comprehend than complex ones"), code redundancy ("duplicated code has to be maintained in all places where it occurs") and coupling ("tightly coupled components are more resistent to change") as key metrics for the quality assessments [2]. These indicators assess clearly defined aspects of maintainability. They can be calculated at least down to the unit level. This allows detailed analysis of the system when drilling down into the results later on.

B. The quality model: mapping to ISO/IEC 9126 dimensions

Since software quality measurement needs a clear mapping to an agreed standard, the measurements are interpreted in the framework of a hierarchical quality model with dimensions according to the ISO/IEC 9126. In the ISO/IEC 9126 standard, maintainability is seen as a general quality characteristic of a software product. Maintainability

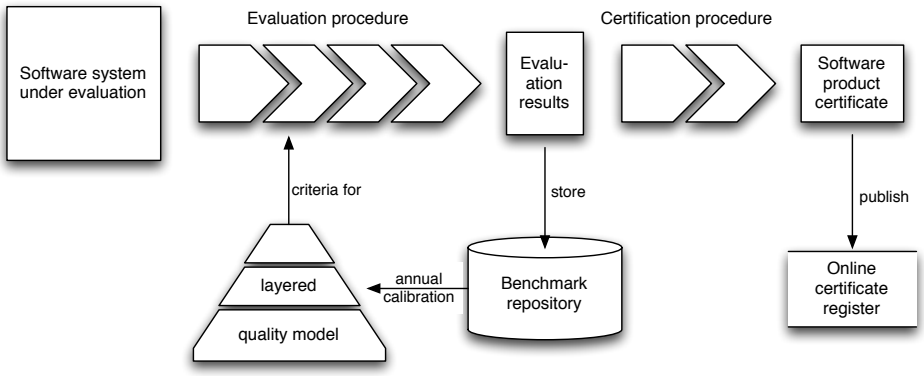

Figure 1. Evaluation, benchmarking and certification of software product quality.

is decomposed into the sub-characteristics analyzability, changeability, stability and testability [1]. In the model, the sub-characteristics are operationalized with the above source code metrics [2]. After completing the measurement, the model is used to calculate the sub-characteristics and the general maintainability score for a given system from the source code metrics [3].

A validation study on open source systems has shown that ratings as awarded by the quality model correlate positively with the speed with which defects are resolved by the system's maintainers [4].

As depicted in Figure 1, the layered quality model provides the criteria for the evaluation procedure.

C. The role of a software benchmark repository

Even with quality dimensions derived from an international standard, quality indices calculated from source code measurements remain arbitrary as long as no comparison with other systems is available. To provide such information, SIG maintains a benchmark repository [5] holding the results from several hundreds of standard system evaluations carried out so far. As shown in Figure 1, the benchmark database is updated with the results of each evaluation that is carried out.

Currently (February 2010), the benchmark repository holds results for over 500 evaluations. These pertain mostly to proprietary systems (about 75%) but also to open source systems. In these systems, a total of 45 different computer languages is used, with Java, C, COBOL, C#, C++, and ABAP as largest contributors in terms of lines of code. Some average values for key metrics are listed per represented programming paradigm in Table I.

With this extensive statistical basis, SIG can compare any system to the whole repository or to similar products in terms of size, technology or industry branch. Furthermore, interpreting the ranking, an evaluator can guide his scrutiny to parts of the code really needing improvement rather than curing minor symptoms. Such comparisons and interpretations are performed in the context of the quality consultancy services described in Section V.

The evaluation data accumulated in the benchmark repository are also used for calibrating the quality model. This is indicated in Figure 1 with a backward arrow. Calibration is performed by selecting the raw metric results from a large number of modern systems and tuning the thresholds of the quality model in such a way that for each lowest-level quality attribute a desired symmetrical distribution of systems over quality ratings is achieved. Concretely, the model is calibrated such that systems have a $\langle 5, 30, 30, 30, 5 \rangle$ percentage-wise distribution over 5 levels of quality.

Such calibration is performed with an updated set of systems at least once per year. This ensures that the quality model remains a reflection of the state of the art in software engineering.

The high number of included systems and the heterogeneity in terms of domains, languages, architectures, owners and/or producers (SIG's clients and their suppliers, as well as open source communities), help to guard the representativeness of the calibration set and to prevent abupt model changes due to re-calibration.

III. Standardized evaluation procedure

A standard evaluation procedure has been defined in which the SIG quality model is applied to software products [6]. The procedure consists of several steps, starting with the take-in of the source code by secure transmission

Table I
AVERAGE VALUES FOR REDUNDANCY AND COMPLEXITY

Paradigm or group	redundant lines	decision density (McCabe / LOC)
OOP (e.g. Java, C#)	11.8 %	20.0 %
Web (e.g. JSP,ASP,PHP)	30.1 %	5.7 %
4GL (e.g. Accell, Informix)	32.1 %	6.5 %
SQL-like (e.g. T-SQL)	28.6 %	7.9 %
C/C++	16.2 %	12.9 %

Figure 2. The quality mark *Trusted Product - Maintainability*.

to the evaluation laboratory. In subsequent steps, the scope of the evaluation is determined and documented, source code analysis tools are applied to obtain measurement results, the quality ratings in accordance with the quality model are determined, and finally the evaluation results are documented in an evaluation report. The procedure conforms to the guidelines of the ISO/IEC 14598 standard for software product evaluation [7], which is a companion standard to the ISO/IEC 9126.

To further ensure the objectivity and traceability of the evaluation, the evaluation laboratory of the SIG that carries out the procedure conforms to the guidelines of the ISO/IEC 17025 international standard for evaluation laboratories [8]. Among other things, this standard requires to have a quality management system in place that strictly separates the role of *evaluator* (who operates source code analysis tools and applies the quality model to produce an evaluation report) and the role of *quality officer* (who performs independent quality control on the activities and results of the evaluator).

IV. Certification of Maintainability

The availability of a benchmark repository provides the means for an objective comparison of software systems in terms of their maintainability. It is thus possible to assign an objective measure of maintainability to every system that undergoes the standardized evaluation procedure. This measure reflects the relative status of the system within the population of all systems evaluated so far.

Based on this system rating, TÜViT set up a certification scheme. In this scheme, systems with maintainability scores above a certain threshold are eligible for the certificate called TÜViT Trusted Product Maintainability [9] (see Figure 2). To achieve a certificate, a system must score at least with 2 stars on all sub-characteristics and at least 3 stars on the overall maintainability score. Besides reaching these minimal rankings, a system description is required to document at least the top-level components. The SIG software laboratory was accredited by the TÜViT certification body to function within this scheme as an evaluation laboratory for software code quality according to ISO/IEC Guide 65 [10].

As indicated in Figure 1, the issued certificates are published in an online registry[1]. For full traceability, the certificates and the underlying evaluation reports are always annotated with the version of the quality model and source code analysis tools that were used for evaluation.

[1]http://www.tuvit.de/english/Maintainability.asp

V. Standardized Consulting Approach

Based on experience from its assessment and monitoring projects in code quality and maintainability [11, 12], SIG developed standardized procedures to collect complementary information about a system under evaluation [13]. Usually, the projects start with submitting the source code via a secure transmission path. Next, a technical session is held together with the development staff of the customer to find out how the code is organized, what decisions were taken during development and so on. During the second, strategic session, SIG collects information from the management, i.e. the reason for the evaluation, history of the system, future plans etc. With a method of structured business scenarios for the future use of the system, together with the customer, SIG attaches risk figures to the various development options the management has for the system under test. This information can be used later on to prioritize investments in code quality linking the quality assessment and the business plan for the system.

In parallel to these session, code analysis is performed. The results of the analysis are communicated to the customers in several ways. To begin with, a validation session is scheduled to resolve results that may contradict the initial briefings. When the results are consolidated, SIG presents its findings in a management session. Finally, an assessment report is provided with the detailed results and all recommendations for improvement established during the assessment.

Thus, the standardized procedure for measuring the maintainability of source code is used both in evaluation projects leading to certification, and in consultancy projects leading to (management-level) recommendations.

Standardized software evaluations have further applications. The evaluation of software maintainability provides managers in development projects with valuable information about the quality of the code they are producing. Deciders purchasing software will reach better decisions when the code quality is evaluated for the products on their shortlist. In tendering or outsourcing procedures, software providers may prove the quality of their product with a certificate. Reaching a certifiable level of maintainability may become part of development contracts and thus raise the quality level of individual software projects.

VI. Related Work

The idea of improving software quality with the use of source code metrics has a long history. An interesting recent work is the Code Quality Management (CQM) framework proposed by Simon, Seng and Mohaupt [14]. Besides applying code metrics in large software projects, the authors introduce the idea of a benchmark repository for comparing

between projects and identifying the best practices across the software industry. However, current emphasis in CQM is given to the definition of new creative quality indicators for object oriented programming rather than to setting up a universal benchmark standard for comparison across the different software development paradigms.

Software benchmarking is usually associated to productivity rather than code quality. Jones [15] provides a treatment of benchmarking software projects. The focus is not on the software product, though the functional size of systems in terms of function points and the technical volume in terms of lines of code, are taken into account.

The International Software Benchmarking Standards Group (ISBSG) [16] collects data about software productivity and disseminates the collected data for benchmarking purposes. Apart from function points and lines of code, no software product measures are taken into account.

VII. Concluding remarks

We have provided an overview of the standardized models and procedures used by SIG and TÜViT for evaluation, benchmarking, and certification of software products. Standardization is achieved by following terminology and requirements of several relevant international standards. We have explained the role of a central benchmark repository in which evaluation results are accumulated to be used in annual calibration of the measurement model. Such calibration enables comparison of software products against industry-wide levels of quality. In combination with standardized procedures for evaluation, the calibrated model is a stable basis for the presented software product certification scheme.

References

[1] International Organization for Standardization, "ISO/IEC 9126-1: Software engineering - product quality - part 1: Quality model," Geneva, Switzerland, 2001.

[2] 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)*. IEEE Computer Society, 2007, pp. 30–39.

[3] J. P. Correia, Y. Kanellopoulos, and J. Visser, "A survey-based study of the mapping of system properties to iso/iec 9126 maintainability characteristics," in *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada*. IEEE, 2009, pp. 61–70.

[4] B. Luijten and J. Visser, "Faster defect resolution with higher technical quality of software," in *4th International Workshop on Software Quality and Maintainability (SQM 2010), March 15, 2010, Madrid, Spain*, 2010.

[5] J. Correia and J. Visser, "Benchmarking technical quality of software products," in *WCRE '08: Proceedings of the 2008 15th Working Conference on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 297–300.

[6] J. P. Correia and J. Visser, "Certification of technical quality of software products," in *International Workshop on Foundations and Techniques bringing together Free/Libre Open Source Software and Formal Methods (FLOSS-FM 2008) & 2nd International Workshop on Foundations and Techniques for Open Source Certification (OpenCert 2008)*, L. Barbosa, P. Breuer, A. Cerone, and S. Pickin, Eds. United Nations University - International Institute for Software Technology (UNU-IIST), Research Report 398, 2008, pp. 35–51.

[7] International Organization for Standardization, "ISO/IEC 14598-1: Information technology - software product evaluation - part 1: General overview," Geneva, Switzerland, 1999.

[8] ——, "ISO/IEC 17025: General requirements for the competence of testing and calibration laboratories," Geneva, Switzerland, 2005.

[9] Software Improvement Group (SIG) and TÜV Informationstechnik GmbH (TÜViT), "SIG/TÜViT evaluation criteria – Trusted Product Maintainability, version 1.0," 2009.

[10] International Organization for Standardization, "ISO/IEC Guide 65: General requirements for bodies operating product certification systems," Geneva, Switzerland, 1996.

[11] A. van Deursen and T. Kuipers, "Source-based software risk assessment," in *ICSM '03: Proc. Int. Conference on Software Maintenance*. IEEE Computer Society, 2003, p. 385.

[12] T. Kuipers and J. Visser, "A tool-based methodology for software portfolio monitoring." in *Proc. 1st Int. Workshop on Software Audit and Metrics, (SAM 2004)*, M. Piattini and M. Serrano, Eds. INSTICC Press, 2004, pp. 118–128.

[13] E. Bouwers, J. Visser, and A. van Deursen, "Criteria for the evaluation of implemented architectures," in *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada*. IEEE, 2009, pp. 73–82.

[14] F. Simon, O. Seng, and T. Mohaupt, *Code Quality Management: Technische Qualität industrieller Softwaresysteme transparent und vergleichbar gemacht*. Heidelberg, Germany: dpunkt-Verlag, 2006.

[15] C. Jones, *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley, 2000.

[16] "International Software Benchmarking Standards Group," www.isbsg.org.