Validation of a Standard- and Metric-Based Software Quality Model

Rüdiger Lincke and Welf Löwe

School of Mathematics and Systems Engineering,
Växjö University, 351 95 Växjö, Sweden
{rudiger.lincke|welf.lowe}@msi.vxu.se

Abstract. This paper describes the layout of a project[1] that we want to discuss with the scientific community. In the project, we will validate the automated assessment of the internal quality of software according to the ISO 9126 quality model. In selected real world projects, automatically derived quality metric values shall be compared with expert opinions and information from bug and test databases. As a side effect, we create a knowledge base containing precise and reusable definitions of automatically assessable metrics and their mapping to factors and criteria of the ISO quality model.

1 Introduction

Different studies show that currently more than half of the total costs in ownership of a software system are maintenance costs [7, 15]. Hence, it is important to control software qualities like maintainability, re-usability, and portability directly during the development of a software system.

Software quality is defined in the ISO/IEC 9126 standard [10], which describes internal and external software qualities and their connection to attributes of software in a so-called Quality Model, cf. ISO/IEC 9126-1:2001. The Quality Model follows the Factor-Criteria-Metric model [17] and defines six quality characteristics (or factors), which are refined into sub-characteristics (or criteria). Sub-characteristics in turn are assessed by metrics; they measure the design and development process and the software itself. ISO/IEC 9126-3:2003 provides a set of internal metrics for measuring attributes of the six defined sub-characteristics.

These measurements are currently intended to be performed manually as they require human insights, e.g., with code and document reviews. Manual approaches, however, have a series of drawbacks:

1. They are error-prone since they highly depend on the subjects performing the measurement. Hence, they are not measurements in the mathematical sense, which are required to be objective and repeatable. Humans might oversee or even deliberately ignore certain problems.

[1] The KK foundation, Sweden: Project ”Validation of metric-based quality control”, 2005/0218.

2. They are time consuming. When taking, e.g., code reviews seriously, people have to read and understand codes that they haven't created in the first place.
3. They might cause tensions in the organizations. There is a conflict of interest when, e.g., the project/quality manager requests reports from a developer, which at the same time is used to evaluate the performance of that developer.

These drawbacks are getting more severe considering current trends in software development, like outsourcing of development and integration of open source components into proprietary systems. For a reliable software production, it is essential to guarantee not only the functional correctness of external components but also the internal qualities of external components. Manual quality measurement is not an option in these settings.

Finally, many customers of software systems, especially governmental organizations or those operating in security and safety critical areas, demand certified process maturity from their vendors, e.g., as specified in the ISO 9000 series [11–13] or the Capability Maturity Model Integration (CMMI) [4]. They require quantitative reasonable statistical control over product quality as a basis for continuous quality improvement in the software and the software development process. This is, for the aforementioned reasons, hard to establish with manual quality control.

Replacing the ISO/IEC 9126 metrics manually assessing internal qualities with metrics allowing for automatic measurement resolves the above problems. The remaining problem is, however, to ensure that this automated approach is appropriate, i.e. provides a reliable assessment for at least some (sub-) characteristics. Our project aims at this validation.

In the long run, we expect to have a set of tools and methods allowing the automated assessment of software systems. Possible usage scenarios could be:

- monitoring of software quality goals during the development process resulting in early corrective actions;
- assessment of maintenance effort for change or redeveloping decisions;
- control if subcontractors or outsourced software developers meet the agreed quality goals;
- foundation to choose between different software products offered by competing companies. Is the cheaper product really the better choice in the long run?

Yet in all these activities, the tools and methods are expected to be indicators of bad quality, making reviews more efficient and directed. In any case, manual reviews are needed to validate an issue or identify false positives. We do not expect to create a fully automated assessment of software quality which can make decisions completely independent of human insight.

The structure of this paper is as follows. Section 2 explains our research goal and the expected results. Section 3 defines the scientific approaches. Section 4 gives an overview of the knowledge repository. Section 5 provides an overview about the participating companies and projects. Section 6 concludes the discussion.

2 Goal of our Research

We replace ISO/IEC 9126 metrics manually assessing internal qualities with metrics allowing for automatic measurement. This defines an adapted Quality Model. We validate the significance of this Quality Model with experiments in four selected software development projects of four Swedish companies[2] ranging from a small software company to a large company from the SAAB group.

The project will deliver both research insights and practical methods and tool support for participating companies.

On the *research side*, we expect two major contributions:

1. We define an adapted Quality Model, assessing internal quality (sub-) characteristics as defined by an industry standard with well-established metric analyses as proposed by the research community. This quality model is published as a compendium of software quality standards and metrics [16].
2. We validate the significance of that novel Quality Model, i.e. we support or disprove the hypothesis that static metric analyses allow for an assessment of (some) internal qualities of software systems.

Together, (1) and (2) provide the theoretical basis for quality management assessing industrially standardized software qualities in an effective way, since they are significant and objective, and in an efficient way, since they are automated.

On the *practical side*, we produce tools and methods supporting the quality assessment of software under development having a thorough theoretical basis. By implementing them in the participating partner companies, we gain understanding of their practicality and usefulness.

3. We get insights on how our theory, tools and methods integrate with different quality management processes existing in industry. This includes insights on initial efforts for integration and possible/necessary adaptations of these processes.
4. We understand the speed-up in performance of assessing internal quality automatically vs. manually, since we implement both approaches: the manual standard approach for validating the significance of the new automated approach.

Regarding the practical contributions we can expect that the participating companies already have quantitative information available, which can be understood as manual metrics. Of course, the quality of the available information can not be expected to be sufficient for our evaluation, and modifications will be necessary. Therefore, both approaches will be implemented and adjusted in parallel.

As a side effect, we expect a higher awareness of internal quality issues in participating companies as a first result. We even expect improvements of software

[2] Respecting our Intellectual Property Rights agreement we do not name the companies.

quality in the companies as well as improvements in their development process. These effects, however, will be evaluated qualitatively and subjectively in the first place and not validated statistically.

3 Scientific Approaches

This section describes the scientific approach for validating the significance of the metric-based Quality Model. First, we informally summarize the general idea for the validation. Thereby, we also give some examples for the metrics we use. Second, we define the precise goal and questions for the validation. Finally, we provide the background for the statistical analysis.

3.1 Validation Idea

The idea for the validation is to use static and dynamic (metric) analyses applied on the version history of particular software systems, and additional information sources like bug databases, and even human insights.

To avoid confusion, we distinguish model metrics from validation metrics. The former are automated metrics in the new Quality Model mapped to sub-characteristics. The latter are metrics assessing the (sub-) characteristics directly, but with much higher effort, i.e. with dynamic analyses or human involvement, or a posteriori, i.e. by looking backward in the project history.

For instance, a model metric for the sub-characteristic "Testability" might assess the number of independent paths in a method like the McCabe Cyclomatic Complexity metric. A corresponding validation metric of this sub-characteristic might count the actual coverage of test cases for that method. The former can be assessed easily by static metrics; the latter requires dynamic analyses.

A model metric of the sub-characteristic "Changeability" might assess the change dependency of client classes in a system triggered by changes in their server classes like the Change Dependency Between Classes [8] metric. A corresponding validation metric of this sub-characteristic might observe the actual change costs when looking backwards in the project history. Again, the former can be assessed easily by static metrics; the latter requires version analyses and human effort in documenting programming cost for changes.

The model metric of the characteristic "Maintainability" is some weighted sum of the aggregated values of the metrics assigned to the sub-characteristics of "Maintainability" (to be defined precisely in the software quality model). The validation metric of "Maintainability" could compare change requests due to bug reports, bug-fixing changes, and new bug reports, which again requires backwards analyses and human annotations.

For each single version in the version history of particular software systems, source and binary codes are available, which are input to our model metrics. Additional information about bugs reported, test reports informing about failed/passed test cases, and costs of work spent on the system for maintenance or development is available too and can be associated with a certain version.

This information is input to our validation metrics. Based on this, we can, on the one hand, determine the quality of each version according to our Quality Model, and on the other hand determine the quality based on the additional information. We assume that a higher quality according to our model correlates to fewer bugs, fewer failed test cases, and lower maintenance and development costs. Opposed to this, a low quality, according to our model, would correlate to many reported bugs, failed test cases, and higher costs for maintenance and development, etc.

Our validation succeeded if software systems having high quality according to our model metrics have also a high quality according to the validation metrics and vice versa.

3.2 Validation Goals and Questions

The project goal is a Quality Model allowing for automated metrics-based quality assessment with validated significance.

For validating the significance, we apply the Goal-Question-Metric (GQM) approach [1]. The GQM approach suggests defining the experiment goals, to specify questions on how to achieve the goals, and to collect a set of metrics, answering the questions in a quantitative way.

The goal is to validate the significance of our Quality Model based on the model metrics. Questions and sub-questions are derived from the ISO/IEC 9126 directly:

Q1: Can one significantly assess re-usability with the model metrics proposed in the Quality Model?

Q1.1 - Q1.4: Can model metrics significantly assess understandability, learnability, operability, and attractiveness, respectively, in a reuse context?

Q2: Can one significantly assess efficiency with the model metrics proposed in the Quality Model?

Q2.1 - Q2.2: Can model metrics significantly assess time behavior and resource utilization?

Q3: Can one significantly assess maintainability with the model metrics proposed?

Q3.1 - Q3.4: Can model metrics significantly assess analyzability, changeability, stability, and testability?

Q4: Can one significantly assess portability with the model metrics proposed?

Q4.1 - Q4.2: Can model metrics significantly assess adaptability, and replaceability?

For answering each sub-question, we need both a number of model metrics, which are defined in our Quality Model, and validating metrics, which are defined in our experimental setup, cf. examples above.

3.3 Background for Statistical Analysis

The basis for the statistical analysis of an experiment is hypothesis testing [18]. A negative hypothesis is defined formally. Then the data collected during the experiment is used to reject the hypothesis, if possible. If the hypothesis can be rejected, then intended positive conclusions could be drawn.

Our negative null hypothesis H_0 states that correlations of model and validation metrics are only coincidental. This hypothesis must be rejected with as high significance as possible. We start for all our analyses with the standard borderline significance level of 0.05, i.e. observations are not coincidental but significant with at most a 5% error possibility. The alternative hypothesis H_1 is the one that we can assume in case H_0 is rejected.

To define the hypothesis, we classify the measured values as high, average, and low. For this classification we use a self-reference in the software systems under development: systems are naturally divided in sub-systems, e.g., packages, modules, classes etc. More precisely, for each (sub-) characteristic c and each sub-system s:

1. We perform measurements of model and validation metrics.
2. The weighted sum as defined in the Quality Model defines aggregated values $VM(c, s)$ from values measured with model metrics. We abstract even further from these values and classify them instead with abstract values $AM(c, s)$. It is:
   - $AM(c, s)$ = high iff $VM(c, s)$ is among the 25% highest values of all sub-systems,
   - $AM(c, s)$ = low iff $VM(c, s)$ is among the 25% lowest values of all sub-systems, and
   - $AM(c, s)$ = average, otherwise.
3. The validation metrics provide values $VV(c, s)$ for direct assessment of (sub-) characteristics.

Abstraction and normalization from the precise metric values VM to the abstract values AM is necessary, since VM delivers values in different ranges and scales. For instance the Line Of Code metric has positive (in theory infinite) integer values, whereas the Tight Class Cohesion metric delivers rationale values between 0.0 and 1.0. However, they all have in common that there is a range in which values are acceptable and outlier ranges which indicate issues.

Selecting 25% as boundary values seems to be a suitable first assumption. We expect that the majority of the values vary around a (ideal) median, whereas the outliers are clearly distinguished. The values will be adjusted if first analysis results suggest other boundaries.

Our statistical evaluation studies the effect of changes in $AM(c, s)$ (independent variables) on $VV(c, s)$ (dependent variables). The hypotheses H_0 and H_1 have the following form:

H_0 : There is no correlation between $AM(c, s)$ and $VV(c, s)$.
H_1 : There is a correlation between $AM(c, s)$ and $VV(c, s)$.

In order to find out which dependent variables were affected by changes in the independent variables, we may use, e.g., the Univariate General Linear Model [20], as part of the SPSS system [19], provided the obtained data is checked for test condition suitability.

4 The Compendium

Currently, we build a knowledge base [16] mapping standard qualities to well-established software metrics and vice versa. This compendium formalizes our hypotheses.

The goal of the compendium is to provide an information resource precisely defining our interpretation of the software quality standards and the software metrics and their variants. Moreover, we propose connections between them. These connections are the hypotheses to be validated.

Currently, the compendium describes only

- 37 software quality properties (attributes, criteria),
- 14 software quality metrics.

The 37 quality properties are taken from the ISO 9126-1 standard. For a description, we refer to the ISO standard [10]. The 14 software quality metrics are taken from different well know metrics suites like Chidamber and Kemerer [5], Li and Henry [14], Bieman and Kang [2], or Hitz and Montazeri [9, 8] and contain among others Weighted Method Count (WMC), Tight Class Cohesion (TCC), Lack of Cohesion in Methods (LCOM), McCabe Cyclomatic Complexity (CC), Lines Of Code (LOC), Number Of Children (NOC), Depth of Inheritance Tree (DIT), Data Abstraction Coupling (DAC), and Change Dependency Between Classes (CDBC), etc. The non-object-oriented metrics are mainly size and complexity metrics. The object-oriented metrics focus on cohesion, coupling, and inheritance structure.

We chose metrics that have been discussed, are accepted, validated in case studies, and commented, e.g., by the FAMOOS project [3]. The quality properties and metrics are linked to each other over a double index, allowing us to determine the relevance between the metrics and criteria from each point of view. However, this compendium is meant to be a live document going beyond the normal experience sharing in conferences and workshops. We wish to create a compendium in the spirit of "A compendium of NP optimization problems" edited by Pierluigi Crescenzi and Viggo Kann [6]. The difference is that we propose a double index. The community is welcome to contribute with new metrics or comments, corrections, and add-ons to already defined ones. References to validating experiments or industrial experiences are especially appreciated. The contributions proposed by the community in the form of web forms or emails will be edited, added to the compendium, and used to create new references in the double index, or to change/remove references proofed invalid.

5 The Participating Companies and Projects

Each software system assessed in our project implies other constraints, and the emphasis on the parts of the applied Quality Model varies. This is because the participating companies have distinct expectations and priorities on the quality factors and criteria. Additionally, individual requirements for the application of the quality model result from the architecture and design, programming languages, and development environments.

All selected projects allow for quantitative and qualitative analysis. It is possible to look back in time by accessing their software repositories and to observe their long term evolution during the three years the project shall go on. During this time, the metrics (cf. Sect. 4) can be validated empirically for their usefulness, their limits, and their applicability in new areas like web based applications.

The first company is developing Web applications with C# as implementation language, running on .NET Framework (2.0, Windows) and Mono (Linux pendant). Because one and the same application is required to run on both systems, portability is a particularly important quality factor. We plan to assess in particular the adaptability and replaceability characteristics according to our quality model described in the compendium (cf. Sect. 4). Relevant metrics are, among others, WMC, TCC, LCOM, and LOC. Additionally, we need to define appropriate new metrics, assessing peculiarities with C# code written for .NET and Mono, since not all code running on .NET runs without problems on Mono, unless some special rules are followed. The details for these metrics still need to be discussed and formalized with the company.

The second company is developing Web applications with Java 1.4 and JSP for JBoss using Eclipse and Borland Together J. The product is currently in the maintenance phase. The main interest is on assessing the maintainability and on preventing decay in architecture and design. As the maintainability factor is of highest interest, its characteristics of analyzability, changeability, stability and testability are assessed. The compendium connects them in particular, with LOC, WMC, and CC assessing the complexity, and TCC and LCOM assessing at the cohesion. A particular challenge is that Java Applets, JSP and HTML are part of the design which need to be taken into account when assessing the maintainability of the software system. Other metrics might be included to adapt the Quality Model to the product specific needs.

The third company is developing Web applications with Java. They are currently interested in software quality in general, how it can be automatically assessed, and what it can do for them. They do not have a particular expectation or need for an emphasis on a certain quality factor; therefore, the complete Quality Model, as described in the compendium, will be applied, covering quality factors like (re-)usability, maintainability, and portability, but also reliability and efficiency. Once some higher awareness about what the internal quality has been achieved, the quality model will be adjusted.

The last company is developing embedded control software with C. Their main concern is the quality of design and maintainability. This means the focus lies on the maintainability factor with an emphasis on the architecture and design. Suitable metrics assessing inheritance, coupling, and cohesion are NOC, DIT, DAC, CDBC, LCOM, and TCC, as described in the compendium. Complementing these metrics, design patterns and anti-patterns might become interesting as well.

We do currently not expect particular problems with collecting data for the model metrics, since the source code in all projects is available in version management systems. Collecting data for the validation metrics is expected to work without problems as well, but might involve more human effort, since the data is not always available in an easily processable way.

6 Conclusions

This paper defines the layout of an experiment in quality assessment. In contrast to other experiments of this kind, it addresses two concerns, which are usually on the common wish list of experiments in this area: comparing automated metrics collection vs. manual metrics collection, and the involvement of industry partners. It should answer the question: Is it possible - in general or to some degree - to automatically assess quality of software as defined by the ISO 9126 standards using appropriate metrics?

We are aware of threats to our approach that are hard to control, like the influence of the projects used for validation (their size, programming language, duration, maturity of company and programmers, etc.) and the validity of the version history and additional information sources. Other problems like a precise definition of standards and metrics (and their information bases) appear only to be resolved as a community activity.

Altogether, the paper aims at entering the discussions on usefulness of such a validation in quality assessment, threads, and possible common efforts.

References

1. Basili, V.R., Rombach, H.D.: The TAME project: Towards improvement-oriented software environments. Trans. Software Engineering 14, 6 (June), IEEE, 1988, pp. 758-773.
2. Bieman, J.M., Kang, B.K.: Cohesion and Reuse in an Object-Oriented System. Proceedings of the ACM Symposium on Software Reusability, April 1995.
3. Bär, H., Bauer, M., Ciupke, O., Demeyer, S., Ducasse, St., Lanza, M., Marinescu, R., Nebbe, R., Nierstrasz, O., Przybilski, M., Richner, T., Rieger, M., Riva, C., Sassen, A., Schulz, B., Steyaert, P., Tichelaar, S., Weisbrod, J.: The FAMOOS Object-Oriented Reengineering Handbook. http://www.iam.unibe.ch/~famoos/handbook/, October 15, 1999.
4. Capability Maturity Model Integration (CMMI). http://www.sei.cmu.edu/cmmi/, 2006.
5. Chidamber, S. R., Kemerer, C. F.: A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, vol. 20, no. 6, pages 476-493, June 1994.

6. Crescenzi, P., Kann, V., Karpinski, M., Woeginger, G.: A compendium of NP optimization problems. `http://www.nada.kth.se/~viggo/wwwcompendium/`, 2006.
7. Erlikh, L.: Leveraging legacy system dollars for E-business. IT Pro, IEEE, May/June 2000, pp. 17-23.
8. Hitz, M., Montazeri, B.: Measure Coupling and Cohesion in Object-Oriented Systems. Proceedings of International Symposium on Applied Corporate Computing (ISAAC'95), pages 24, 25, 274, 279, October 1995.
9. Hitz, M., Montazeri, B.: Chidamber and Kemerer's Metrics Suite; A Measurement Theory Perspective. IEEE Transactions on Software Engineering, vol. 22, no. 4, pages 267-271, April 1996.
10. ISO/IEC 9126-1 Software engineering - Product Quality - Part 1: Quality model, 2001.
11. ISO 9000:2005 Quality management systems Fundamentals and vocabulary, 2005.
12. ISO 9001:2000 Quality management systems Requirements, 2001.
13. ISO 9004:2000 Quality management systems Guidelines for performance improvement, 2000.
14. Li, W., Henry, S.: Maintenance Metrics for the Object Oriented Paradigm. IEEE Proceedings of the First International Software Metrics Symposium, pages 52-60, May 1993.
15. Lientz, B.P.,Swanson, E.: Problems in application software maintenance. Communications of the ACM 24 (11), ACM, 1981, pp. 763-769.
16. Lincke, R., Löwe, W.: Compendium of Software Quality Standards and Metrics. `http://www.arisa.se/compendium/`, 2006.
17. McCall, J. A., Richards, P.G., Walters, G.F.: Factors in Software Quality. Volume I. NTIS AD/A-049 014, NTIS Springfield, VA, 1977.
18. Spiegel, M., Schiller, J., Srinivasan, A.: Probability and statistics. New York: McGraw-Hill, 2001.
19. SPSS. `http://www.spss.com`, 2005.
20. Walpole, R.E.: Probability and Statistics for Engineers and Scientists. Prentice Hall, NJ, 2002.