Software Quality Modelling Put Into Practice

Benedikt Mas y Parareda and Jonathan Streit

itestra GmbH, Ludwigstrasse 35, 86916 Kaufering, Germany
{mas, streit}@itestra.de
http://www.itestra.de

Abstract. This paper describes experiences from the practical application of a conceptual quality model in a large-scale commercial organization. The goal was to increase the economic efficiency of the existing application landscape. We used an activity-based two-dimensional quality model to evaluate software systems and to deduce effective and cost-efficient quality improvement actions. An important aspect of our work was the communication of results to the stakeholders, convincing them to implement the recommended actions.

1 Software quality assessment

Most of today's businesses strongly depend on large deployments of software, both for internal operation and as parts of their products. For example, back-end accounting systems of large companies comprise several million lines of code and cause significant annual costs. Operation and on-going development—also known as maintenance [1]—entail the major part of these expenses.

The costs caused by a software system are largely determined by its quality [2] and the quality of the processes managing the system. The scientific community has developed various approaches for modeling quality (see [3] for an overview) and the evaluation of software systems. However, there are only few reports on how these models can be applied in commercial settings and how they need to be adapted respectively extended for practical application.

In the past three years, we analysed the quality of software systems with a total of more than 20 million lines of code of various technologies for different clients as part of our software quality consulting work [4].

As part of these consulting activities, we adapted the two-dimensional quality model TUM-QM [3] to allow objective assessments of large scale legacy applications. These adaptations provide convincing argumentations for economically justified improvement actions.

1.1 Scientific foundations

The two-dimensional quality model TUM-QM models software quality by separating a hierarchy of technical and organizational *facts* of a system from the *activities* that are influenced by these facts. Activities drive the costs of the system under evaluation.

In contrast to other quality models, this approach enables the identification of defects of great economic relevance and therefore find improvement actions that deliver an optimal return on investment.

1.2 Instantiation of the quality model

To apply the rather high-level quality meta-model TUM-QM to the analysis of software systems, we had to determine the particular activities that are of interest to the owner of the system. Furthermore, we had to find facts influencing these activities and methods to assess these facts.

A simplified extract of the instantiated quality model is shown in figure 1 (actions are introduced in section 2.1).

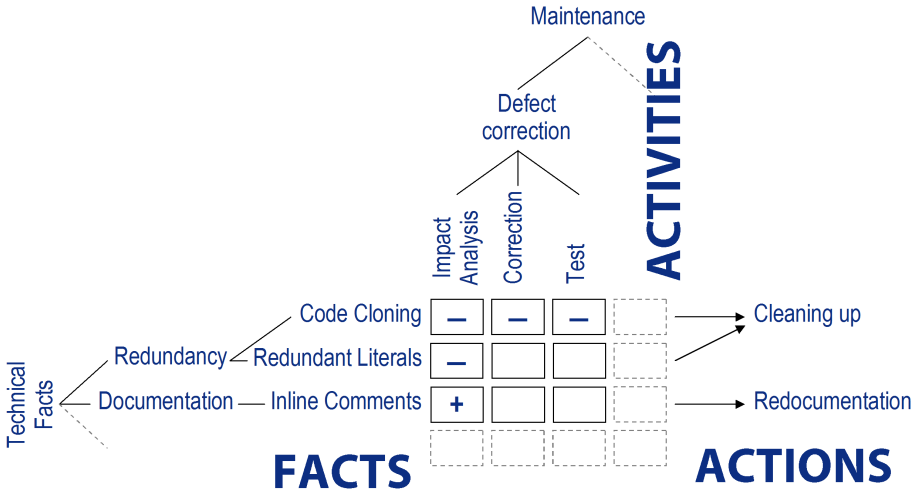

Fig. 1. A simplified extract of the quality model

Relevant activities. Since the importance of activities corresponds to the relative costs of these activities, we decided to derive the set of activities directly from the cost structure used by the clients to perform financial controlling. As virtually every organization of significant size performs some form of financial controlling, deriving activities for TUM-QM becomes straightforward and guarantees relevance to and understanding by the owner of the system to be assessed.

The main cost items (and corresponding activities) we identified were: *development*, *maintenance*, *operation* and *hard- and software resources*. Each cost item was split up into more detailed cost items (e.g., user support is contained within operational cost).

Interestingly, financial controlling data repeatingly shows that operation plus hard- and software-resources (licenses) cause two thirds of all long-term costs. Accordingly, technical facts affecting the run-time performance have gained greater importance during our investigations than previously expected.

Useful facts. The topmost elements of the facts hierarchy, namely strategical, organisational and technical facts, were inspired by existing catalogues such as the *Software Reengineering Assessment Handbook* [5]. Using a top-down approach, we recursively refined the top-level elements into measurable metrics. The technical facts extend to areas such as data management and documentation. All metrics were designed to describe a fact that has a significant impact on relevant cost items.

Our current facts catalogue contains about 260 facts, measured using both well-established and innovative metrics such as *Code Cloning Ratio*, *IF-Ratio*, *Literal Redundancy*, *Size of the Identifier Vocabulary*, *Number of Attributes per Database Table* and *Number of Stakeholders actually knowing the system*. If possible, we design our metrics to be intuitively understandable, such as *Percentage of Duplicated Code*. Simple scales can be communicated more easily and the analysis process becomes more comprehensible for observers.

Descriptive facts such as *Age of the System*, *Code Size* or *Used Programming Languages* that have no apparent economic impact complete the facts catalogue. These descriptive facts can provide valuable evidence for suggesting efficient improvement actions and are useful to demonstrate a thorough knowledge of the system under evaluation.

1.3 Assessment process

Commercial quality assessment has to adhere to rigorous time and budget constraints (sometimes only a few hours per system) without sacrificing validity of the results. Therefore, we apply a combination of assessment methods:

- Stakeholder interviews using a fixed set of approximately 100 questions. The questions are distributed to the stakeholders beforehand, so that the actual interview can be conducted in one hour.
- An automated technical analysis using an adapted and extended version of the toolkit ConQAT [6].
- Analysis of compiler output as well as data from the configuration management system and the database management system.
- Structured inspections on a randomized sample of source code and documents. A scale of school grades is used to rate these properties.

Manual and semi-automatic evaluations play an important role in our assessment process. This is due to the top-down refinement approach that selects facts according to their relevance, regardless of whether they can be assessed by a tool or not. Experience shows that manual inspections provide extremely valuable

information about the state of a system. Many relevant facts, such as *Usefulness of Code Comments*, can not be evaluated automatically, as they require semantic understanding and thus human assessment. Although the inspections are based only on a sample and on subjective judgement, their results are usually consistent both within different parts of a system and different observers, and our stakeholders expressed no doubts about their validity. On the contrary, inspections can serve to collect illustrative excerpts of code or documentation to support the analysis result.

2 From quality defects to cost reduction

In a commercial environment, measuring quality alone is in vain. A quality analysis is only worthwhile if improvement actions with quantifiable economic benefits can be derived. Additionally, stakeholders need to be convinced that the economic advantage is attractive, realistic and achievable with controlled risk.

2.1 Deducing improvement actions

For each fact in our facts catalogue we defined improvement actions that have a positive impact on this fact. The actions are organized into the categories *process optimization*, *retrieval and organisation of information*, *defect correction* and *reengineering*.

Only improvement actions with an expected positive return on investement are candidates for implementation. We deduce improvement actions in three steps:

1. Current data from financial controlling indicates the most expensive activities.
2. Each of these activities is related to a set of facts through the quality model. If any of these facts is assessed critically, high potential for economic improvement exists.
3. For each improvement action linked to critical facts, a cost-benefit analysis is performed and presented to management.

As the quality model in its current form does not provide any quantitative information on the influence a fact has on a cost item, economic potential and impact of the improvement actions were estimated manually.

2.2 Communication

Stakeholders ranging from top management to programmers are often strongly attached to a software system—sometimes in an irrational way—and thus unwilling to accept the need for quality improvement and to initiate and support the required actions. Hence, a presentation strategy for the results that conveys the message of economic potential convincingly is crucial for success.

We start our argumentation by explaining the analysis method and describing the system under evaluation using facts such as *Lines of Code* or *Number of Database Tables*. This phase serves to manifest that a detailed analysis had taken place.

As a second step, we compare the system under evaluation with other systems within the same company and the State of the Art in software engineering. This comparison is made using a benchmark that aggregates a subset of the results. This benchmarking is crucial, as it gives everybody in the audience a point of reference for the interpretation of the results. A visualization of the benchmark results for a set of systems is shown in figure 2. Systems with both high annual costs and severe quality defects and thus high potential for improvement are located in the upper right area of the diagram.

Using this picture, we claim that potential for improvement exists and present our estimate of the investement required to realize this potential.

Subsequently, we support these claims by presenting a selection of metrics that illustrate the state of the system. We increase the credibility of the metrics through code snippets and other examples taken from the actual system. Contrary to popular belief, these excerpts were very well-received even by management audiences and provided strong evidence in discussion. Stakeholders that had been reluctant to accept that a particular property of their system was harmful could often be convinced when confronted with an example, such as redundant pieces of code where the same bug fix had been applied and tested multiple times.

Finally, we present improvement actions deduced from the assessment results together with a detailed cost-benefit analysis. This gives everybody in the audience the chance to identify benefits for their department.

2.3 Successful implementation of improvement actions

We have been able to demonstrate the effectiveness of our analysis method by implementing the proposed improvement actions.

In one case, for example, the cost data for the system under evaluation showed high spendings for load-dependent resource usage. In the analysis of the system we identified inefficient algorithms, indicating potential for improvement. The cost-benefit analysis suggested a positive return on investement for the optimization of the most expensive components. Hence, this optimization was recommended to management.

The optimization project proved to be highly successful, as it provided a positive return on investement after just a few months and created a long-lasting economic benefit.

3 Conclusion

In this experience report we described how a quality model can be applied in a commercial environment and the extensions that are needed to perform successful software quality assessments. Several of our analyses using the presented

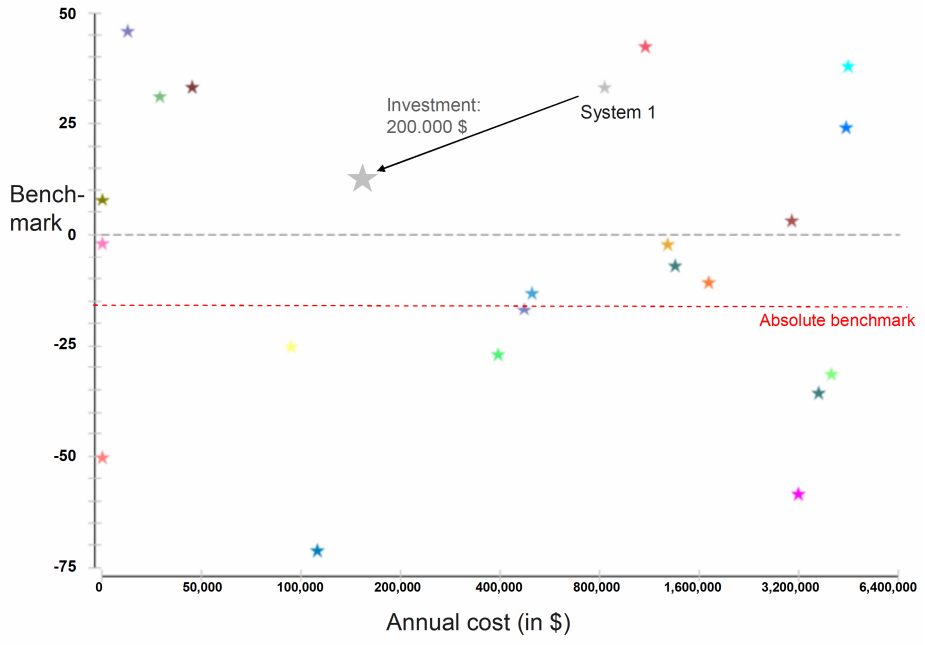

Fig. 2. A sample benchmark analysis. The arrow indicates the technical and the resulting economic improvement opportunity.

method have led to investments in improvement actions. The predicted benefits were achieved or even exceeded.

In commercial settings, the challenge in quality assessment is not achieving ultimate precision in measurement, but providing a reliable, objective and sound estimation of a system's quality and delivering a convincing business case for improvement.

Usefulness of TUM-QM. The quality model TUM-QM has proven to be a valuable base for quality assessments. For our purposes, the main advantage of TUM-QM is the inclusion of activities in the quality model. This guarantees that an analysis based on TUM-QM does not evaluate abstract quality, but identifies properties that entail severe consequences for the selected activities. Supposed quality defects that only violate the ideal conception of a system but do not involve a financial penalty are ignored.

Unfortunately, due to the missing valuation of the relation between facts and activities, the quality model alone is not sufficient to quantify the financial impact of quality defects. A significant manual effort is required to perform the cost-benefit analysis of improvement actions. Even an approximate weighting would greatly reduce the effort required to apply the quality model.

References

1. Pigoski, T.M.: Practical Software Maintenance: Best Practices for Managing Your Software Investment. John Wiley & Sons, Inc., New York, NY, USA (1996)
2. Wagner, S.: Using Economics as Basis for Modelling and Evaluating Software Quality. In: Proc. First International Workshop on the Economics of Software and Computation (ESC-1). (2007)
3. Deissenböck, F., Wagner, S., Pizka, M., Teuchert, S., Girard, J.F.: An activity-based quality model for maintainability. In: Proceedings of the 23rd International Conference on Software Maintenance (ICSM 2007), IEEE CS Press (2007)
4. Mas y Parareda, B., Pizka, M.: Reengineering web-basierter und anderer junger Legacy-Systeme – Erfahrungsbericht. Technical report, itestra GmbH, Garching, Germany (2006)
5. Johnson, R.E.: Software Reengineering Assessment Handbook, Version 3.0. US Department of Defense (1997)
6. Deissenböck, F., Pizka, M., Seifert, T.: Tool Support for Continuous Quality Assessment. In: STEP '05: Proceedings of the Workshop on Software Technology and Engineering Practice. (2005)