# T U M

I N S T I T U T F Ü R I N F O R M A T I K

# Workshop-Band Software-Qualitätsmodellierung und -bewertung (SQMB ’08)

Stefan Wagner, Manfred Broy, Florian Deissenboeck, Jürgen Münch, Peter Liggesmeyer

TUM-I0811
April 08

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-04-I0811-0/1.-FI


Druck: Institut für Informatik der
Technischen Universität München

# Vorwort

Qualität ist seit Beginn der kommerziellen Entwicklung von Software ein wichtiges Thema in Forschung und Praxis und diese Bedeutung verstärkt sich noch weiter. Heutige Entwicklungen stellen zusätzliche Anforderungen an verschiedenste Qualitätsaspekte dar. Beispielsweise führt die Durchdringung von kritischen Systemen, wie Flugzeugen oder Automobilen, zu immer höheren *Sicherheitsanforderungen* an Software. Der starke Anstieg der durchschnittlichen Code-Größen und die Langlebigkeit von Software-Systemen machen die *Wartbarkeit* zu einer wichtigen Eigenschaft. Die Beherrschung von Software-Qualität stellt somit ein wichtiges Ziel im Software Engineering dar.

Diesem Ziel steht aber die Komplexität und Vielschichtigkeit von Qualität gegenüber. Es existiert eine große Zahl an unterschiedlichen Sichten und eine entsprechende Vielzahl an Herangehensweisen zu diesem Thema. Für die praktische Anwendung in der Software-Entwicklung stehen aufgrund dieser Vielfalt überwiegend nur Insellösungen zur Verfügung, die keine ganzheitliche Behandlung des Themas ermöglichen. Beispielsweise sind trotz der engen Verbindung Bewertungen von Zuverlässigkeit und Nutzbarkeit typischerweise nicht integriert.

Ein verbreitetes Vorgehen zur Bewältigung dieser Probleme stellt die Verwendung von Qualitätsmodellen und daraus abgeleiteter bzw. damit in Beziehung gesetzter Bewertungen dar. Ein solches Vorgehen wird sowohl in der Forschung untersucht, als auch bereits in der Praxis angewendet. Es hat sich aber oft gezeigt, dass Standards, wie die ISO 9126, nicht direkt anwendbar sind und eigene Qualitätsmodelle für spezifische Situationen erstellt und empirisch abgesichert werden müssen. Dies resultiert in teilweise sehr unterschiedlichen Ansätzen zur Qualitätsmodellierung und -bewertung.

Dieser Workshop, der zusammen mit der Konferenz SE 2008 stattfindet, bringt Interessierte zu diesen Themen aus dem industriellen und akademischen Umfeld zusammen, um die aktuellen Probleme und Lösungsansätze zu diskutieren. Ziel dieses Workshops ist es, die existierenden Ansätze vorzustellen und zu diskutieren. Idealerweise soll daraus ein gemeinsamer Forschungsplan für zukünftige Entwicklungen in der Qualitätsmodellierung resultieren. Dies ist besonders wichtig im Hinblick auf die *SQuaRE*-Reihe neuer Qualitätsnormen der ISO, an denen derzeit gearbeitet wird.

April 2008

Stefan Wagner, Manfred Broy, Florian Deißenböck,
Peter Liggesmeyer, Jürgen Münch
Organisatoren
SQMB '08

# Organisation

Der Workshop SQMB '08 wurde in Zusammenarbeit der Technische Universität München und des Fraunhofer IESE organisiert. Der Workshop fand im Zusammenhang mit der Konferenz SE 2008 statt.

## Organisatoren

Stefan Wagner, Technische Universität München
Manfred Broy, Technische Universität München
Florian Deißenböck, Technische Universität München
Peter Liggesmeyer, Fraunhofer IESE
Jürgen Münch, Fraunhofer IESE

## Programmkomitee

Ralf Ackermann
Thomas Beil
Manfred Broy
Horst Degen-Hientz
Axel Dold
Florian Deißenböck
Reiner Dumke
Gregor Engels
Heike Frank
Christian Körner
Claus Lewerentz
Peter Liggesmeyer
Oliver Mäckel
Jürgen Münch
Dietmar Pfahl
Markus Pizka
Reinhold Plösch
Ralf Reussner
Wilhelm Schäfer
Kurt Schneider
Andy Schürr
Dirk Voelz
Stefan Wagner
Andreas Zeller
Rolf Ziegler

# Inhaltsverzeichnis

# Software-Qualitätssicherung gestern und heute: Theorie und Erfahrung, Standards und Common Sense

Peter Liggesmeyer

Fraunhofer IESE,
Fraunhofer-Platz 1, 67663 Kaiserslautern

**Zusammenfassung** Wer heute Software-Qualitätssicherung verantwortet, diese konzipieren und steuern muss, ist „eingeklemmt“ zwischen einem Mangel an Theorie, der hohen Bedeutung der Prüfung und der unvermeidlichen Ressourcenknappheit der Praxis. Standards und empirisch abgesicherte Erkenntnisse zu Themen, die sich rein theoretisch nur schwierig erschließen lassen, können Hilfestellungen bieten. Der Vortrag beleuchtet die unterschiedlichen Kriterien, nach denen die Software-Qualitätssicherung heute beurteilt werden muss, und zeigt Wege zum Ziel auf.

# 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

**Zusammenfassung** 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).

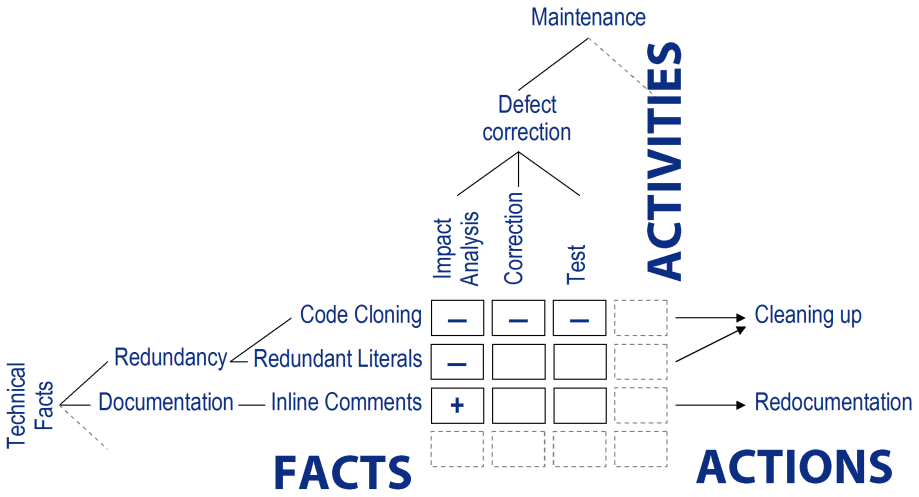


**Abbildung 1.** A simplified extract of the quality model

**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

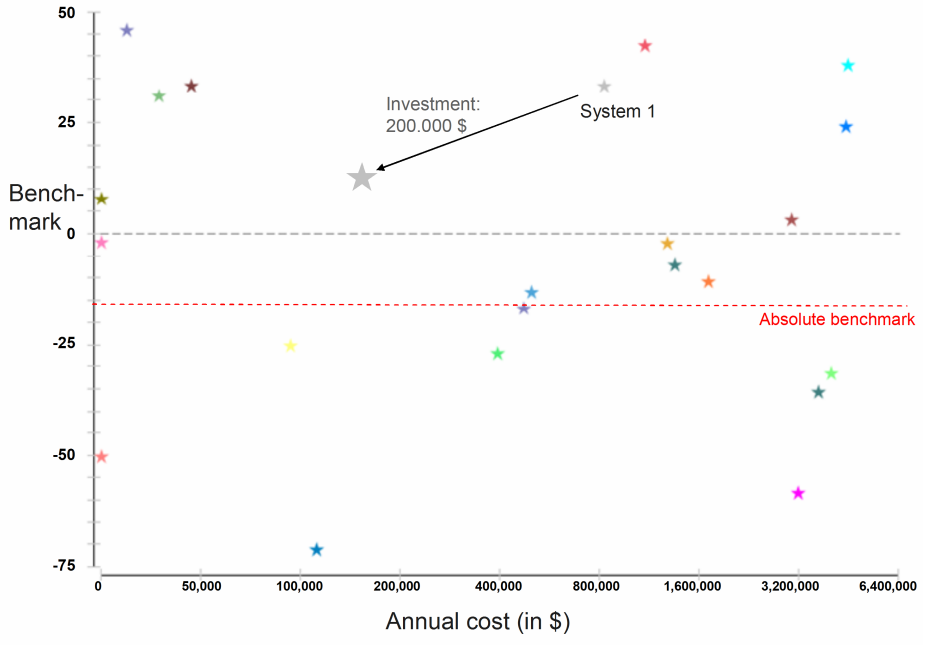


**Abbildung 2.** A sample benchmark analysis. The arrow indicates the technical and the resulting economic improvement opportunity.

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.

## Literatur

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 (October 2006)
5. Johnson, R.E.: Software Reengineering Assessment Handbook, Version 3.0. US Department of Defense (March 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)

# Software-Qualitätsmodelle in der Praxis: Erfahrungen mit aktivitätenbasierten Modellen

Stefan Wagner, Florian Deißenböck, Martin Feilkas, Elmar Jürgens

Institut für Informatik
Technische Universität München
{wagnerst,deissenb,feilkas,juergens}@in.tum.de

**Zusammenfassung** Software-Qualität ist ein komplexes und vielschichtiges Gebiet, das eine hohe und sich ständig verstärkende Praxis-Relevanz aufweist. Dieser Komplexität wird üblicherweise mit Qualitätsmodellen begegnet, die Qualität in Attribute und Subattribute aufspalten. Die Modelle haben aber in der Anwendung verschiedene Probleme, wodurch sie sich nicht allgemein durchsetzen lassen. Eine neuartige Strukturierung von Qualitätsmodellen basiert auf der Trennung von technischen Eigenschaften und Aktivitäten. Das Papier faßt die Vorteile dieser Art von Qualitätsmodellen zusammen und stellt die umfangreichen industriellen Erfahrungen damit kurz dar.

## 1 Qualitätsmodelle für Software

Qualität ist grundsätzlich ein komplexes und vielschichtiges Konzept [1]. Aufgrund ihrer großen praktischen Relevanz müssen aber trotzdem Mittel und Wege gefunden werden, mit dieser Komplexität umzugehen.

### 1.1 Motivation und Probleme

Qualitätsmodelle beschreiben abstrakt, was Qualität bei Software-Systemen bedeutet. Dazu wird Qualität typischerweise in verschiedene Faktoren zerlegt, wie beispielsweise in der ISO-Norm 9126 [2]. Solche hierarchischen Strukturen als Basis für Qualitätsmodelle sind bereits seit langem im Einsatz. Ein Modell, das zuerst von Boehm et al. [3] und McCall et al. [4] benutzt wurde, besteht aus drei Schichten: Faktoren, Kriterien und Metriken. Die Faktoren auf der obersten Hierarchieebene modellieren die Hauptqualitätsziele. Diese Faktoren werden in Kriterien und Subkriterien weiter zergliedert. Typischerweise werden dabei drei Ebenen definiert, wobei für die unterste Ebene eine Metrik zur Messung des Kriteriums angegeben wird. Jedoch ist die genutzte Dekomposition oft zu abstrakt und unpräzise, um in der Analyse und der Messung direkt einsetzbar zu sein.

Darüber hinaus fehlt in solchen Modellen typischerweise der Bezug zu den Aktivitäten im Entwicklungsprozess und somit zu Kosten und Nutzen. Auch neuere Arbeiten zur Messung und Modellierung von Software-Qualität, wie beispielsweise Seffah et al. [5] ändern an den genannten Kritikpunkten nichts.

Zusammenfassend weisen Qualitätsmodelle heute typischerweise vier Probleme auf:

1. *Bewertbarkeit.* Die meisten Qualitätsmodelle besitzen viele Kriterien, die zu grob-granular sind, als dass sie direkt bewertet werden könnten.
2. *Begründung.* Oft fehlt auch für viele Kriterien die Erklärung des Einflusses, den sie auf die Entwicklung, den Betrieb oder die Nutzung des Systems haben.
3. *Homogenität.* Da eine eindeutige Aufteilung in verschiedene Qualitätsaspekte fehlt, besitzen die meisten Qualitätsmodelle heute Qualitätskriterien, die teilsweise nicht disjunkt sind und auf sehr unterschiedlichen Abstraktionsebenen liegen.
4. *Operationalisierung.* Typische Qualitätsmodelle existieren nur in Form von Text und Graphiken und sind kein lebendiger Teil des Entwicklungsprozesses.

### 1.2 Aktivitätenbasierte Qualitätsmodelle

Aus den genannten Problemen ist ersichtlich, dass es notwendig ist die Strukturierung von Qualitätsmodellen zu verbessern. Zwei der Probleme, ungenügende Begründung und Homogenität, stammen aus der nicht ausreichenden Struktur gängiger Modelle. Aus diesem Grund wird in [6] und [7] vorgeschlagen, ein explizites Metamodell für Qualitätsmodelle zu verwenden. Dieses Metamodell trennt technische Eigenschaften von den auf und mit dem System auszuführenden Aktivitäten. Weiterhin werden die Einflüsse der Systemeigenschaften auf diese Aktivitäten explizit modelliert. Dies resultiert in einer „Qualitätsmatrix“, wie sie in Abb. 1 dargestellt ist.

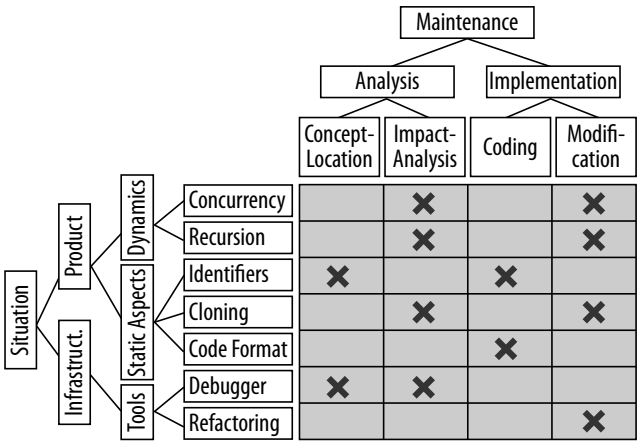


**Abbildung 1.** Eine beispielhafte Qualitätsmatrix für Wartungsaktivitäten

Ein Kreuz zeigt hier den Einfluss von bspw. „Cloning“ auf die Aktivität „Modification“ an. Typischerweise werden solche Einflüsse noch weiter spezifiziert,

z.B. in positive und negative Einflüsse. Dies (zusammen mit textuellen Beschreibungen) reicht bereits aus, um Review-Richtlinien zu generieren. Für weitere Analysen können Qualitäts-Cockpits, wie z.B. ConQAT[1], angebunden werden. Das Modell selbst kann in einem eigens dafür entwickelten Editor bearbeitet werden und liegt als XML-Datei vor. Dadurch wird das Modell zu einem lebendigen Artefakt in der Entwicklung.

### 1.3 Modellbasiertes Qualitätscontrolling

Um den oben genannten Problemen gebräuchlicher Qualitätsmodelle zu begegnen, ist die stärkere Operationalisierung von Qualitätsmodellen als lebendiges Artefakt im Software-Entwicklungsprozess wichtig [6]. Dies wird auch „modellbasiertes Qualitätscontrolling“ genannt, wobei hier nicht klassische Software-Modelle im Sinne der UML gemeint sind, sondern ein Qualitätsmodell die Basis von Qualitätsanalysen darstellt (Abb. 2).

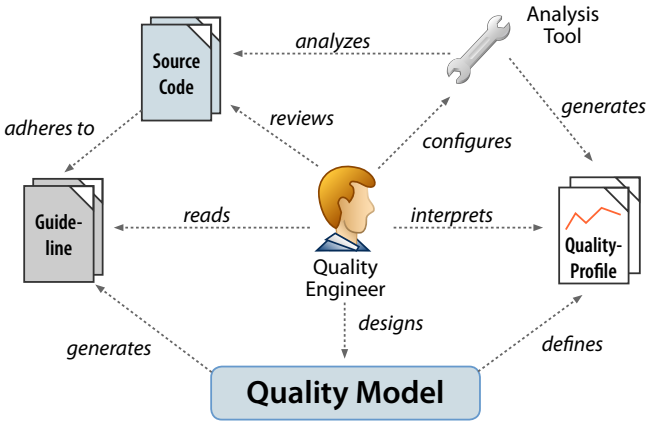


**Abbildung 2.** Modellbasiertes Qualitätscontrolling

Im Zentrum steht ein „Qualitäts-Ingenieur“, der ein Qualitätsmodell entwirft und auf dieser Basis Richtlinien generiert und Qualitätsprofile definiert. Weiterhin setzt er Analysewerkzeuge ein, um Quellcode und andere Artefakte zu analysieren und deren Konformität mit dem Qualitätsmodell zu prüfen.

## 2 Praxiserfahrungen

Der aktivitätenbasierte Ansatz zur Qualitätsmodellierung wurde inzwischen in einer Reihe von Industriekooperationen eingesetzt. Die wichtigsten Beispiele sind die folgenden drei.

[1] `http://conqat.in.tum.de/`

*Siemens SBS* Die Siemens Business Services arbeiten an betrieblichen Informationssystemen, die teilweise sehr lange Einsatzzeiten vorweisen können. Gerade bei diesen langlebigen Systemen ist eine strukturierte Bewertung der Wartbarkeit von großer Bedeutung. In Zusammenarbeit mit SBS sind die Grundstrukturen unserer Qualitätsmodellierung entstanden [8]. Es wurden Richtlinien gesammelt, vervollständigt und Inkonsistenzen und Widersprüche aufgedeckt.

*MAN Nutzfahrzeuge* Die MAN Nutzfahrzeuge ist ein internationaler Hersteller von Lastkraftwagen und Bussen mit eigener Elektronik- und Software-Entwicklung. Dort wird ein Qualitätsmodell eingesetzt, dass vor allem die Qualität von Matlab/Simulink und Stateflow abdeckt [6]. Aus diesem Qualitätsmodell, werden Review-Richtlinien generiert, die direkt im Entwicklungsprozess eingesetzt werden. Bei der Erstellung des Modells wurden wiederum in bestehenden Richtlinien Inkonsistenzen und Widersprüche entdeckt.

*Münchener Rück* Die Münchener-Rück-Gruppe gehört weltweit zu den größten Rückversicherern und entwickelt und betreibt eine Vielzahl betrieblicher Informationssysteme zur Unterstützung ihrer Geschäftsprozesse. In einer Kooperation mit der Münchener Rück ist ein Qualitätsmodell entstanden, das Programmierrichtlinien und Qualitätskriterien für diese betrieblichen Informationssysteme beschreibt. Das Qualitätsmodell bildet die Grundlage von statischen Code-Analysen, die automatisiert zentrale Qualitätskriterien überprüfen. Die Qualitätsanalysen werden mittels ConQAT in den Entwicklungsprozess integriert und unterstützen dadurch Reviews und ermöglichen ein kontinuierliches Qualitäts-Controlling.

In allen praktischen Anwendungen hat es sich aber gezeigt, dass das Erstellen und Pflegen solcher Modelle aufwändig ist. Ein operationalisiertes Modell, das zur Ableitung von Analysen verwendet wird, muss entsprechend umfangreich sein, um die verschiedenen Qualitätsfacetten abbilden zu können. Es ist also ein Kosten/Nutzen-Verhältnis abzuwägen. Dies ist nach unseren Erfahrungen positiv, hängt aber stark von (1) der Werkzeugunterstützung und (2) der Einbindung in den Entwicklungsprozess ab. Speziell bei MAN hat sich die Verwendung des Modells als vorteilhaft gezeigt, da sowohl automatische Analysen als auch Review-Richtlinien abgeleitet werden und auch Messungen auf dieser Basis geplant sind.

## 3 Zusammenfassung

Die strukturierte Definition von Software-Qualität in Qualitätsmodellen ist essentiell für ihre Handhabung in der Praxis. Nur ein Modell, das als operationalisiertes Artefakt während der Entwicklung auch weiterentwickelt wird, kann langfristig Qualität sicherstellen. Aus den gemachten Erfahrungen in den industriellen Anwendungen kann geschlossen werden, dass dies möglich ist. Wichtig ist aber, dass sowohl eine sinnvolle Struktur, wie auch eine umfangreiche Werkzeugunterstützung vorhanden ist. Die Strukturierung auf Basis von Aktivitäten

hat sich in der Praxis bewährt, weil die Aktivitäten als Hauptkostentreiber eine direkte Verbindung zu ökonomischen Bewertungen ermöglichen. Weiterhin hat sich herausgestellt, dass ein Qualitätsmodell nur erfolgreich sein kann, wenn es sich stark auf die jeweiligen Bedürfnisse der Domäne und Technologien zuschneiden lässt.

## Literatur

1. Garvin, D.A.: What does product quality really mean? MIT Sloan Manage. Rev. **26**(1) (1984) 25–43
2. ISO: ISO 9126: Product Quality – Part 1: Quality Model (2003)
3. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., Macleod, G.J., Merrit, M.J.: Characteristics of Software Quality. North-Holland (1978)
4. McCall, J.A., Richards, P.K., Walters, G.F.: Factors in software quality. Reports NTIS AD/A-049 014, 015, 055, US Rome Air Development Center (1977)
5. Seffah, A., Donyaee, M., Kline, R.B., Padda, H.K.: Usability measurement and metrics: A consolidated model. Software Quality Control **14**(2) (2006) 159–178
6. Deissenboeck, F., Wagner, S., Pizka, M., Teuchert, S., Girard, J.F.: An activity-based quality model for maintainability. In: Proc. 23rd International Conference on Software Maintenance (ICSM '07), IEEE Computer Society Press (2007)
7. Winter, S., Wagner, S., Deissenboeck, F.: A Comprehensive Model of Usability. In: Proc. Engineering Interactive Systems 2007 (EIS '07), Springer (2008) Zur Veröffentlichung angenommen.
8. Broy, M., Deissenboeck, F., Pizka, M.: Demystifying maintainability. In: Proc. 4th Workshop on Software Quality (4-WoSQ), ACM Press (2006) 21–26

# Kostenaspekte bei der Modellierung von Softwarequalität

Lars Karg, Dirk Voelz

SAP Research, CEC Darmstadt, Germany
{Lars.Karg, Dirk.Voelz}@sap.com

**Zusammenfassung** Mit zunehmender Reife der Softwarebranche und einem verstärkten Wettbewerb gewinnt Softwarequalität weiterhin an Bedeutung. Wissenschaftler und Softwarehersteller reagieren auf diesen Trend mit der Entwicklung von Qualitätsmodellen, die möglichst viele Aspekte des Qualitätsmanagements berücksichtigen und gleichzeitig den Erfordernissen der Praxis gerecht werden müssen. Dieses Positionspapier kritisiert, dass Kostenaspekte bei der Modellierung und Bewertung von Qualität unterrepräsentiert sind. Dabei identifizierten wir drei Bereiche, die besondere Aufmerksamkeit erfordern: (1) Verständnis der Zusammenhänge zwischen Kostengrößen und spezifischen Unternehmensfaktoren, (2) Verfügbarkeit und Qualität von Kostendaten in Entwicklungsorganisationen und (3) umfassende Validierung der gewonnen Erkenntnisse.

## 1 Einleitung

Seit Jahrzehnten leiden Anwender von Softwarelösungen unter Qualitätsproblemen [1]. Dieser Herausforderung versuchen Softwarehersteller durch Adaption von Konzepten reiferer Industrien zu begegnen [2]. Dies ist jedoch ohne Anpassung an die Besonderheiten der Softwareentwicklung zum Scheitern verurteilt [3]. In jüngster Zeit wird verstärkt Hoffnung in die Industrialisierung der Softwareentwicklung gesetzt [4]. Neben der Einführung und zielgerichteten Adaption von Konzepten reiferer Industrien ist die ganzheitliche betriebswirtschaftliche Sicht auf die Softwareentwicklung im Fokus der Betrachtung.

Lange Zeit wurde die strategische Bedeutung von Qualität für den Unternehmenserfolg unterschätzt. Dies lässt sich durch die sehr funktional geprägte Sichtweise in der Softwareentwicklung sowie durch herstellerfreundliche Haftungsbestimmungen bei Qualitätsproblemen erklären. Zur Erzielung einer hinreichenden Qualität versuchen Softwarehersteller daher verstärkt mittels Qualitätsmodellen die Zusammenhänge zwischen Qualität und Unternehmensfaktoren zu verstehen und zu modellieren. Kostenüberlegungen nehmen hierbei eine zentrale Rolle ein. Nur durch Berücksichtigung dieser besteht die Möglichkeit der wirtschaftlichen Steuerung des Qualitätsmanagements. Ein übergreifendes Modell, das sämtliche der genannten Aspekte berücksichtigt und gleichzeitig den Erfordernissen der Praxis gerecht wird, existiert jedoch nicht, und es ist fraglich, ob dies je der Fall sein wird.

Dieses Positionspapier kritisiert, dass Kostenaspekte bei der Modellierung und Bewertung von Software Qualität nur unzureichend Berücksichtigung finden. Hierzu wird zunächst auf die Modellierung von Software Qualität im Allgemeinen eingegangen (Abschnitt 2), bevor im Anschluss die existierenden Forschungslücken bei der Integration von Kostenaspekten in die Qualitätsmodellierung aufgezeigt werden (Abschnitt 3). Abschließend werden die gewonnenen Erkenntnisse zusammengefasst (Abschnitt 4).

## 2 Modellierung von Software-Qualität

Softwarequalität hängt von vielen Einflussfaktoren ab und muss daher unabhängig von dem betrachteten Artefakt (wie Prozess oder Produkt) als multi-dimensionales Konstrukt verstanden werden [5]. Obwohl dieser Sachverhalt empirisch nachgewiesen wurde, wird er bis heute in der Forschung häufig vernachlässigt [6]. Die Ausprägung der einzelnen Dimensionen, in ISO 9126 wird von Charakteristiken gesprochen, hängt hierbei von den Anforderungen der Projektbeteiligten und dem Artefakt selbst ab. Bei einer Softwarelösung, die als Dienst angeboten wird, sind dies eher Dimensionen der Dienstleistungsqualität [7]. Bei einem traditionellen Softwareprodukt sind hingegen Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, nderbarkeit und bertragbarkeit mögliche Qualitätsdimensionen [8]. Die Projektbeteiligten definieren ihre Anforderungen, die mittels der Qualitätsdimensionen abgedeckt und durch Metriken auf Erfüllung hin überprüft werden. Das auf Basis der Anforderungen entwickelte Qualitätsmodell (z.B. nach ISO 9126) kann hierbei sowohl der Bewertung und Steuerung der Qualität, als auch in einer Vorstufe ausschließlich der Entwicklung eines Verständnisses der Zusammenhänge dienen. In unseren Ausführungen beziehen wir uns auf diesen zweiten Aspekt, da er unserer Meinung nach die Basis jedes guten Qualitätsmodells bildet. Neben beispielsweise einem einfachen Tailorings des Qualitätsmodell, das die Anwendbarkeit auf verschiedenen Granularitäten und in verschiedenen Domänen sicherstellt, bilden Kostenaspekte einen zentralen Bestandteil der Softwareherstellung [9]. Sie sind in der Forschung bis heute nur unzureichend untersucht [10], obwohl sie einen zentralen Baustein bei der wirtschaftlichen Betrachtung und Bewertung von Qualität bilden. Kosten der Nichterreichung und zusätzliche Aufwände zur nachträglichen Anforderungserfüllung bleiben meist unberücksichtigt oder sind teilweise sogar unbekannt. Im Folgenden wird auf diese finanzielle Perspektive eingegangen und es werden bestehende Forschungslücken benannt [11].

## 3 Kostenaspekte bei der Qualitätsmodellierung

Kostenaspekte spielen bei der Modellierung von Qualität ein zentrale Rolle (siehe Abschnitt 2) und sind in reiferen Industrien umfassend untersucht worden [12]. Abhängig vom gewählten Qualitätsmodell erlauben diese Aussagen zu finanziellen Auswirkungen bei Nichterfüllung der Qualitätsziele. Hierzu ist ein umfassendes Verständnis der einzelnen Kosten- und Einflussgrößen notwendig. Hierbei

lassen sich drei Kostengrößen unterscheiden: *Prevention, Appraisal* und *Failure Costs* [13]. Ausgehend vom multi-dimensionalen Konstrukt Qualität (siehe Abschnitt 2) lassen sich *Prevention Costs* als die Kosten auffassen, die zur Vorbeugung von Anforderungsabweichungen aufgewendet werden. *Appraisal Costs* sind die Kosten, die bei der berprüfung der Anforderungserreichung und Identifikation von Abweichungen anfallen. *Failure Costs* resultieren aus den Kosten, die durch die verbleibenden Abweichungen verursacht werden. In den gängigen Qualitätsmodellen finden diese Kostengrößen jedoch keine Beachtung. Um die Kostenaspekte dennoch hinreichend abdecken zu können, sehen wir in folgenden drei Bereichen Forschungsbedarf:

- Die Verfügbarkeit und Qualität von quantitativen Kostendaten ist unzureichend und wenn vorhanden zu grobgranular. Hier ist ein höherer Detaillierungsgrad erforderlich. Insbesondere gilt dies für die Verfügbarkeit von zusammenhängenden Datensätzen. In der Literatur lassen sich eine Vielzahl deskriptiver Arbeiten [14] zu einzelnen Kostengrößen finden; übergreifende, zu mehreren Kostengrößen, fehlen hingegen größtenteils. Dies ist jedoch Voraussetzung zur Modellierung und Identifikation der Einflussgrößen und Zusammenhänge. In der Konsequenz sind viele Unternehmen heute nicht in der Lage zu beziffern, was sie ein Fehler kostet.
- Je nach Kostengrößen ist eine unterschiedlich gute Abdeckung seitens der Forschung gegeben. *Appraisal Costs* wurden auf feingranularer Ebene bereits umfassend untersucht, insbesondere die Kosten und Wirkungszusammenhänge häufig eingesetzter Qualitätssicherungstechniken, wie Inspektionen [15] sowie Testautomatisierung und Regressionstests [16]. Auf höherer Ebene, wie Projekt oder Programm, hingegen besteht Nachholbedarf. Dies gilt ebenfalls für *Prevention* und *Failure Costs*. Eine verstärkte Erforschung sämtlicher Kostengrößen ist insbesondere im Hinblick auf die wirtschaftliche Modellierung von Qualität wünschenswert.
- Zusätzlich ist im größeren Umfang eine Cross- und Revalidierung der gewonnen Erkenntnisse erforderlich. Bekannte Kostenzusammenhänge, wie die Zunahme der Fehlerbehebungskosten in späteren Entwicklungsphasen [17], bedürfen einer fortlaufenden Revalidierung bedingt durch den anhaltenden Wandel der Entwicklungsprozesse durch technische Innovationen. Ein Verzicht birgt die Gefahr, ersten Modellprototypen blind zu folgen und die tatsächlichen Wirkungszusammenhänge nicht zu erfassen.

Durch verstärkte Erforschung der genannten Bereiche können die Auswirkungen unzureichender Qualität besser verstanden werden. Aus diesem Wissen lassen sich Maßnahmen ableiten, wie die im Qualitätsmodell definierten Ziele wirtschaftlich zu erreichen sind. Mit dem Aufkommen des wertorientierten Software Engineerings [9] und einem verstärkten Wettbewerb, bei dem Funktionalität nicht mehr das entscheidende Verkaufsargument ist, nehmen Kostenüberlegungen eine zentrale Rolle ein. Die Bedeutung des Kostenaspektes wird mit der Marktdurchdringung des Software-as-a-Service-Ansatzes weiter steigen. Nur durch die Berücksichtigung der Kosten bei der Qualitätsmodellierung besteht die Möglichkeit einer wirtschaftlichen Steuerung des Qualitätsmanagements.

## 4 Zusammenfassung und Ausblick

Unternehmen, die sich in den kommenden Jahren der Herausforderung neuer Qualitätsmodelle stellen, haben gute Chancen, in Zeiten industrialisierter Softwareherstellung wettbewerbsfähig zu bleiben. Erste Ansätze zur Entwicklung von umfassenderen Qualitätsmodellen sind im Gange. Anwendung und Nutzen werden primär davon abhängen, inwiefern die Adressierung der drei Bereiche (1) Verfügbarkeit und Qualität von Daten als primäre Herausforderung für die Praxis sowie (2) Verständnis von Zusammenhängen zwischen Kostengrößen und spezifischen Unternehmensfaktoren sowie (3) umfassende Validierung der gewonnen Erkenntnisse als primäre Herausforderungen für die Theorie erfolgreich gelingt.

## Literatur

1. Whittaker, J.A. und Voas, J.M.: 50 Years of Software: Key Principles for Quality. IT Pro Nov/Dec (2002) 28-35
2. Antony, J. und Fergusson, C.: Six sigma in the software industry: results from a pilot study. Managerial Auditing Journal 19 (2004) 1025-1032
3. Basili, V.C. und Caldiera, G.: Improve Software Quality by Reusing Knowledge and Experience. Sloan Management Review 37 (1995) 55-64
4. Kilian-Kehr, R.; Terzidis, O. und Voelz, D.: Industrialisation of the Software Sector. Wirtschaftsinformatik 49 (2007) S62-S71
5. Usrey, M. und Dooley, K.: The Dimensions of Software Quality. Quality Management Journal 3 (1996) 67-86
6. Stone-Romero, E.; Stone, D. und Grewal, D.: Development of a multi-dimensional measure of perceived product quality. Journal of Quality Management 2 (1997) 87-111
7. Parasuraman, A.; Zeithaml, V.A. und Berry, L.L.: SERVQUAL: A Multiple-Item Scale for Measuring Consumer Perceptions of Service Quality Journal of Retailing 64 (1988) 12-37
8. ISO/IEC: Software engineering - product quality - part 1: quality model. (2001)
9. Biffl, S.; Aurum, A.; Boehm, B.; Erdogmus, H. und Grnbacher, P.: Value-Based Software Engineering. Springer (2005)
10. Karg, L.: Ansätze und Modelle zur Qualitätskostenbestimmung im Leistungserstellungsprozess der Softwarebranche. SAP Research (2007)
11. Horngren, C.T.; Datar, S.M. und Foster, G.: Cost Accounting - A Managerial Emphasis. Pearson Prentice Hall, New Jersey (2006)
12. Williams, A.R.T.; Wiele, A.v.d. und Dale, B.G.: Quality costing: a management review. International Journal of Management Reviews 1 (1999) 441-460
13. Pressman, R.: Software Engineering: A Practitioner's Approach. McGraw Hill Higher Education (2004)
14. Wagner, S.: A Literature Survey of the Software Quality Economics of Defect-Detection Techniques. Technische Universitt Mnchen, Mnchen (2006) 50
15. Freimut, B.; Briand, L.C. und Vollei, F.: Determining Inspection Cost-Effectiveness by Combining Project Data and Expert Opinion. IEEE Transactions on Software Engineering 31 (2005) 1074-1092
16. Elbaum, S.; Malishevsky, A.G. und Rothermel, G.: Test Case Prioritization: A Family of Empirical Studies. IEEE Transactions on Software Engineering 28 (2002) 159-182
17. Boehm, B.W.: Software Engineering Economics. Prentice-Hall (1981)

# Ein Kosten-Nutzen-Modell für Prüfungen

Tilmann Hampp, Jochen Ludewig

Institut für Softwaretechnologie, Universität Stuttgart
{hampp, ludewig}@informatik.uni-stuttgart.de

**Zusammenfassung** Prüfungen in Softwareprojekten sind teuer, der Verzicht darauf kann aber langfristig noch teurer sein. In diesem Artikel wird ein Modell für den Kosten-Nutzen-Vergleich vorgestellt.

## 1 Problem und Ziel

Prüfungen, d. h. Reviews und Tests, können auf verschiedene Weise und mehr oder minder intensiv durchgeführt werden. Sie erhöhen (zunächst) Aufwand, Dauer und Personalbedarf des Softwareprojekts. Dem stehen Verbesserungen der Qualität und Einsparungen in späteren Phasen gegenüber. Die Nachteile (d. h. die Kosten) der Prüfungen werden sofort, die Vorteile (Einsparungen) werden erst später sichtbar. Die Einsparungen sind schwer vorherzusagen und zu messen, ein subjektiver Eindruck der Qualität kann nicht quantifiziert werden. Der Projektleiter steht also bei der Prüfplanung und -kontrolle [9] vor einer schwierigen, unübersichtlichen Aufgabe. Wie werden sich seine Entscheidungen auswirken? Steht dem Projektleiter aber ein quantitatives Modell zur Verfügung, so kann er Kosten und Nutzen aus dem Modell ableiten. Ebenso lassen sich auf der Basis eines solchen Modells Aufwand, Dauer und Personalbedarf prognostizieren.

## 2 Verwandte Arbeiten

Die Modelle in [8, 16, 18] bewerten Reviewnutzen durch vermiedene Korrekturkosten. Schwinn [18] fokussiert die Auswahl des Prüflings. Rico [15] betrachtet Prozessverbesserungen einzeln. Wagner [19] bewertet Kosten und Nutzen von Prüfungen einschließlich der Fehlerfolgekosten. Boehm et al. [5] klassifiziert den Prüfprozess und schätzt Dauer, Aufwand und Personalbedarf einzelner Phasen; das Produkt wird anhand erzielter Zuverlässigkeit und der Geschäftsziele bewertet. Anders als diese Arbeiten unterstützt das folgende Modell konkrete Entscheidungen zum Vorgehen und zur Intensität von Prüfungen. Fehlerfolgekosten werden detailliert geschätzt, Auswirkungen auf einzelne Aktivitäten betrachtet.

## 3 Das Modell

### 3.1 Grundlagen

Kosten werden durch anfallenden Aufwand, Dauer und Personalbedarf dargestellt. Der Nutzen wird durch vermiedenen Aufwand, Dauer, Personalbedarf und

vermiedene Fehlerfolgekosten dargestellt (Tabelle 1). Kosten und Nutzen werden auf Geldwerte abgebildet. Der Aufwand wird durch COCOMO-II-Faktoren und die Produktgröße beeinflusst, er bestimmt Dauer und Personalbedarf [4]. Die Zahl der zu entdeckenden Fehler hängt von der Produktgröße ab. Jede Prüfung entdeckt einen Anteil der Fehler [6]. Die Korrekturkosten hängen von Fehlerzahl, -typ, -schwere und Entdeckungszeitpunkt ab [6, 11, 14]; die späte Korrektur ist teuer. Blockierende Fehler wiegen besonders schwer, denn sie unterbrechen die Testsequenz, so dass Testfälle der Sequenz wiederholt werden müssen [2, 14]. Organisatorischer Aufwand wird durch Zuschläge berücksichtigt.

**Tabelle 1.** Kosten und Nutzen von Prüfungen

| Tätigkeiten, um Fehler frühzeitig zu erkennen und zu beseitigen: | Tätigkeiten und Kosten, die entfallen, wenn Fehler frühzeitig erkannt und beseitigt werden: |
|---|---|
| • Organisation der Prüfungen<br>• Vorbereitung der Prüfungen<br>• Durchführung der Prüfungen<br>• Auswertung der Prüfungen<br>• Fehlersuche<br>• Fehlerbeseitigung | Falls Fehler in späteren Prüfungen erkannt werden:<br>• Organisation (Fehlerverwaltung)<br>• Fehlersuche<br>• Fehlerbeseitigung (inkl. propagierte Änderungen)<br>• Testwiederholung nach blockierenden Fehlern<br>• Test nach der Fehlerbeseitigung<br>Falls Fehler erst im Betrieb auffallen:<br>• Fehlerfolgekosten<br>• Korrektive Wartung |

Das Modell gestattet es, Kosten und Nutzen von Spezifikations-, Entwurfs- und Codereviews und des Systemtests zu schätzen. Entscheidungsoptionen werden als Parameter dargestellt, um verschiedene Prüfkombinationen mit unterschiedlicher Intensität zu vergleichen.

### 3.2 Modellparameter

Als Prüfungen können Spezifikations-, Entwurfs- oder Codereviews, Unit-, Integrations-, System- oder Feldtest kombiniert werden. Für Reviews werden Zahl, Eignung und Vorbereitungsintensität der Gutachter und die Überdeckung gewählt. Für den Systemtest werden die Testmethoden gewählt: Funktionstest, Äquivalenzklassentest oder Test weiterer Sonderfälle im Black-Box-Test, im ergänzenden Glass-Box-Test die zu erzielende Überdeckung von Anweisungen, Zweigen, Termen (MC/DC-Überdeckung) und Schleifen (Boundary-Interior-Überdeckung), der Zeitpunkt der Testvorbereitung [10] und die Kompetenz der Tester. Die projektabhängigen Fehlerfolgekosten werden wie Risiken bewertet, abhängig von der Wahrscheinlichkeit, dass der Fehler zu einem Fehlverhalten bei einer Verwendung führt, von dem Schaden, den das Fehlverhalten verursacht, und von der Verwendungshäufigkeit bis zur Fehlerkorrektur. Sie berechnen sich aus $Schaden \times Wahrscheinlichkeit \times Verwendungshäufigkeit$. Fehler werden auf grobe Klassifikationen für diese drei Größen verteilt. Basisparameter sind die

COCOMO-II-Faktoren [4], die Produktgröße in Function Points und die Programmiersprache.

Reviews werden mit Moderator, Autor, Notar und Gutachtern [7] modelliert. Die Zahl der Gutachter, ihre Eignung [3] und die Intensität der Vorbereitung bestimmen Aufwand und Fehlerentdeckung; nur mit gründlicher Vorbereitung wird ein hoher Fehleranteil entdeckt. Im Systemtest kann ein Teil der Fehler jeweils nur im Black-Box-Test, nur im Glass-Box-Test oder in beiden Tests entdeckt werden [1, 13]. Im Black-Box-Test werden Fehler abhängig von den gewählten Methoden und ihrer Vollständigkeit entdeckt. Der Aufwand wächst linear mit der Testvollständigkeit. Erfolgt die Vorbereitung in den frühen Phasen, werden Fehler früh entdeckt und korrigiert. Weil der Black-Box-Test Code überdeckt, werden im Glass-Box-Test Fehler abhängig von den bereits erreichten und zu erzielenden Überdeckungen entdeckt. Der Aufwand wächst überproportional mit der Überdeckung, abhängig vom Code-Umfang. Kompetente Tester finden mit weniger Aufwand mehr Fehler [1].

### 3.3 Ein Beispiel

Die Berechnung der Modellresultate aus den Eingabeparametern zeigen wir im Folgenden mit einem vereinfachten Beispiel. Anders als im Modell werden im Beispiel die Fehler nicht nach Fehlerschwere unterschieden, und die Zahlen sind gerundet. Im Beispiel hat das Produkt einen Umfang von 200 Function Points. Das Projekt wird unter typischen Randbedingungen durchgeführt, die COCOMO-II-Einflussfaktoren sind darum auf den Nominalfall eingestellt. Es werden die folgenden Prüfungen gewählt: Ein Spezifikationsreview und ein Entwurfsreview finden jeweils mit drei geeigneten Gutachtern, die sich gründlich vorbereiten, statt. Modultest, Integrationstest, Systemtest und Feldtest werden durchgeführt.

Das Modell berechnet aus 200 Function Points Umfang 640 zu entdeckende Fehler [11]. Nach Spezifikations- und Entwurfsreview, nach Modul- und Integrationstest sind noch 294 Fehler nicht entdeckt (Tabelle 2).

**Tabelle 2.** Entdeckbare und entdeckte Fehler im Sytemtest

| | Spezifikationsfehler | Entwurfsfehler | Codefehler | Summe Fehler |
|---|---|---|---|---|
| Insgesamt zu entdeckende Fehler | 160 | 200 | 280 | 640 |
| Vor dem Systemtest entdeckt | 100 | 123 | 123 | 346 |
| Vor dem Systemtest nicht entdeckt | 60 | 77 | 157 | 294 |
| Im Systemtest entdeckt und korrigiert | 6 | 12 | 55 | 73 |

**Kosten und Fehlerentdeckung im Systemtest.** Im Beispiel werden Kosten und Nutzen für einen (typischen) Systemtest der Funktionen und Äquivalenzklassen berechnet. Sonderfälle werden nicht gezielt geprüft, es findet kein Glass-Box-Test und keine frühe Testvorbereitung statt. Der Tester hat eine mittlere Kompetenz. Die gewählten Methoden bestimmen die Testfallzahl, aus der Testfallzahl wird die Fehlerentdeckung und der Testaufwand berechnet: Jones [12] gibt maximal 0,6 Testfälle pro Function Point an. Im Beispiel werden Sonderfälle aber nicht gezielt geprüft, darum wird mit 80 Testfällen gerechnet. Aus 80 Testfällen resultieren die typischen Fehleranteile [11]: 10 % der Spezifikationsfehler, 15 % der Entwurfsfehler und 35 % der Codefehler. Es werden im Systemtest 73 Fehler entdeckt (Tabelle 2).

Die Vorbereitung und Durchführung kostet für jeden Testfall 1 Eh (Entwicklerstunde) Aufwand [11]. Insgesamt fallen also 80 Eh an. Einer der entdeckten Fehler blockiert den Test. Erst nach der Korrektur dieses Fehlers kann der Test fortgesetzt werden, dazu muss aber ein Teil der Testfälle wiederholt werden. Diese Wiederholung kostet weitere 10 Eh (25 % des Testaufwands für die Wiederholung [17] von 50 % der Testfälle). Nach COCOMO-II wird im Systemtest ein Tester eingesetzt. Damit dauert der Systemtest mit 90 Eh rund 12 Tage.

Im Systemtest kostet die Korrektur eines Fehlers 10 Eh [11], frühe Fehler sind teurer [14]. Im Modell werden 15 Eh pro Spezifikationsfehler, 12 Eh pro Entwurfsfehler und 10 Eh pro Codefehler aufgewendet. Insgesamt ergeben sich 784 Eh Korrekturaufwand. Nach COCOMO-II werden im Projekt drei Programmierer beschäftigt. Werden alle Programmierer eingesetzt, dauert die Korrektur 33 Tage.

**Nutzen des Systemtests.** Weil im Systemtest Fehler entdeckt und korrigiert werden, entfallen für diese Fehler Kosten in den folgenden Prüfungen und in der Wartung: Während nach frühen Prüfungen der Nutzen in mehreren Folgeprüfungen erreicht wird, kommt nach dem Systemtest nur noch ein Feldtest in Frage.

**Tabelle 3.** Entdeckte und korrigierte Fehler, die später entfallen

| | Spezifikationsfehler | Entwurfsfehler | Codefehler | Summe Fehler |
|---|---|---|---|---|
| Im Systemtest entdeckt und korrigiert | 6 | 12 | 55 | 73 |
| • davon würden im Feldtest entdeckt | 1 | 2 | 14 | 17 |
| • davon würden nach Auslieferung entdeckt | 5 | 10 | 41 | 56 |

Im Beispiel wird der Feldtest durchgeführt. Durch den Feldtest werden 20 % der Spezifikations- und Entwurfsfehler und 25 % der Codefehler entdeckt [11].

Daraus folgt, dass von den 73 Fehlern, die im Systemtest entdeckt werden, 17 Fehler im Feldtest entdeckt werden würden (Tabelle 3). Die Korrektur dieser Fehler wird durch den Systemtest eingespart (179 Eh, 8 Tage, 3 Programmierer). Blockierende Fehler spielen im Feldtest keine Rolle. Bei Fehlern muss aber der Test nach der Korrektur wiederholt werden, dass Modell berechnet für den Feldtest 25 % Zuschlag auf die Korrekturkosten (45 Eh, 6 Tage, 1 Tester).

Die übrigen 56 Fehler müssten in der Wartung korrigiert werden. In der Wartung kostet jeder Spezifikationsfehler 25 Eh, jeder Entwurfsfehler 20 Eh und jeder Codefehler 15 Eh, so dass 940 Eh vermieden werden.

Zur Berechnung der Fehlerfolgekosten wird ein Beispielszenario angenommen: Fehler in der Software führen ausschließlich zu Komfortverlusten. Darum verursacht ein auftretender Fehler 10 € Schaden. Die Software wird während der Korrektur eines Fehlers 10mal verwendet. Vereinfachend für das Beispiel wird angenommen, dass jeder Fehler mit einer Wahrscheinlichkeit von 0,5 bei einer Verwendung zu einem Fehlverhalten führt. Durch den Systemtest wird vermieden, dass 56 Fehler ausgeliefert werden. Als statistische Folgekosten der 56 Fehler ergeben sich 2800 €.

## 4 Modellerprobung

Das Modell ist quantitativ und deterministisch. Sein Aufbau orientiert sich an den Projektaktivitäten. Die Modellparameter wurden möglichst so gewählt, dass sie in realen Projekten einfach erhoben werden können. Die Zusammenhänge sind durch Gleichungen beschrieben, quantifiziert für Auftragsprojekte [1–4, 10–14], und bislang als Spreadsheet realisiert.

### 4.1 Ergebnisse des Modells

Abb. 1 zeigt Ergebnisse für ein Projekt mit 200 Function Points mit gründlichen Spezifikations- und Entwurfsreviews mit drei Gutachtern, ohne Codereview, ohne Feldtest und mit einem typischen Black-Box-Test (Szenario 1), mit zusätzlichem Glass-Box-Test (Szenario 2) und mit verkürzten Reviews mit zwei hastig vorbereiteten Gutachtern und typischem Black-Box-Test (Szenario 3).

In Szenario 1 kosten die Reviews 544 Eh (Entwicklerstunden) und sparen 582 Eh im Projekt, gespart wird korrektive Wartung. Die Reviews wirken sich in gründlicheren Tests deutlicher aus (Szenario 2), damit wird bei vernachlässigten Tests der Reviewnutzen erst nach Auslieferung sichtbar. Der Systemtest spart Wartung. Vernachlässigte Reviews führen zu einem teureren Systemtest und niedrigerem Nutzen (Szenario 3).

Abb. 2 zeigt für Szenario 1 zwei Alternativen mit niedrigen Folgekosten (Szenario 1a) und mit hohen Folgekosten (Szenario 1b). Der Aufwand wird mit 100 € pro Eh bewertet, Dauer und Personal werden nicht bewertet. Niedrige Folgekosten werden durch Komfortverluste mit jeweils 10 € Schaden und 10 Verwendungen geschätzt. Die durch Prüfungen und Korrektur vermiedenen niedrigen Folgekosten tragen wenig zum Nutzen bei, es werden rund 10.000 € vermieden.

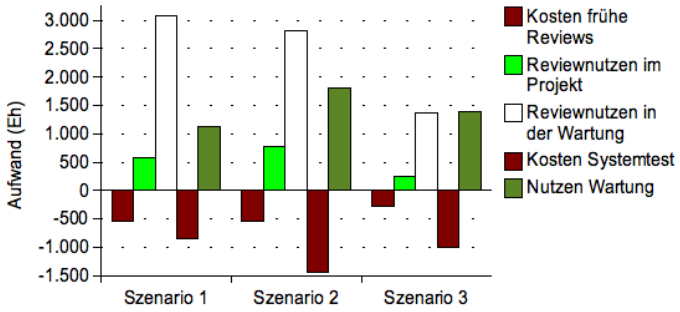


**Abbildung 1.** Ergebnisse für drei Szenarien

Kosten aber kritische Fehler einen Arbeitstag (1000 €) und wird das Produkt bis zu 100 mal verwendet, überwiegen die vermiedenen Folgekosten (1,9 Mio. €). Noch deutlicher wird dies bei sicherheitskritischen Anwendungen mit sehr hohen Folgekosten eines Fehlers.

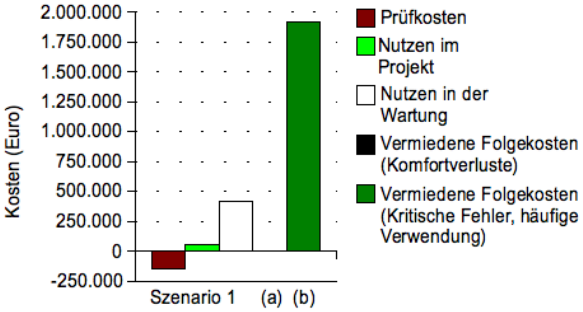


**Abbildung 2.** Ergebnisse für Szenario 1 mit unterschiedlichen Folgekosten

### 4.2 Empirische Untersuchung

Zur Erprobung des Modells haben wir es auf die gemittelten Daten aus 22 studentischen Projekten angewendet. Alle Projekte haben gleiche Anforderungen und wurden mit den gleichen Vorgaben für Prüfungen und Meilensteine durchgeführt. Die Modellparameter wurden an das Vorgehen und die Anforderungen angepasst. Das Modell wurde mit der gemittelten Produktivität kalibriert. Als Schätzwert für die Zahl der im System enthaltenen Fehler wurde die mittlere Zahl der bis zum Abnahmetest gefundenen Fehler verwendet. Die ersten Ergebnisse zeigen, dass das Modell die Mittelwerte der Projekte für Fehlerentdeckung, Aufwand und Dauer plausibel schätzt. Einzelne Projekte zeigen aber deutliche Abweichungen. Eine Untersuchung der einzelnen Arbeiten soll zeigen, ob

sich individuelle Unterschiede der Projekte in den Modellergebnissen zeigen. Die Prognose der Folgekosten kann in dieser Untersuchung nicht validiert werden; mit einem passend gewählten Szenario erscheinen die Ergebnisse aber plausibel.

## 5 Diskussion

Das Modell ist grundsätzlich geeignet, die Qualitätsplanung zu unterstützen. Es zeigt plausible Ergebnisse. Es bestehen aber Einschänkungen: Das Modell ist auf Auftragsprojekte mit stabilen Anforderungen und ohne Iterationen zugeschnitten [11]. Damit diese Aspekte in das Modell übernommen werden können, muss der Einfluss sich ändernder Anforderungen geklärt und modelliert werden. Objektorientierte Methoden spielen in den verwendeten Daten eine geringe Rolle. Die erste Erprobung zeigt geringe Unterschiede im Unittest und durch die kontinuierliche Integration. Die Prognose der Folgekosten ist kaum geeignet, um katastrophale Fehler, die mit extrem geringer Wahrscheinlichkeit auftreten, abzuschätzen.

Das Modell enthält kein differenziertes Qualitätsmodell, alle Mängel sind als Fehler definiert. Es gibt beispielsweise keine Unterscheidung zwischen Problemen mit der Bedienung oder mit der Performanz. Die Wartbarkeit wird nicht betrachtet. Ein differenziertes Qualitätsmodell würde erlauben, gezielte Verbesserungen einzelner Qualitätsattribute zu bewerten, etwa Usability Tests, Stress- und Lasttests oder Verbesserungen der Wartbarkeit. Dafür fehlen aber empirisch belegte Zusammenhänge.

Wir halten die bisherige Validierung mit studentischen Projekten für unzureichend. Die Validierung kann grundsätzlich global oder auf Detailebene erfolgen. Eine globale Validierung ist kaum möglich: Das Modell könnte direkt nur durch ein (unbezahlbares) Experiment validiert werden. Feldversuche erlauben nicht, Auswirkungen unterschiedlicher Entscheidungen direkt zu vergleichen. Eine Plausibilitätsprüfung ist aber in jedem Fall möglich. Auf Detailebene sind einige Beziehungen empirisch belegt, vor allem die Zusammenhänge der Reviews und die grundlegenden Zusammenhänge für Aufwand und Dauer. Andere Beziehungen, insbesondere für den Systemtest, sind noch durch empirische Untersuchungen zu zeigen. Die Prognose der Fehlerfolgekosten lässt sich kaum quantitativ validieren, so dass wir für diesen Bereich auf Hypothesen und qualitative Bewertungen angewiesen sind. Zusätzlich wird das Modellverhalten durch eine Sensitivitätsanalyse geprüft.

## Literatur

1. Basili, V. R.; Selby, R. W.: Comparing the Effectiveness of Software Testing Strategies. IEEE TSE, 13, 12 (1987) 1278–1296
2. Bassin, K.; Biyani, S.; Santhanam, P.: Metrics to Evaluate Vendor Developed Software Based on Test Case Execution Results. IBM Systems Journal, 41, 1 (2002)
3. Biffl, S., Halling, M.: Investigating the Influence of Inspector Capability Factors with Four Inspection Techniques on Inspection Performance. Proc. of the 8th IEEE Symposium on Software Metrics (2002) 107–117

4. Boehm, B.W.: Software Cost Estimation with COCOMO II. Prentice-Hall (2000)
5. Boehm, B.; Huang, L.; Jain, A.; Madachy, R.: The ROI of Software Dependability: The iDave Model. IEEE Software, 21, 3 (2004) 54–61
6. Drappa, A.; Ludewig, J.: Simulation in Software Engineering Training. Proc. of the 22nd International Conference on Software Engineering (2000) 199–208
7. Freedman, D. P., Weinberg, G. M.: Handbook of Walkthroughs, Inspections, and Technical Reviews. Little, Brown and Company, 3rd Ed. (1982)
8. Freimut, B.; Briand, L. C.; Vollei, F.: Determining Inspection Cost-Effectiveness by Combining Project Data and Expert Opinion. IEEE TSE, 31, 12 (2005) 1074–1092
9. IEEE Guide. Adoption of PMI Standard. IEEE Std. 1490-2003 (2003)
10. Jalote, P.: CMM in Practice. at Infosys. Addison–Wesley (2000)
11. Jones, C.: Applied Software Measurement. McGraw–Hill, 2nd Ed. (1997)
12. Jones, C.: Estimating Software Costs. McGraw–Hill, 2nd Ed. (2007)
13. Juristo, N.; Moreno, A.; Vegas, S.: Reviewing of 25 Years of Testing Technique Experiments. Empirical Software Engineering, 9, 1-2 (2004) 7–44
14. Kan, S. H.: Metrics and Models in Software Quality Engineering. Addison Wesley, 2nd Ed. (2003)
15. Rico, D. F.: ROI of Software Process Improvement. J. Ross Publishing (2004)
16. Rubey, R. J., Browning, L. A., Roberts, A. R.: Cost Effectiveness of Software Quality Assurance. Proc. of the IEEE 1989 NAECON , 4 (1989) 1614–1620
17. Van Megen, R., Meyerhoff, D. B.: Costs and Benefits of Early Defect Detection: Experiences from Developing Client Server and Host Applications. Software Quality Journal, 4, 4 (1995) 247–256
18. Schwinn, T.: Effiziente Software-Inspektion durch ein Rahmenwerk zur antizipativen Berücksichtigung des Return-On-Investment. Dissertation. Fakultät für Informatik, Universität Ulm (2003)
19. Wagner, S.: Cost-Optimisation of Analytical Software Quality Assurance. Dissertation. Institut für Informatik der Technischen Universität München (2007)

# Balancing Upfront Definition and Customization of Quality Models

Michael Kläs, Jürgen Münch

Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
{michael.klaes, juergen.muench}@iese.fraunhofer.de

**Zusammenfassung** The selection and customization of quality models for the development of software systems or software-intensive systems and services is a challenging task. Due to the nature of software development, quality models such as reliability models or defect models need to be adapted to specific project goals and environment characteristics. Currently, there is a tremendous deficit in understanding the selection and customization of appropriate quality models. Quality models that balance upfront definition and customization are widely missing. This article describes two types of quality models, i.e., fixed models and define-your-own models, gives an initial overview of possible variabilities in quality models, and finally sketches elements for the definition of balanced quality models.

## 1 Introduction

In contrast to production engineering, software development usually produces individual results. In addition, the software development process is, to a large extent, a creative, human-based activity that varies from project to project. However, quality assurance in software development is often performed in a way similar to that in production engineering: Usually, it is assumed that each project is the result of the same process. In consequence, many software quality models are fixed quality models without systematic customization support. Quality models that can be tailored to the individual goals and environment characteristics of software development projects are widely missing.

Software quality can be seen from a multitude of perspectives and is relevant in nearly all application domains. In most cases, software quality is attached to products, i.e., artifacts such as design documents, code modules, or the resulting system. Definitions of quality provided in standards like ISO 8402 [1] usually focus on a product-based or user-based view: "the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs". ISO 14598 [2] provides a very general definition of a quality model in the context of information systems: "the set of characteristics and relationships between them which provides the basis for specifying quality requirements and evaluating quality". Following this definition, a quality model can be considered as an operationalization of the term quality.

Although most quality models presented in the literature or defined in standards are models for product quality (e.g., [6], [3]), they can also be attached to processes (e.g., maturity models, process adherence models), resources (e.g., server availability model, qualification model), or projects (e.g., milestone slippage model).

## 2 Fixed versus Define-Your-Own Models

In general, there are two main directions in software quality modeling [4]: On the one hand, there are *fixed-model approaches*, which usually prescribe important quality characteristics as a subset of the relevant quality focus in a published model. Sometimes, predefined measures for the quality characteristics are also given. On the other hand, *define-your-own-model approaches* support the derivation of relevant quality characteristics, i.e., they provide concepts and procedures on how to derive custom-tailored quality models.

Prominent examples of fixed-model approaches are the models proposed by McCall [5], Boehm [6], and ISO9126 [3]. Literature typically concentrates on them when discussing quality models. The models proposed by McCall [5] and Boehm [6] both present key characteristics of quality identified from a user perspective. ISO9126 [3] was developed in the early 1990s as an attempt to consolidate the many views of software quality. Further fixed model approaches are, e.g., the FURPS/FURPS+ model developed by Hewlett-Packard [7], [8], the DGQ model [9], and the SATC Software Quality Model [10] developed at NASA.

According to [11], it is not realistic to assume that one can provide a prescriptive set of necessary and sufficient quality characteristics to describe the quality requirements of any project; therefore, they propose specifying how quality characteristics should be specified rather than specifying a fixed set.

A prominent example of such a define-your-own-model approach is the SQUID approach [11]. It provides guidance in the customization of a quality model and aims at clarifying the link between process and product quality, but provides no detailed guidance regarding the measurement of quality characteristics.

To some extent, the GQM approach [15] can also be seen as an approach that supports the custom-tailored definition of quality models. The primary objective of the GQM approach is to provide a framework for defining measurement goals and deriving measures that help to answer these goals. Further define-your-own model approaches are presented, e.g., by Dromey [12] and Gilb [14].

## 3 Balanced Quality Models

Support of an appropriate level of upfront definition and opportunities for customization is an important criterion for the applicability of a quality model. On the one hand, it is in general not possible to define quality characteristics and measures that are valid for any purpose in any context. On the other hand, very generic define-your-own approaches require very high skills and significant

efforts for creating quality models. We state the following hypothesis: For specific domains and specific purposes, custom-tailored, so-called *balanced quality models* can be constructed by adapting a core model that captures the underlying quality concept in the specific domain. The adaptation needs to be guided by a detailed process so that it is reproducible. A prerequisite for defining such a balanced quality model is, besides other issues, a deeper understanding of the variable elements of quality models and their degree of variability. Therefore, below we sketch variabilities in quality models and potential impact factors on these variabilities.

Popular quality models such as [3], [5], [6], [7], and [11] have in common that they decompose quality into characteristics and sub-characteristics in a tree-like-fashion to make them ultimately measurable. Usually, this kind of approach is called Factor Criteria Metric approach [13]. Therefore, the (1) *decomposition of quality into characteristics* and the (2) *definition of metrics for characteristics* can be considered as the two basic elements of any quality model. They are sufficient if the model is used for characterizing quality.

If the model is to be used to derive development guidelines or to identify areas of possible improvements, factors influencing the quality (characteristics) have to be known at least on a qualitative level (e.g., avoiding redundancy in code increases final product maintainability, performing a fault tree analysis increases product robustness, developers experience reduces product cost). Therefore, (3) *influencing factors* are required in quality models used for improvement.

If the model is to support the estimation of quality characteristics for planning a software project (e.g., cost estimation) or for controlling a project (e.g., prediction of product reliability), the relation between the influencing factors and the influenced quality (characteristic) have to be quantitatively modeled. Therefore, a (4) *quantitative model of influencing factors* is required in models applied for prediction.

Finally, if the model is to be used to plan overall quality and specify quality requirements (i.e., setting target values for the quality characteristics), the dependencies between different quality characteristics have to be known. Since quality characteristics can influence each other (e.g., an increased functionality can reduce the performance or usability of a product), tradeoff decisions must usually to be performed. Therefore, (5) *dependencies between quality characteristics* are an element in models applied in overall quality planning.

Fig. 1 gives an overview of the described concepts of a quality model. The mentioned variation points and influencing factors should not be seen as a comprehensive and complete list. There is rather an open research question of how to describe customizable quality models that can be custom-tailored efficiently by practitioners in their own environment.

## 4 Conclusion

A balanced software quality model needs to be context- and purpose-oriented and should be derived from a domain-specific core model by using a fine-grained

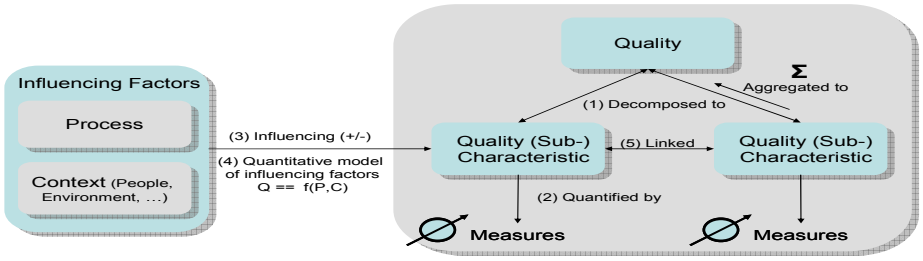


**Abbildung 1.** Elements that can be customizable in a quality model

customization process. The definition of such models requires a much deeper understanding of the variabilities of quality models. As a vision, widely accepted domain-specific software quality standards that allow for systematic customization should be created.

## Literatur

1. ISO 8402, Quality management and Quality assurance - Vocabulary
2. ISO/IEC 14598 12207 International Standard, Standard for Information technology – Software product evaluation – Part 1: General overview
3. ISO/IEC 9126 International Standard, Software engineering Product quality, Part 1: Quality model, 2001.
4. Norman E. Fenton and Shari Lawrence Pfleeger, Software Metrics - A Practical and Rigorous Approach. International Thomson Computer Press, 2nd edition ed., 1996.
5. James A. McCall, Encyclopedia of Software Engineering. Volume 2., ch. Quality Factors, pp. 1083–1093. John Wiley Sons, 2002.
6. Barry W. Boehm, John R. Brown, Hans Kaspar, Myron Lipow, Gordon J. MacLeod, and Michhhael J. Merritt, Characteristics of Software Quality. North Holland Publishing Company, 1978.
7. Robert B. Grady and Deborah L. Caswell, Software Metrics: Establishing a Company- Wide Program, Prentice-Hall, 1987.
8. Robert B. Grady, Practical Software Metrics for Project Management and Process Improvement. Prentice Hall, 1992.
9. Deutsche Gesellschaft fr Qualitt, Software-Qualittssicherung. No. DGQ-NTG-Schrift 12-51, VDE-Verlag Berlin, 1986.
10. Larry Hyatt and Linda Rosenberg, A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality, in Proceddings of the 8th Annual Software Technology Conference, 1996.
11. Barbara Kitchenham, Steve Linkman, Alberto Pasquini, and Vincento Nanni, The SQUID-Approach to Defining a Quality Model, Software Quality Journal, vol. 6, no. 3, pp. 211-233, 1997.
12. Geoff Dromey, A Model for Software Product Quality, IEEE Transactions on Software Engineering, vol. 21, pp. 146–162, Feb. 1995.

13. J.A.McCall, P.K.Richards, G.F.Walters, Factors in Software Quality, RADC TR-77- 369, 1977. Vols I,II,III', US Rome Air Development Center Reports NTIS AD/A-049 014, 015, 055, 1977.
14. Tom Gilb, Principles of Software Engineering Management, Addison-Wesley, 1988.
15. Victor R. Basili: Software Modeling and Measurement: The Goal/Question/Metric paradigm, Technical Report CS-TR-2956, Department of Computer Science, University of Maryland, College Park, MD 20742, 1992.

# Security Requirements Addressing Security Risks for Improving Software Quality*

Shareeful Islam[1] and Wei Dong[1,2]

[1] Institut für Informatik, Technische Universität München, Germany
[2] School of Computer, National University of Defense Technology, P.R. China
{islam,dongw}@in.tum.de

**Zusammenfassung** Software systems that serve the modern society have become more complex, extensible, and are mostly distributed through the high speed untrustworthy networks. Any security vulnerability of the software can make a devastating impact on organizations data, financial cost and recovery time. Software security is a challenging issue tightly related with the software quality. High quality software means the secure software. This paper attempts to extract the software security risks impact on software quality, and identifies the security requirements needed to be considered in the early stage of development. Some corresponding technologies which can mitigate the security risks for high quality software are also presented.

## 1 Introduction

Software systems integrate peoples mind, information and process to conduct all manners of real time functions, such as completing financial transactions, maintaining confidential information etc. These complex functionalities are implemented and interrelated in heterogeneous distributed environments. Interested parties access the software from different geographical locations through network and web enable applications. Any security vulnerability hides in these untrustworthy platforms can cause direct risks for the business, brand value and customer confidence. High quality software should obviously be highly secure software which is hard to be achieved. This may because of the constant changes of software and hardware environments, the pressure of delivering products to the market in a highly competitive environment, and the security breaches that still faced despite of using the best secure algorithms, protocols and the most advanced security devices. Software security considers mitigation of the risks that may impact business goal, asset and product quality. Security is part of the software quality needs to be achieved by ensuring confidentiality, integrity, availability, authenticity, non repudiation, etc., of the data and services offered by the software [3]. Obtaining security should think of the security requirements

* The work is partly supported by the German Academic Exchange Service (DAAD) and the National Natural Science Foundation of China (No.60673118), National 863 Hi-Tech Research and Development Program of China (No.2006AA01Z429).

in the early stage of software development, and consider what should be protected, for whom it needs to be protected, and how long with how much cost. This paper shows how security risks impact on the criteria of McCalls software quality model, and what are the security requirements, analysis process and corresponding technologies to address these risks.

The rest of the paper is organized as follows. In section 2, relations between software quality and software security with their basic definitions are presented. Impacts of security risks on software quality are elaborated in section 3. Critical software security risks and their impacts on the software operation metrics of McCalls model are also considered in this section. Section 4 identifies the security requirements to mitigate the security risks into an acceptable level, and the technologies involved for ensuring the security requirements are presented. Section 5 gives a generic process of how to establish security requirements from the identified security risks. Finally, conclusion and future work are mentioned in section 6.

## 2 Software Quality vs. Software Security

Software quality conforms to the system specifications and ensures fitness for the intended purpose for which customers are willing to pay. It can be defined through the hierarchical model. McCalls software quality model includes three major areas, i.e. software operations, revision and transition, as shown in figure 1 [5]. Each area has its own criteria which are evaluated by a set of metrics. Depending on the nature of the application to be developed, some of the metrics need to be ensured. But most software applications deployed today contain vulnerabilities that can be exploited to cause considerable loss or business interruption. So security risks can directly impact on the software quality. Software security ensures that only authorized parties can access to the system and modify the data in authorized ways. The impacts of security are highly contextual depending on the context where the software is deployed. Most software runs in the open environments and interacts with other software (e.g. operating systems) and hardware. Thus, the security should be concerned for the whole environment. Software quality and security assurance will require fundamentally different approaches. Quality problem can be defined as failure of execution according to the specifications. Security problem can be defined as software vulnerability which allows specified functions to be compromised by the attackers. Quality is when the software does not fault and security is when the software does not compromise.

Security vulnerabilities do not commonly be visible to everyone except the hackers or may be latent for a long time, but symptoms of the poor quality software is visible to everyone. Security is one of the software quality attributes as considering the product being secured from all possible vulnerabilities. The properties for a secure software quality can be considered by a set of security requirements. Security requirements, a subgroup of software requirements, are the protection profile identifies desired security properties of a product [2]. The

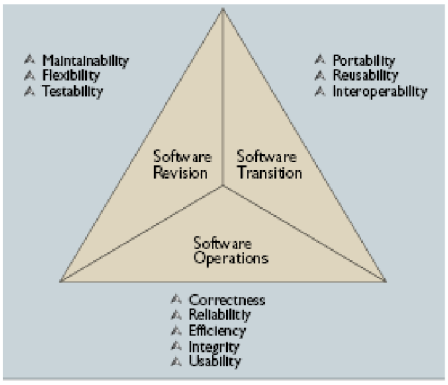


**Abbildung 1.** McCall's model of software quality

identified security requirements should merge into the traditional requirement engineering at the requirement engineering phase of software development.

To identify the requirements and corresponding techniques addressing software security, the relations or mapping between security risks and software quality metrics should be deeply studied.

## 3 Security Risks and Quality Factors

Software operation criteria of McCalls quality model are directly affected by the security risks. Other criteria of the model, such as portability of software transition and maintainability of software revision, are also crucial for the overall software product quality. But for software security risks, we targeted only to all criteria of software operational capabilities. A complete critical asset based software security risk management need to be considered at the early stage, so that possible risks can be identified, analyzed and mitigated. Risks on confidentiality, integrity and availability of software functionalities with their direct impacts on the quality criteria including correctness, reliability, efficiency, integrity and usability are presented in table 1. A single tick ($\surd$) indicates negative impact with less damage, and double tick ($\surd\surd$) indicates strong negative impact with severe damage.

**Correctness**: The ability to satisfies the specifications, so that software can behave correctly and meet the customers requirements in the prescribe environment. Security vulnerability can arise from incomplete, vague, missing specification or if the specification is not properly implemented or traced for continuous evolution [3]. As a consequence of this vulnerability, intruders can access and modify data by many ways such as buffer overflow, broken access control, session hijacking etc. Major risk consideration due to failure meet of correctness is the loss of authenticity which impact on integrity, privacy and customers satisfaction.

**Reliability**: The probability of failure-free software operation for a specified period of time in a specified environment. Vulnerability on reliability can arise when functionalities of software are not working properly due to the errors which frequently deviate from correct state [7]. Root causes of these errors may be system failures, improper configurations or man-made intentional and unintentional faults. Failures of reliability make software more vulnerable for security breaches which can impact on integrity, service availability and customers confidence.

**Efficiency**: It concerns with measuring the deterministic values such as computation time, hardware, accuracy, response time etc. A security breach can affect all these deterministic values and degrade the efficiency and system performance. Service availability and customers satisfaction may meet the risk based on efficiency.

**Integrity**: Protection of the program, data or service from unauthorized accesses, improper alternations and corruption. Ensuring integrity requires built-in security functions to cope with the threats. There are lots of causes, such as broken access control, IP spoofing or even by social engineering etc., based on which users can access the system illegally, and as a result integrity can be lost. Loss of integrity might have strong negative impact and severe damage on unauthorized modification, corruption of confidential data and breach of privacy.

**Usability**: Evaluating the efficient and correct usage of the system. It is the user experience with the software, which may be determined by the time and resources required to learn and operate the software, or if it is easy to remember how to use and recover software from errors. Security breaches provide poor usability. On the contrary, sometimes more security adds to a system, makes it less usable due to extra complexity, confusion and pain to the user [9]. Main risk from poor usability is customers dissatisfaction and denial of service.

Human factors are one of the root causes for theses security risks. User inside the organization and external hackers may have intentional or unintentional motivations to exploit security breaches. Causes also arise from the system due to its improper configuration, or failures of legal and regulator requirements. Some causes are very common for all types of security risks. These identified security risks need to be controlled at an acceptable level for high quality software. A set of security requirements and corresponding technologies should be identified to accomplish the tasks, which are also shown in table 1.

## 4 Security Requirements and Security Technologies

Complete and rigorous software security risk management needs to be conducted at the early stage so that security requirements are identified to address security risks. Security requirements are quality requirements considering quality factors such as access control, integrity, non reputation, privacy, malicious activities detection, physical security, etc., to ensure secure quality of software [1]. They are elaborate requirements which enforce the implementation of security policy for an organization. The software security risk assessment identifies threats, risks

**Tabelle 1.** Security requirements addressing security risks and quality criteria

| Security Risks | Root Causes | Correctness | Reliability | Efficiency | Integrity | Usability | Security Requirements (abbr. req) | Security Standards/ Technologies/ Modeling |
|---|---|---|---|---|---|---|---|---|
| Loss of authenticity & integrity | Broken access control; unauthorized modification; buffer overflow; SQL injection; cross-site scripting; IP spoofing; weak password based system; invalidated input; improper Error handling; session hijacking; improper use of SSL; improper file access; social engineering; misconfiguration | √√ | √ | | √√ | | User authentication & authorization req; integrity req; non reputation req; malicious activities detection req; security auditing req; system maintenance & upgrade req; physical security req | **Authenticity:** Mandatory access control (file & application, network), role-based security; kerberos; PKI, SSL, SSH; smart card; secure tokens; password management; biometric auth. **Integrity:** Cryptographic; digital signature; IPSec, SET, integrity audit |
| Breach of privacy | Broken access control; disclose of information; cross-site scripting; IP spoofing; session hijacking; weak password based system; social engineering; physical faults; failure to protect network traffic, improper file access | √ | | | √ | | Privacy req; user authentication and authorization req; integrity req; malicious activities detection req; security auditing req; physical security req; Legal and regulator req. | Mandatory access control; password management; user lock limit; network auditing (port, IDS, log); Bodyguard; alarm; firewall; cryptographic; SSL, ACL,PKI, PGP |
| Denial of service | Service failure; SYN flooding; race conditions; buffer overflow; lack of personnel skill; system & network problem; hardware fault; unavailability of support personnel; misconfiguration; lack of awareness | | √√ | √√ | √ | √ | Service availability req; backup & recovery req; security auditing req; malicious activities detection req; system maintenance & upgrade req | VPN; packet filter; firewall; network auditing (port, IDS, log); source code audit; devices performance monitor; secure backup; adequate training |
| Customer's dissatisfaction | Service failure; lack of personnel skill, lack of awareness; poor usability; QoS degradation; system & network problem; loss of privacy ; breach of legal, regulatory or contractual obligation | √ | √ | √ | √ | √ | Service availability req; backup & recovery req; system maintenance & upgrade req; Legal and regulator req. | Requirements modeling; verification and validation; simulation; model based testing; security awareness program; educating users |

and their impacts on the product that need the effective cost tradeoff by choosing a set of security requirements.

Security requirements considered in table 1 specify that software security risks must not be allowed to happen. User authentication, authorization and integrity requirements ensure the availability and integrity of the software when someone performing an operation in an authorized way [8]. These combined security requirements also verify the user identity, its access and privilege level to use the system and ensure that data and services are not intentionally corrupted through any malicious activities. Non-repudiation requirements prevent users from denying after participation in any interaction. Privacy and physical security requirements ensure that both internal and external unauthorized individuals cannot gain access to confidential information. System maintenance and upgrade requirements prevent from accidentally defeating of security mechanisms while executing system maintenance (updating patch, fixing bugs, etc.) [8]. Backup and recovery requirements specify the extent to which an application shall survive, protect the system from loss or destruction, and ensure proper back up. Legal and regulator requirements consider the legal issues that product need to attain. Identified security requirements validate whether security mechanisms (e.g. password based access control), security devices (e.g. firewall, IDS) and security protocols (e.g. secure socket layer, cryptography) are integrated into the software architecture. Improved security specifies the requirements of user interfaces, secure coding and security testing that need to be done for secure software quality.

The last column in table 1 presents some corresponding solutions and methods to overcome the risks and improve the secure software quality. These security technologies are based on the identified security requirements. Security devices such as firewall and IDS can filter unauthorized access, detect malicious activities and log user activities. Digital signature and IPSec can be used for integrity and authenticity assurance. Physical authentication tools such as smart card, authentication protocols such as Kerberos, secure token and PKI, can be used for user authentication and authorization requirements. Mandatory access control, password management etc. can improve the privacy. Device performance monitor, secure backup, network monitor etc. can help for avoiding denial of service. Due to the space limitation, not all existing security solutions are specified in table 1.

Customer dissatisfaction can directly affect all quality criteria. To reduce the customer dissatisfaction, the exact customer requirements should be achieved and described by requirement modeling techniques. Then the methods such as formal verification, simulation, model-based testing, can be carried out to determine whether the system meet these requirements:

- Formal verification. There have been many researches about the verification of the security properties, such as security protocol verification, model checking of security specifications, etc. These techniques can be used to ensure the security, in which abstraction may be needed.

- Simulation. The executable prototype, model, or preliminary system can run in the simulated environment, and all kinds of attacks can be produced to find the vulnerability of the system.
- Model-based testing. Testing is still the main methods to ensure software quality. Because the security requirements are specified in the early stage, and the requirement models can be constructed. Then the security test cases can be generated based on the model, and will be applied in the testing activities.

The results of these methods can be used to improve the secure quality of the software, e.g. the counter example of model checking can be analyzed to determine the traces of the successful attack, and finally the countermeasures that satisfy the user specifications can be found. Usability needs to give more attention for customer satisfaction. Improve security should also improve usability of the software rather than make the whole system complex. Security awareness and adequate training are also required among users, customers and all other interested parties so that unintentional security flaws can be prevented.

## 5 Process model for security requirements identification

In this section, we will give a simplified process of establishing security requirements from the identified risks to attain secure software. The procedure is presented in figure 2.

The whole process initiates through a complete asset based risk management during the requirement engineering of the software development. All possible risks relating to the specified project are identified and analyzed by quantifying their impacts. These risks are also mapped to the corresponding quality attributes according to table 1. These identified risks need to be controlled at an acceptable level by risk management strategies based on the resulting security requirements. Finally security technologies, protocols, mechanisms will be adopted, and the security design should be obtained to attain secure quality of the software.

## 6 Conclusion

No system is entirely secure due to the high complexity, cost and usability of the developed software, and the improving technologies used by the cyber criminal community. It is also hard to prove that the software is defect free. Considering the quality and security at the early stage, and identifying security requirements through software risk management to mitigate the security risks into an acceptable level are essential for developing high quality software. This paper gives our preliminary works of how to establish security requirements from the security risks, and presents the relations among software security risks, quality, requirements and corresponding technologies. In our future work, these relations and detailed, practical steps will be further studied.

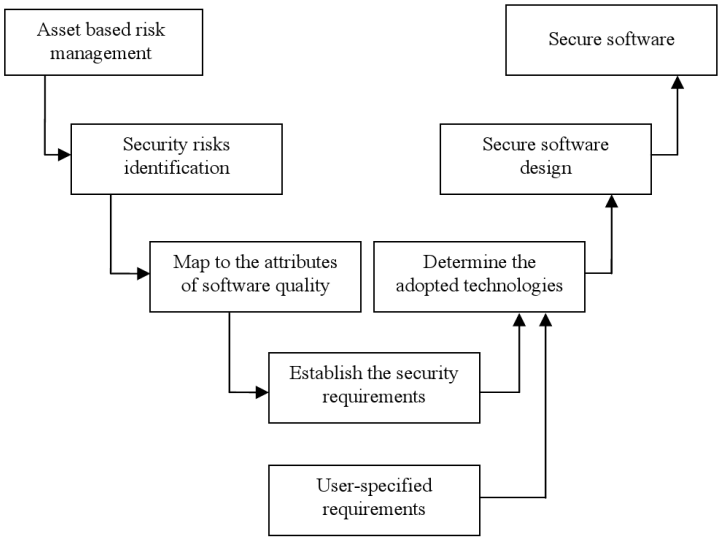


**Abbildung 2.** Security requirements analysis process

## Literatur

1. Firesmith, D.G. Specifying Reusable Security Requirements. Journal of Object Technology. Vol.3, No.1, pp.61-75, January-February 2004.
2. R. Anderson. Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley, 2001.
3. McGregor J.D. Secure Software. Journal of Object Technology, Vol.4, No.4, 33-42, 2005.
4. H. Wang, C. Wang. Taxonomy of security considerations and software quality. comm-unications of the ACM, Vol. 46, No. 6, 2003.
5. Gillies C. A. Software Quality: Theory and Management. Chapman & Hall Computing, 1992.
6. Mead R. N. How To Compare the Security Quality Requirements Engineering (SQUARE) Method with Other Methods. Technical Note, CMU/SEI, 2007.
7. Avizienis A., Laprie C.J., Randell B., Landwehr C. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, pp11-33, January 2004.
8. Firesmith, D.G. Engineering Security Requirements. Journal of Object Technology. Vol.2, No.1, 53-68, January-February 2003.
9. K. Rozinov. Are Usability and Security Two Opposite Directions in Computer Systems?. Whitepaper in Department of Computer and Information Science, Polytechnic University, 2004. `http://rozinov.sfs.poly.edu/papers/security_vs_usability.pdf`.

# Ein verfeinerter GQM-Ansatz zur Qualitätsbewertung von Software-Modellen

Hendrik Voigt und Gregor Engels

Universität Paderborn, Deutschland
hvoigt,engels@upb.de

**Zusammenfassung** Wir stellen einen Qualitätsmanagementansatz zur Bewertung von Software-Modellen vor. Unser Ansatz basiert auf der Goal Question Metric (GQM). Wir verfeinern GQM und fügen wichtige Konzepte und Aktivitäten hinzu, um auf diese Weise die Besonderheiten bei der Qualitätsbewertung von Software-Modellen berücksichtigen zu können. Dabei konzentrieren wir uns insbesondere auf den Kontext eines Software-Modells als entscheidenden Einflussfaktor für die Dokumentation von Informationsbedürfnissen, Qualitätsverständnis, Messung und Analyse. Aktuell wird eine Werkzeugunterstützung für die Eclipse Plattform entwickelt, damit unser Qualitätsmanagementansatz evaluiert und wirtschaftlich eingesetzt werden kann.

## 1 Motivation

Im Rahmen der modellbasierten Softwareentwicklung bilden Software-Modelle ein Bindeglied zwischen der Problemdomne und dem zu implementierenden Softwaresystem. Software-Modelle werden aus mehreren Grnden eingesetzt. Sie frdern die Kommunikation ber Probleme und Lsungen, die Koordination von Aufgaben und somit die Zusammenarbeit innerhalb eines Projektteams. Zudem lsst sich mit Software-Modellen der Entwicklungsfortschritt whrend der Analyse der Problemdomne und beim Entwurf der Systemarchitektur dokumentieren. Dadurch knnen Software-Modelle helfen, die Komplexitt des zu implementierenden Softwaresystems bei gleichzeitigem Kosten- und Termindruck zu beherrschen. Infolgedessen stellen Software-Modelle zentrale Entwicklungsartefakte dar.

Qualittsmngel in Software-Modellen sollten mglichst frh erkannt werden, damit sich notwendige Korrekturen nur auf wenige Entwicklungsphasen und die entsprechenden Entwicklungsartefakte beziehen. Um die Qualitt von Software-Modellen kontrollieren zu knnen, bentigen wir einen Qualittsmanagementansatz. Dabei stellen sich eine Reihe wichtiger Herausforderungen. Diese werden in Kapitel 2 thematisiert. Kapitel 3 stellt Arbeiten zur Goal Question Metric (GQM) vor. In Kapitel 4 wird unser Lsungsansatz in den Grundzgen vorgestellt und die laufenden Arbeiten beschrieben.

## 2 Herausforderungen

Qualität ist durch die Wahrnehmung von Menschen bestimmt und ist somit inhärent subjektiv. Jedoch werfen individuelle Unterschiede im Qualitätsverständnis innerhalb eines Projektteams Probleme auf. Deshalb halten wir es für sinnvoll, dass sich Beteiligte eines Projektteams, eines Unternehmens oder sogar einer Anwendungsdomäne auf ein *gemeinsames Qualitätsverständnis* einigen. Dies ist allerdings schwierig.

Software-Modelle sind Zwischenprodukte und Produktqualität ist definiert als die Gesamtheit der Charakteristika eines Produkts, die sich auf deren Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen (vgl. [1]). Dementsprechend muss die *Gesamtheit dieser Modell-Charakteristika* gefunden und dokumentiert werden.

Für die Dokumentation der Modell-Charakteristika bieten sich *Qualitäts-Modelle* an. Qualitäts-Modelle stellen einen weit verbreiteten Ansatz dar, um Qualität in mehrere Bestandteile zu gliedern. Dabei werden Qualitätscharakteristiken solange verfeinert, bis sie schließlich quantifizierbar sind. Diese quantifizierbaren Qualitätscharakteristiken bezeichnen wir als Qualitätsattribute.

Für die Quantifizierung sind *effektive Messungen* erforderlich. Sie müssen einen kausalen Zusammenhang zum betrachteten Qualitätsattribut aufweisen, der idealer Weise zusätzlich evaluiert wurde. Zudem gilt diese Forderung ausnahmslos für alle Qualitätsattribute. Das bedeutet, dass die Menge der Messungen umfassend sein muss, damit durch die Messungen kein einseitiger und verzerrter Qualitätseindruck der gegebenen Situation entsteht.

*Indikatoren* kombinieren Messungen und vergleichen diese mit Grenz- oder Zielwerten. Dadurch unterstützen Indikatoren die Interpretation der Messergebnisse und helfen, ein Qualitätsattribut zu beurteilen. Allerdings benötigt man für die Definition von Indikatoren Erfahrungswissen, das für Software-Modelle nur partiell vorliegt. Folglich stellen Indikatoren einen optionalen Bestandteil von Qualitätsplanungen dar.

Es existieren bereits diverse Arbeiten, die die *Kombination von Qualitätsattributen in Qualitäts-Modellen mit Messungen und Indikatoren* darlegen (vgl. z.B. [2]). Diese Qualitäts-Modelle bestehen aus mehreren Ebenen. Die Qualitätscharakteristiken werden in Teilcharakteristiken und schließlich in quantifizierbare Qualitätsattribute zerlegt. Attribute stellen die unterste Ebene eines Qualitäts-Modells dar und werden direkt mit Messungen sowie Indikatoren verknüpft. Diese bestehenden Arbeiten möchten wir wiederverwenden.

Erschwert wird die Bewältigung der obigen Herausforderungen durch wirtschaftliche Überlegungen. Die Etablierung, Anwendung und Pflege eines Qualitätsmanagements für Software-Modelle produziert zuerst Kosten. Diese Kosten können sich amortisieren, indem Qualitätsprobleme frühzeitig erkannt und mit geringem Aufwand behoben werden. Zudem kann ein Unternehmen durch eine höhere Endproduktqualität eine bessere Kundenbindung erreichen. Folglich muss ein *Qualitätsmanagement effizient* sein.

## 3 Verwandte Arbeiten

Wir beurteilen die Goal Question Metric (GQM) (vgl. [3]) als grundsätzlich geeignet, um bei der Qualitätsbewertung von Software-Modellen ein geeignetes *Qualitäts-Modell* auszuwählen bzw. selbst zu definieren und um *effektive Messungen* sowie *Indikatoren* zu identifizieren. GQM ist eine von Victor Basili geprägte Qualitätsmanagementstrategie für das Software Engineering. Die Grundidee von GQM basiert auf der Annahme, *that for an organization to measure in a purposeful way it must first specify the goals for itself and its projects, then it must trace those goals to the data that are intended to define those goals operationally, and finally provide a framework for interpreting the data with respect to the stated goals.*

Es existieren eine Reihe von Arbeiten, die den GQM Ansatz beschreiben, erweitern oder spezialisieren. Unsere Spezialisierung beruht auf den Ansätzen *GQM, GQ(I)M und GQM++*, die wir im Folgenden vorstellen.

Im Rahmen des TAME (Tailoring A Measurement Enviroment) Projekts wurde der ursprüngliche GQM-Ansatz durch ein Metamodell und durch einen aus sechs Schritten bestehenden Prozess zur Erstellung von Qualitätsplanungen beschrieben (vgl. [4], [5], [6], [3] [7] und [8]).

Der GQM Prozess beginnt mit der Charakterisierung des Organisations- und Projektumfeldes. Unter Berücksichtigung des Umfeldes werden im zweiten Schritt Informationsbedürfnisse mittels Zielen und korrespondierenden Fragen erfasst. Anschließend werden im dritten Schritt Messungen dokumentiert, die der Quantifizierung dieser Informationsbedürfnisse dienen. Im vierten und fünften Schritt werden die Messungen durchgeführt und die resultierenden Daten interpretiert. Abschließend findet im sechsten Schritt eine Nachbereitung statt. Dabei werden beispielsweise die Qualitßtsplanung und gewonnene Erkenntnisse gesichert.

Das Metamodell ist zweigeteilt und auf Basis von Entity Relationship Diagrammen modelliert. Das SE (Software Engineering) Metamodell enthält Konzepte zur Charakterisierung des Umfeldes (vgl. Schritt 1). Das GQM Metamodell setzt sich aus den Konzepten *Ziel*, *Frage* und *Metrik* zusammen und bezieht sich somit auf Schritt 2 und 3. Ziele werden im GQM Ansatz mit einer Qualitätscharakteristik in Beziehung gesetzt. Auf diese Weise wird ein Qualitäts-Modell impliziert, dass genau eine Ebene aufweist.

Die beiden Teile des Metamodells beschreiben das logische Schema der Experience Factory. Die Experience Factory enthält die Wissensbasis, um Anwender des GQM Ansatzes bei der Erstellung von Qualitätsplanungen und der Interpretation von Messdaten zu unterstützen (vgl. Schritt 4, 5 und 6).

In [9] wird der GQM Ansatz um eine Definition von Indikatoren erweitert und dadurch der GQ(I)M Ansatz definiert. Die Indikatoren sind zwischen den Fragen und den Messungen angeordnet und sollen einerseits die Auswahl effektiver Messungen und andererseits die Interpretation der Messergebnisse erleichtern.

GQM++ verfeinert den GQM Ansatz um die optionale Möglichkeit, Ziele durch Subziele, Fragen durch Subfragen und Metriken durch Submetriken zu

verfeinern (vgl. [10] and [11]). Durch diese Verfeinerungen wird ein systematisches Ableiten von Messungen gefördert. Zudem lässt sich auf diese Weise eine höhere Genauigkeit der Qualitätsplanung erzielen. Z.B. werden die Abhängigkeiten zwischen Messungen und entsprechenden Fragen detailliert. Wie in [12] beschrieben, ermöglicht GQM++ durch die Verfeinerung von Zielen in Subziele zusätzlich die Definition eines mehrschichtigen Qualitäts-Modells.

# 4 Lösungsansatz

## 4.1 GQM-Spezialisierung für Software-Modelle

GQM ist ein sehr genereller Qualitätsmanagementansatz und berücksichtigt aufgrund seiner Positionierung Besonderheiten von Software-Modellen nicht. Deshalb spezialisieren wir GQM für die Qualitätsplanung von Software-Modellen (vgl. auch [13]).

Damit unser Ansatz ein *gemeinsames Qualitätsverständnis*, die Identifizierung der *Gesamtheit der Qualitätscharakteristiken*, die *Kombination von Qualitätsattributen in Qualitäts-Modellen mit Messungen und Indikatoren* und ein *effizientes Qualitätsmanagement* fördert, spezialisieren wir die in Kapitel 3 vorgestellten GQM-Ansätze. Unsere Spezialisierung betrifft insbesondere folgende Punkte:

*Kontext von Software-Modellen:* Es sind nur wenige Aspekte des SE Metamodells für unseren Lösungsansatz interessant. Stattdessen konzentrieren wir uns auf Erkenntnisse der Modelltheorie (vgl. [14] und [15]) und des Konsistenzmanagements von Software-Modellen (vgl. [16]). Auf dieser Basis haben wir ein Kontext-Metamodell entwickelt, damit der Kontext eines zu betrachtenden Software-Modells detailliert spezifiziert werden kann. Anschließend verwenden wir diesen Kontext, um Ziele sowie damit zusammenhängende Qualitätscharakteristiken weitestgehend *vollständig* und *effizient* abzuleiten.

*Qualitäts-Modelle:* Messungen werden in den GQM-Ansätzen mit Fragen und nicht mit Qualitätsattributen in Verbindung gesetzt. In GQM++ und darauf basierenden Ansätzen entsteht ein mehrschichtiges Qualitäts-Modell auch nur dann, wenn Ziele durch Subziele verfeinert werden. Zudem besteht ein weiteres Problem bei der Verwendung von Qualitäts-Modellen. Die Bedeutung einiger Begriffe, die Qualitätscharakteristiken und -attribute beschreiben sollen, ist zum Teil überladen oder unklar.

Aus diesen Gründen weisen wir Fragen einen Qualitätsschwerpunkt in Form von Qualitätsattributen zu. Für alle Elemente des Qualitäts-Modells fordern wir zudem, dass sie definiert werden müssen. Dadurch fördern wir ein *gemeinsames Qualitätsverständnis* und ermöglichen die *Kombination von Qualitätsattributen in Qualitäts-Modellen mit Messungen und Indikatoren.*

*Experience Factory:* Die Konzentration auf Software-Modelle wirkt sich gleichfalls auf die Experience Factory aus. Für Software-Modelle alleine kann eine höhere Abdeckung der relevanten Bestandteile einer Qualitätsplanung erzielt werden, als dies für alle am Entwicklungsprozess beteiligten Produkte, Ressourcen und Prozesse möglich ist. Dieses Erfahrungswissen kann helfen, abhängig

vom gegebenen Kontext, Vorschläge für die Qualitätsplanung zu unterbreiten. Aus diesem Grund messen wir der Kontextdokumentation eine sehr hohe Bedeutung für die *Effizienzsteigerung* während der Erstellung von Qualitätsplanungen bei.

*Formalisierung des Ansatzes:* Die zuvor genannten Aspekte beeinflussen das von uns erstellte Metamodell signifikant. Folglich müssen wir im Vergleich zu den GQM-Ansätzen zahlreiche inhaltliche Änderungen auf Struktur- und Prozessebene vornehmen. Die UML gilt als de facto Standard in der modellbasierten Softwareentwicklung und wir erwarten, dass die Anwender unseres Ansatzes über UML Kenntnisse verfügen. Aus diesem Grund definieren wir unseren Ansatz weitestgehend mit Hilfe von UML-Diagrammen und können Strukturen und Verhalten basierend auf einem einzigen, integrierten UML-Modell beschreiben.

### 4.2 Kurzbeschreibung

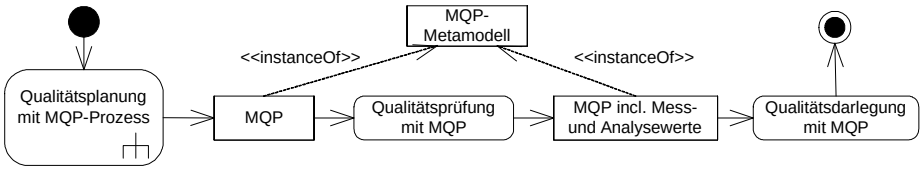


**Abbildung 1.** Übersicht Lösungsansatz

Das Ergebnis unseres Qualitätsmanagementansatzes bezeichnen wir als Modell-Qualitäts-Plan (MQP). Der MQP-Prozess stellt die GQM-Prozessspezialisierung dar und dient als Leitfaden bei der Erstellung eines MQPs. Das MQP-Metamodell stellt die Zusammenhänge zwischen den verwendeten Konzepten her und spezifiziert die möglichen Inhalte eines MQPs (vgl. Abb. 1).

Unser MQP-Prozess ist ein inkrementeller und iterativer Prozess (vgl. Abb. 2). Er beginnt mit der Dokumentation des Kontextes, um die Besonderheiten des betrachteten Software-Modells in seinem Entwicklungskontext festzustellen. Die Identifikation von Zielen und Fragen stellt in GQM eine sehr kreative Tätigkeit dar. Deshalb nutzen wir die vorgelagerte Kontextdokumentation dazu, Ziele systematisch abzuleiten. Die Ziele und Fragen werden mit Qualitätscharakteristiken und -attributen in Beziehung gesetzt. Dieses Vorgehen ermöglicht entweder die Auswahl eines geeigneten Qualitäts-Modells und dessen Anpassung oder die Erstellung eines eigenen. Die Definitionen der entsprechenden Charakteristiken und Attribute sowie deren Abhängigkeiten bilden das grundlegende Qualitätsverständnis. Für die Quantifizierung der Qualitätsattribute unterscheiden wir Basiskennzahlen, abgeleitete Kennzahlen und Indikatoren (vgl. [17]). Abschließend findet die Planung der Darlegung statt, in der Mess- und Analyseergebnisse unter Berücksichtigung des Qualitätsverständnisses, der Informationsbedürfnisse und des Kontextes aufbereitet werden.

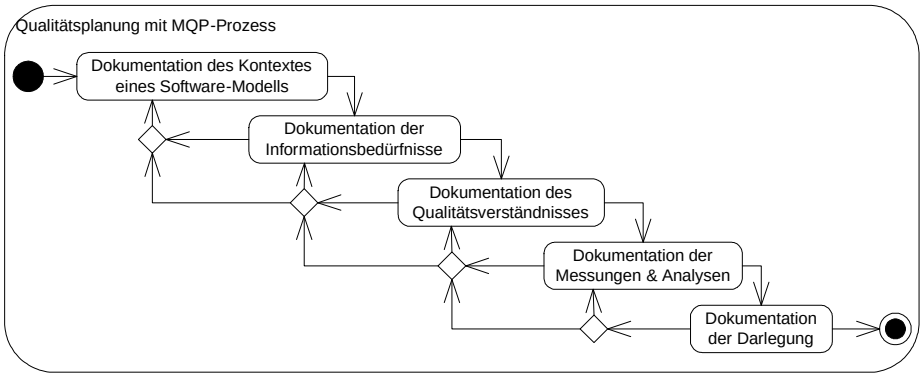


**Abbildung 2.** Übersicht MQP-Prozess

Das MQP-Metamodell besteht zur Zeit aus ca. 50 Klassen, die in fünf Pakete unterteilt sind. Jedes Paket fasst Konzepte fachlich zusammen. Auf Basis des MQP-Metamodells verknüpfen wir Konzepte aus mehreren Ansätzen (GQM, Qualitäts-Modelle, Mess-Theorie) und ergänzen es zusätzlich um Konzepte zur Kontextdokumentation. Das entsprechende Paketdiagramm mit einigen zusätzlichen Details zum Paket *Kontext-Metamodell* ist in Abb. 3 skizziert.

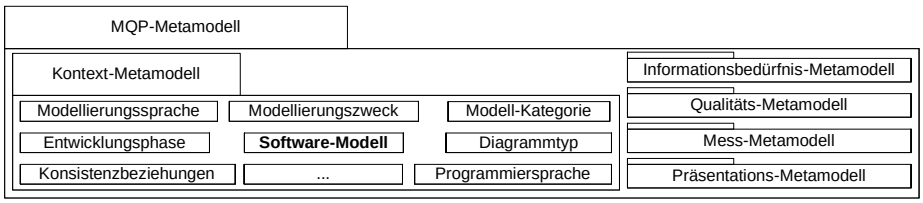


**Abbildung 3.** Übersicht MQP-Metamodell

MQPs haben aufgrund des Detaillierungsgrades einen initial erheblichen Erstellungsaufwand. Dies beeinflusst die Herausforderung *Effizienz* negativ. Trotzdem sind wir davon überzeugt, dass eine auf dieser Basis durchgeführte Qualitätsbewertung aus zwei Gründen langfristig effizient sein wird. Durch die explizite Betrachtung der Kontextbedingungen ergibt sich zum Einen ein hohes Wiederverwendungspotential. Zum Anderen können wir den Erstellungsaufwand durch eine geeignete Werkzeugunterstützung erheblich reduzieren.

### 4.3 Beispiel

Die folgenden Beispiele in Abb. 4 stellen zwei Qualitätsplanungen basierend auf unserem MQP Ansatz gegenüber. Die linke Seite zeigt einen MQP Ausschnitt für

ein *Software-Modell repräsentiert als Klassendiagramm*. Die rechte Seite dagegen legt einen MQP Ausschnitt für ein *Software-Modell repräsentiert als Statechart Diagramm* dar. Die beiden MQPs sind größtenteils identisch. Obwohl sich Ziel, Frage und Qualitäts-Modell gleichen, sind die zu messenden Kennzahlen verschieden. Die Ursache hierfür liegt im Kontext. Diese Gegenüberstellung hebt die Relevanz der Kontextdokumentation hervor.

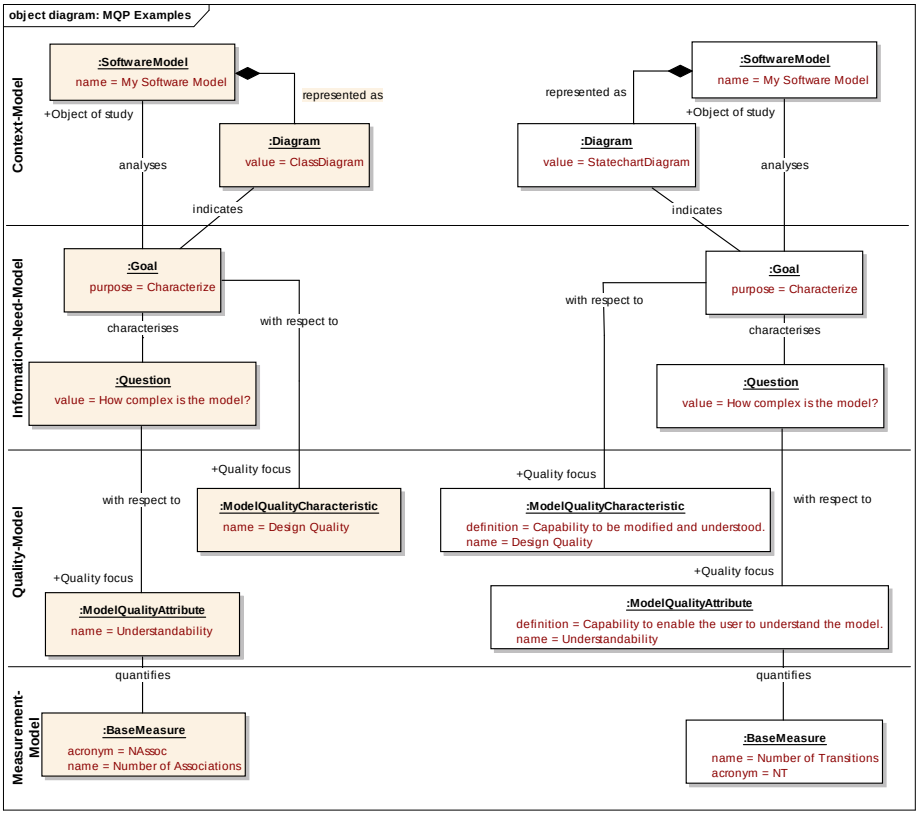


**Abbildung 4.** Gegenüberstellung zweier MQP Beispiele

## 4.4 Aktuelle Arbeiten

Eine einjährige studentische Projektgruppe von 11 Studierenden mit dem Titel *Model Cockpit* arbeitet zur Zeit an der Universität Paderborn an einer Werkzeugunterstützung und setzt die theoretischen Arbeiten prototypisch um. Im Rahmen dieser Projektgruppe entsteht ein Eclipse-Plugin, mit dem MQPs erstellt und auf Software-Modelle angewandt werden können.

Ergänzend zu den Implementierungsarbeiten wird bestehende Literatur gesichtet, auf Basis des MQP-Metamodells klassifiziert und in einem Media-Wiki abgelegt. Diese Informationen werden anschließend synthetisiert und in die MQP-Komponente als Wissensbasis integriert, damit das Werkzeug einen MQP-Ersteller auch fachlich unterstützen kann.

An dieser Stelle möchten wir uns für die informativen und konstruktiven Kommentare der Reviewer bedanken.

## Literatur

1. ISO/IEC: International organization for standardization (iso) and international electrotechnical commision (iec) 9126:2001: Software engineering product quality part 1: Quality model (2001)
2. Reißing, R.: Bewertung der Qualität objektorientierter Entwürfe [Assessment of the Quality of Object-Oriented Designs]. PhD thesis, Universität Stuttgart, Fakultät Informatik (2002)
3. Basili, V.R., Caldiera, G., Rombach, H.: The goal question metric approach. Encyclopedia of Software Engineering (1994) pp. 528–532
4. Basili, V.R., Oivo, M.: Representing software engineering models: The tame goal-oriented approach. Technical Report UMIACS-RE-91-155, CS-TR-2798, University of Maryland (October 1991)
5. Basili, V.R., Rombach, H.D.: The tame project: Towards improvement-oriented software environments. IEEE Trans. Software Eng. **14**(6) (1988) 758–773
6. Jeffery, D.R., Basili, V.R.: Validating the tame resource data model. In: ICSE. (1988) 187–201
7. Oivo, M., Basili, V.R.: Representing software engineering models: The tame goal oriented approach. IEEE Trans. Software Eng. **18**(10) (1992) 886–898
8. Briand, L.C., Differding, C.M., Rombach, H.D.: Practical guidelines for measurement-based process improvement. Special issue of International Journal of Software Engineering & Knowledge Engineering **2** (1997)
9. Park, R.E., Goethert, W.B., Florac, W.A.: Goal-Driven Software Measurement A Guidebook. Software Engineering Institute Carnegie Mellon University, Pittsburgh, PA 15213. (August 1996)
10. Gray, A., MacDonell, S.G.: Gqm++ a full life cycle framework for the development and implementation of software metric programs (1997)
11. MacDonell, S.G.: Deriving relevant functional measures for automated development projects (1993)
12. Gresse, C., Hoisl, B., Wuest, J.: Stti-report: A process model for gqm-based measurement. Technical report, University of Kaiserslautern, Department of Computer Science (1995)
13. Voigt, H., Engels, G.: Kontextsensitive qualitätsplanung für software-modelle. In: Modellierung. LNI, GI (2008)
14. Stachowiak, H.: Allgemeine Modelltheorie. Springer-Verlag, Wien (1973)
15. Ludewig, J.: Models in software engineering. Software and System Modeling **2**(1) (2003) 5–14
16. Küster, J.M.: Consistency Management of Object-Oriented Behavioral Models. PhD thesis, University of Paderborn (March 2004)
17. ISO/IEC: International organization for standardization (iso) and international electrotechnical commision (iec) 15939:2002: Software engineering software measurement process (2002)

# Eine Forschungsagenda für Softwarequalität

Stefan Wagner[1], Manfred Broy[1], Florian Deißenböck[1], Michael Kläs[2], Peter Liggesmeyer[2], Jürgen Münch[2], and Jonathan Streit[3]

[1] Institut für Informatik, Technische Universität München,
Boltzmannstr. 3, 85748 Garching b. München
[2] Fraunhofer IESE,
Fraunhofer-Platz 1, 67663 Kaiserslautern
[3] itestra GmbH,
Ludwigstrasse 35, 86916 Kaufering

**Zusammenfassung** Softwarequalität ist für die Praxis, gerade in einem Technologiestandort wie Deutschland, eine große Chance aber auch eine enorme Herausforderung. Qualitätsfragen haben einen starken wirtschaftlichen Einfluß, aber durch die Komplexität und Vielschichtigkeit von Qualität sind sie schwer greifbar. Diese Papier stellt das Ergebnis der Diskussionen von Qualitätsexperten aus Forschung und Praxis dar, welche Fragestellungen in der Zukunft am dringlichsten im Bereich der Softwarequalität bearbeitet werden müssen, damit sowohl der Stand der Forschung als auch der Technik entscheidend verbessert werden kann.

## 1 Einleitung

Die hohe Qualität von Ingenieursprodukten ist ein traditionelles deutsches Markenzeichen, das immer wieder zur Auszeichnung als Exportweltmeister verhilft. Leider ist dieser Ruf stark auf Maschinenbau-lastige Unternehmen beschränkt und überträgt sich auf deutsche Softwareprodukte nicht immer. Interessanterweise wird aber gerade auch im Maschinenbau Software immer wichtiger. Nicht nur in Deutschland, sondern weltweit, steht Software im Ruf häufig Qualitätsprobleme zu haben.

Dem gegenüber steht aber die hohe wirtschaftliche Bedeutung der Qualität von Software [1]. Beispielsweise wird in [2] dargelegt, dass der Erfolg der deutschen Softwareindustrie von ihrer Fähigkeit mit Qualität umzugehen abhängt. Softwarefehler haben immer wieder hohe Kosten durch Rückrufaktionen verursacht und können auch volkswirtschaftlich enorme Schäden anrichten. Auch um im globalen Konkurrenzkampf bestehen zu können wir eine Differenzierung über Qualität immer wichtiger. Es besteht also ein umfangreicher Handlungsbedarf sowohl in der Forschung, als auch in der Praxis.

Jedoch stellt sich immer noch für jeden, der sich mit Softwarequalität beschäftigt zuvorderst die Frage: Was ist Qualität? Es gibt verschiedenste Blickwinkel, aus denen Produktqualität betrachtet werden kann. Garvin legte das in seinem bekannten Papier [3] eingängig dar. Solch grundlegende Fragen müssen weiterhin betrachtet werden. Qualitätsmodelle sind das Mittel der Wahl, um genauer darzustellen, was für Attribute für Softwarequalität entscheidend sind. Dabei gibt es

bestehende Standards, wie die ISO-Norm 9126 [4], firmenspezifische Standards, aber auch neuere akademische Ansätze [5].

Aber auch in diesen Qualitätsmodellen gibt es verschiedenste Fragen, die noch zu beantworten sind. Beispielsweise ist die Aggregation von Teilergebnissen in Qualitätsmodellen zu höherwertigen Aussagen immer noch nicht eindeutig geklärt. Eine andere Frage, die sich durch die Diskussionen zieht, ist der Zusammenhang zu Kosten/Nutzen-Modellen. Müssen Kosten im Zusammenhang mit Qualität berücksichtigt werden?

Dieses Papier hat das Ziel die dringendsten Fragen im Umfeld von Softwarequalität nicht nur, aber insbesondere, in Deutschland zusammenzutragen. Aus diesen Forschungsfragen leiten wir dann eine Forschungsagenda für Softwarequalität ab, die der Community aus Forschung und Praxis eine gemeinsame Zielrichtung geben soll.

Wir haben hierzu im Workshop "Software-Qualitätsmodellierung und -bewertung" im Zusammenhang mit der SE 2008 in ausführlichen Diskussionen drei große Herausforderungen identifiziert. Erstens stellt die Anpassung von Qualitätsmodellen ein noch weitgehend unbearbeitetes, aber praktisch relevantes Thema dar, da Software in verschiedensten Domänen und Kontexten eingesetzt wird. Zweitens ist es wichtig, eine klare Strategie bei der Einführung von Qualitätsmodellen in Unternehmen zu haben, da von der Akzeptanz bei den Entwicklern der Erfolg des Modells abhängt. Schließlich drittens ist das Messen von Qualität, selbst mithilfe von Qualitätsmodellen, noch oft unsystematisch. Hierzu fehlen noch die richtigen Maße und Meßmethoden.

In den Qualitätsmodellen läßt sich meist beobachten, dass nur Defizite beschrieben werden, aber keine Möglichkeiten und Stellschrauben, um diese zu beeinflussen oder zu beheben. Weiterhin werden in verbreiteten Modellen Kosten nicht explizit berücksichtigt, obwohl dies offensichtlich ein wichtiger Aspekt in ihrer Anwendung in kommerziellen Projekten ist. Weiterhin wurde in verschiedenen Projekten die Erfahrung gemacht, dass in der konkreten Anwendung Qualitätsmodelle sehr groß werden können, da sie umfangreiche Informationen zu verschiedensten Qualitätsaspekten abdecken müssen. Trotz dieser Kritikpunkte wurde aber auch die Erfahrung gemacht, dass Qualitätsmodelle sehr wertvolle Informationen liefern, die im Entwicklungsprozess noch besser nutzbar gemacht werden müssen.

In den folgenden Abschnitten 2–4 stellen wir die drei Hauptthemen im Detail dar. Im Abschnitt 5 fassen wir die Forschungsfragen in einer Forschungsagenda zusammen und schließen mit einer Zusammenfassung in Abschnitt 6.

## 2 Anpassung von Qualitätsmodellen

Software ist in den verschiedensten Bereichen und Domänen im Einsatz, wodurch auch sehr unterschiedliche Anforderungen an die Software-Systeme gestellt werden. Dies trifft auch auf Qualitätsanforderungen zu, die im Zusammenhang zu einem Qualitätsmodell definiert werden müssen. Die normierten Modelle, wie die ISO 9126, bieten heute keine konkreten Vorgaben, wie sie an diese Gegebenheiten

anzupassen sind. Dies ist ein starker Hinderungsgrund gegen die operationalisierte Verwendung von Qualitätsmodellen in Software-Projekten. Wir haben hier bewußt das Wort "Anpassung" gewählt, da die Alternative "Tailoring" oft nur für das Weglassen von Teilen, aber nicht für die Erweiterung steht, was aber in diesem Kontext auch wichtig ist.

### 2.1 Erfahrungen und Herausforderungen

Es existieren bereits eine Reihe von Qualitätsmodellen aus verschiedensten Quellen. Modelle finden sich in der wissenschaftlichen Literatur [5–9] und in verschiedenen allgemeinen und Domänen spezifischen Standards wie [4, 10]. Darüberhinaus definieren Unternehmen, die sich weitergehend mit Softwareentwicklung beschäftigen typischerweise firmenintern Richtlinien, die auf die konkreten Anforderungen in den jeweiligen Projekten angepaßt sind. Schließlich kann auch der Einsatz von Mess-Werkzeugen ein entsprechendes Qualitätsmodell implizieren.

Grundsätzlich bewegt man sich bei Qualitätsmodellen für die Software-Entwicklung innerhalb des Spektrums zwischen der kompletten Eigendefinition des Modells und der direkten Nutzung eines bestehenden Modells. Ersteres erlaubt das genaue Zuschneiden auf die konkreten Bedürfnisse, bedarf aber auch eines enormen Aufwands und entsprechender Expertise. Letzteres hingegen minimiert die Aufwände, kann aber erhebliche Probleme verursachen, da das eingesetzte Modell eventuell nicht auf die eigentlichen Gegebenheiten im Unternehmen oder Projekt passt.

Aus diesen Gründen wird ein Qualitätsmodell als sinnvoll erachtet, dass ein normiertes Grundmodell definiert, aber auch vorgegebenen Prozess umfasst, der die Anpassung an die eigenen Anforderungen und Gegebenheiten ermöglicht. Zur Entwicklung eines solchen Modells, müssen aber noch einige grundlegende Fragen geklärt werden.

Zuvorderst muss festgelegt werden, was genau unter einem Qualitätsmodell zu verstehen ist. Dazu existieren sehr unterschiedliche Ansichten. Definitionen aus Standards [11] helfen hier aufgrund ihrer sehr allgemeinen Formulierung nur wenig weiter. Für viele sind die Definition und Verfeinerung von "-ilities", wie *Reliability* oder *Functionality* der ISO 9126, ausreichende Bestandteile eines Qualitätsmodells. Andere Ansätze gehen diesbezüglich wesentlich weiter und beschreiben auf sehr detaillierter Ebene Wirkzusammenhänge [5] oder den Zusammenhang zu automatisierten Qualitätsüberprüfungen [12]. Daher wäre also ein einheitliches Meta-Modell wünschenswert, dass notwendige und optionale Bestandteile eines Qualitätsmodells beschreibt.

Die Methoden *Factors-Criteria-Metrics* (FCM) [13] und *Goal/Question/Metric* (GQM) [14] beschreiben stärker diese Meta-Ebene sowie ein generisches Vorgehen zur Entwicklung angepasster Messsysteme. Nur der SQUID-Ansatz [15] beschreibt derzeit ein Vorgehen zur spezifischen Definition von Qualitätsmodellen, bietet aber weniger Details zu deren Operationalisierung. Es fehlt heute eine detaillierte Untersuchung wo überall in Qualitätsmodellen Möglichkeiten zur Anpassung berücksichtigt werden und welche Kriterien diese Anpassungen

notwendig machen. Es ist also eine Herausforderung Einflussfaktoren auf die Anpassung zu finden und ihre Auswirkungen zu untersuchen.

### 2.2 Einsatz von Qualitätsmodellen

Um eine sinnvolle Anpassung von Qualitätsmodellen zu ermöglichen, muss zuerst noch der Einsatz von Qualitätsmodellen in der Software-Entwicklung diskutiert werden. Generell stellen Qualitätsmodelle eine Abstraktion von Qualitätsmerkmalen für Software dar. Sie verallgemeinern von Qualitätsdefekten und bieten auch eine Grundlage für Messungen. Dafür sollte eine Hierarchie an Nutzungsszenarien entwickelt werden, die jeweils aufeinander aufbauen. Beispielsweise könnte das Modell zuerst verwendet werden, um Qualität zu *verstehen*. Die komplexen und vielschichtigen Aspekte von Software-Qualität sollten in einem Qualitätsmodell dabei so beschrieben sein, dass das Verständnis verbessert wird. Darauf aufbauend kann Qualität besser *definiert* werden. Nur ein strukturiertes Vorgehen, wie es ein Qualitätsmodell bietet, erlaubt überhaupt eine sinnvolle Definition von Qualität. Diese Definitionen können wiederum als Ausgangspunkt für die *Analyse* von Qualität verwendet werden. Das Modell kann hierbei Indikatoren und Metriken definieren und bei der Interpretation der Messergebnissen unterstützen. Weiterhin sollte das Modell auch Wege zur *Verbesserung* der Qualität aufzeigen und im Idealfall helfen Qualität *vorherzusagen*.

Grundsätzlich lässt sich festhalten, dass Qualitätsmodelle kein Selbstzweck sein dürfen, so wie auch das Messen von Qualität kein Selbstzweck sein kann. Vielmehr müssen sie immer als Entscheidungsgrundlage im Software-Projekt dienen. Beispielsweise können sie helfen eine *Make or Buy*-Entscheidung zu treffen oder festzulegen, ob eine Software reif zur Auslieferung ist.

### 2.3 Empfehlungen

Ein wichtiger Punkt in der Anpassung von Qualitätsmodellen ist die Identifizierung der Anpassungsdimensionen bzw. der Einflussfaktoren. Insbesondere spielt es eine Rolle, was die Ziele des Unternehmens oder Projekts sind. Danach muss untersucht werden, wie sich dies im Qualitätsmodell auswirkt und das Modell entsprechend angepasst werden. Insbesondere ist wichtig diese Anpassungen zu operationalisieren. Schließlich muss eine Validierung stattfinden, also überprüft werden, ob die Anpassungen sinnvoll sind. Dies kann beispielsweise durch Rückkopplung mit den Entwicklern und Qualitätssicherungspersonal geschehen.

Für eine längerfristige Anwendung ist auch die Integration in das verwendete Prozessmodell notwendig. Es muss beispielsweise festgelegt werden, an welchen Entscheidungspunkten (Quality-Gates) das Modell zum Einsatz kommt oder wann Messungen stattfinden. Zuletzt kann noch eine konkrete Anreicherung um Kosten erfolgen, die monetäre Schätzungen und Bewertungen erlaubt.

Es ist unwahrscheinlich, dass zu Beginn eines Projekts entweder die Qualitätsanforderungen oder die Qualitätsmodelle vollständig verfügbar sind. Deshalb scheint ein *Bootstrapping*-Verfahren am sinnvollsten: Anforderungen und

Qualitätsmodelle werden im Projektverlauf iterativ und inkrementell abgeglichen [16, 17]. Dabei werden auch Prioritäten und Gewichtungen der einzelnen Teile des Qualitätsmodells festgelegt, was wiederum das Budget für Qualität bestimmen könnte, wie im *Activity-based Costing*. Bei festgelegtem Budget könnte dieses natürlich wiederum die Gewichtungen bestimmen.

### 2.4 Forschungsfragen

Aus diesen Herausforderungen und Empfehlungen können nun eine Reihe von noch offenen Forschungsfragen abgeleitet werden, deren Beantwortung als notwendig für eine praktische Anpassung von Qualitätsmodellen gesehen wird. Zuerst muss untersucht werden, welches Spektrum an Qualitätsmodellen zur Zeit existiert. Dies ist notwendig, um zu entscheiden, wie flexibel die Anpassungsmechanismen sein müssen. Ähnlich dazu muss analysiert werden, welche Komponenten ein Qualitätsmodell notwendigerweise und optional umfasst. Dazu gehört einerseits die Struktur eines Qualitätsmodells, andererseits auch welche Inhalte zwingend und welche optional sind. Insbesondere im Bezug auf Kosten stellt sich die Frage, welche Beziehung zwischen bestehenden Kostenrechnungs -und Qualitätsmodellen besteht, da dies ein wichtiger Faktor in der Anpassung sein knnte.

Eine weitere wichtige Frage ist, wie groß die Variabilität in Qualitätsmodellen über Domänen, Projekte, Technologien hinweg überhaupt ist. Aus den dargestellten Erfahrungen ergibt sich eine signifikante Variabilität, dies wurde aber noch nicht wissenschaftlich untersucht. Diese Information ist jedoch wichtig ist um einen brauchbaren Anpassungsmechanismus abzuleiten. Daran anschließend ist noch unklar, wie ein umfassendes und umfangreiches Qualitätsmodell (nach seiner Anpassung) validiert werden kann. Eine vollständige empirische Überprüfung kann aufgrund des dafür nötigen Aufwands sicher ausgeschlossen werden.

## 3 Einführung von Qualitätsmodellen

Unklar ist bisher, welches die nötigen Schritte zur Einführung von Qualitätsmodellen in Unternehmen sind. Dies ist nicht nur durch die allgemeine Uneinigkeit bzgl. einer geeigneten Operationalisierung von Qualitätsmodellen bedingt, sondern vor allem durch die langfristige Natur der Fragestellungen, die von ihnen adressiert werden. Abstrakt betrachtet, hat die Einführung von Qualitätsmodellen das Ziel, "Qualitätseigenschaften die Freiwilligkeit zu nehmen". Das heißt, ähnlich wie bei funktionalen Anforderungen soll erreicht werden, das Qualitätsanforderungen selbstverständlich umgesetzt und nicht nach belieben weggelassen oder verändert werden.

### 3.1 Erfahrungen und Herausforderungen

Qualitätsmodelle werden verwendet um in einem gegebenen Projekt- oder Unternehmenskontext ein gemeinsames Verständnis von Produkt-Qualität zu eta-

blieren. Ihr Einsatz erfolgt meist im Rahmen von langfristig angelegten Qualitätsinitiativen und nur selten zur Behebung akuter Probleme. Aufgrund dieser Langfristigkeit ergeben sich eine Reihe von Problemen bzgl. der Motivation der beteiligten Personen, die den unmittelbaren Nutzen der Initiativen nicht einfach erkennen können. Verstärkt wird dieses Problem durch den Umstand, dass solche Qualitätsinitiativen von den meisten Beteiligten, zumindest zu Beginn, eine gewisse Mehrarbeit erfordern, von Managementseite jedoch selten die entsprechenden Ressourcen zur Verfügung gestellt werden.

Darüber hinaus wurde mehrfach festgestellt, dass nicht alle Projektbeteiligten über die entsprechende Ausbildung verfügen, die es Ihnen erlauben würde, den langfristigen Nutzen von Qualitätsinitiativen zu erkennen bzw. zu realisieren, was die Nicht-Einhaltung von Qualitätsstandards zur Folge hat.

Ein weiteres Problem bei der Einführung von Qualitätsmodellen können sich durch die organisatorische Trennungen von Mitarbeitern unterschiedlicher Prozessphasen ergeben. So ist es u. U. schwierig Entwickler, die nie mit Wartungsaufgaben betraut werden, von den Vorteilen von lesbarem Code zu überzeugen.

Neben diesen motivatorisch-personellen Problemen stellt die in Abschnitt 2 diskutierte Inflexibilität heutiger Qualitätsmodelle eine Schwierigkeit dar, da sie eine Anpassung auf die projektspezifischen Qualitätsbedürfnisse erschwert und damit weder zur Effektivität noch zur Effizienz des Qualitätsmanagements beiträgt.

### 3.2 Empfehlungen

Dieser Inflexibilität kann mit den im letzten Abschnitt diskutierten Anpassungsmechanismen entgegnet werden. Aus Sicht der Einführung von Qualitätsmodellen sind hierbei folgende Dimensionen der Anpassung von Bedeutung:

- *Projekt-spezifische Anpassung.* Eine Anpassung des Qualitätsmodells an das Projekt ist wichtig um den Projektbeteiligten zu demonstrieren, dass sich bei dem eingesetzten Qualitätsmodell nicht um einen bürokratischen Selbstzweck sondern um ein Mittel zur langfristigen Steigerung der Projekt-Produktivität handelt.
- *Mitarbeiter-spezifische Sichten.* Neben der tatsächlichen Anpassung sollte das Qualitätsmodell die Möglichkeit bieten für die unterschiedlichen Projektbeteiligten unterschiedliche Sichten zu definieren. Dies ermöglicht es, jedem Projekt-beteiligten nur den für ihn relevanten Anteil zur Verfügung zu stellen und ihn nicht mit unnötiger Information zu belasten.
- *Konfiguration der Überprüfungsmethoden.* Speziell automatische Überprüfungsmethoden für Qualitätskriterien leiden oft unter einer hohen Zahl von *False Positives*, die schnell zu einer Demotivation der Projektbeteiligten führen können. Daher ist es notwendig, Überprüfungswerkzeuge und -methoden so auf den jeweiligen Kontext anzupassen, dass die Anzahl der *False Positives* auf ein erträgliches Maß (<5%) beschränkt wird.

Darüber hinaus wird empfohlen, die Einführung von Qualitätsmodellen möglichst behutsam durchzuführen, um die Beteiligten nicht durch eine Fülle von

Neuerung zu überfordern. Als äußerst hilfreich hat es sich erwiesen, einen unmittelbaren Nutzen für einzelne Beteiligten zu identifizieren, da dieser oft dazu beiträgt, die Bereitschaft der Beteiligten auch für langfristigere Neuerungen, ohne unmittelbaren Nutzen, zu erhöhen. Jenseits dieser unmittelbaren Nutzen, muss natürlich dafür gesorgt werden, dass alle Beteiligten ein entsprechendes Verständnis für die Ziele, die mit Hilfe von Qualitätsmodellen erreicht werden sollen, haben und sich inbesondere über die langfristigen Konsequenzen einer Nicht-Berücksichtigung von Qualitätseigenschaften bewusst sind.

### 3.3 Forschungsfragen

Aus diesen Erkenntnissen und Empfehlungen ergeben sich eine Reihe von Forschungsfragen die bzgl. der Einführung von Qualitätsmodelle relevant sind. Neben den eher technischen Fragen bzgl. der verbesserten Anpassbarkeit und besseren Werkzeugunterstützung von Qualitätsmodellen spielen hierbei vor allem sozio-psychologische Aspekte *(Change Management)* und ein tiefergehendes Verständnis von Kosten-Nutzen-Zusammenhängen eine Rolle.

Hierbei wäre insbesondere eine strukturierte Aufarbeitung der Erkenntnisse aus anderen Disziplinen, z. B. Prozesseinführung, ein vielversprechender Startpunkt um besser zu verstehen, welchen Probleme die Einführung von Qualitätsmodellen auf einer sozio-psychologische Ebenen begegnet.

Darüber hinaus würde eine systematische Aufarbeitung Kosten-Nutzen- Zusammenhänge im Bereich der Software-Qualität sicherlich die Einführung von Qualitätsmodellen deutlich erleichtern. Durch die Bereitstellung von Kosten-Nutzen-Modellen, die in der Lage sind quantitative Aussagen bzgl. des langfristigen Nutzens von Qualitäts-Aktiväten zu treffen, könnten viele Aussagen, die heute eher anekdotischer Natur sind, auf eine sicheres Fundament gestellt werden und damit zur sachlichen Diskussion beitragen.

Jenseits dieser konkreten Vorschläge, ist es sinnvoll, die Qualitätsthemen noch stärker in der Ausbildung zukünftiger Fachkräfte zu berücksichtigen und damit langfristig eine Situation zu erreichen, in der Qualitätsmodelle gar nicht eingeführt werden müssen, da sie integraler Bestandteil eines jeden Softwareprojekts sind.

## 4 Qualitätsmessungen mit Qualitätsmodellen

Die Definition von Qualitätsmodellen und deren Nutzung bei der Qualitätsplanung und -sicherung im Software- und Systems-Engineering erfordert die Quantifizierung von Soll- und Istwerten. Die Auswahl geeigneter Metriken ist dabei von einer Menge von Faktoren (wie z.B. dem Entwicklungskontext) abhängig, ohne deren Beachtung Qualitätsmodelle einen Großteil ihres Nutzens einbüßen. Dies macht die Nutzung von derzeit existierenden Standards für Software-Qualität schwierig, da diese oftmals durch ein hohes Abstraktionsniveau Variabilität vermeiden und ihr praktischer Nutzen somit äußerst fragwürdig ist. Grundvoraussetzung für ein messbasiertes Qualitätsmanagement ist die Schaffung der hierfür

notwendigen Akzeptanz in Unternehmen und Organisationen. Die Einführung von Messprogrammen bzw. die Änderung existierender Messverfahren ist ein aufwändiger Prozess, der sorgfältig geplant und im Vorhinein auf den zu erwartenden Nutzen analysiert werden muss. Im Rahmen des Workshops wurden folgende Themen erörtert und in der angegebenen Reihenfolge bezüglich ihrer Dringlichkeit aus praktischer Sicht priorisiert:

1. Wie lassen sich Qualitätsmessungen in Unternehmen und Organisationen motivieren?
2. Wie lässt sich Qualität definieren und an den Entwicklungskontext anpassen?
3. Wie werden Qualitätsmessungen eingeführt?
4. Welche Messverfahren und Werkzeuge können zur Umsetzung genutzt werden?

### 4.1 Erfahrungen und Herausforderungen

Die Notwendigkeit von Software-Messungen und Aktivitäten für Softwarequalität ist bis heute in vielen Unternehmen und Organisationen nicht selbstverständlich. Vielfach wird der Aspekt der Software-Qualität ignoriert oder findet erst Beachtung, wenn bereits erhebliche Probleme aufgetreten sind. Häufig finden Qualitätsaspekte erst sehr spät im Entwicklungsprozess Aufmerksamkeit, in der Regel im Zusammenhang mit ersten Testprozessen. Zu diesem Zeitpunkt bestehen allerdings nur noch sehr begrenzte Möglichkeiten auf die Qualität der resultierenden Produkte Einfluss zu nehmen. Die meisten Qualitätsmängel lassen sich zu diesem Zeitpunkt nicht mehr bzw. nur noch mit sehr hohem Aufwand beheben.

Eine Ursache für das fehlende Bewusstsein für Software-Qualität in Entwicklungsorganisationen ist, dass die wirtschaftliche Bedeutung von Software-Qualität Entscheidungsträgern oftmals unklar ist. Der Bezug von Maßnahmen zur Erzielung qualitativ hochwertiger Software zu höheren Zielen einer Organisation, insbesondere zu Geschäftszielen, ist vielfach nicht explizit dargestellt. Die Kosten von Maßnahmen und Messungen sind dabei klar sichtbar, während der Nutzen in Form von zukünftigen Einsparungen und vermiedenen Folgekosten nicht offensichtlich und schwer quantifizierbar ist.

Hinzu kommt der Gegensatz zwischen der vermeintlich langfristigen Amortisierung von Qualitätsmaßnahmen und kurzfristigem Planungshorizonten auf Managementebene. Erfahrungen aus der industriellen Praxis zeigen, dass oftmals sehr hohe ROI-Erwartungen in Bezug auf Maßnahmen zur Verbesserung der Software-Qualität existieren. Im Vordergrund stehen die Kosten von Software-Qualität und Messungen. Dies hängt damit zusammen, dass Software in vielen Unternehmen als umsetzender und nicht als wertschöpfender Faktor in Bezug auf die Unternehmensziele gesehen wird. Diese Tatsache ist bedenklich, da in vielen Branchen Software mittlerweile der bedeutendste Innovationstreiber ist.

Oftmals ist Software-Qualität auch nicht relevant bei der Vergabe von Aufträgen. Bei Ausschreibungen zählt in der Regel das wirtschaftlich günstigere Angebot. Langfristige Kosten (insbesondere durch Qualitätsmängel verursachte

Kosten) werden bei der Auftragsvergabe nicht oder nicht ausreichend berücksichtigt. Dies ist unter anderem dadurch bedingt, dass die Bedeutung nichtfunktionaler Eigenschaften schwierig zu vermitteln ist und nichtfunktionale Anforderungen schwierig in vertragliche Regelwerke zu integrieren sind. Im öffentlichen Bereich gibt es mittlerweile erste Ansätze, um Wirtschaftlichkeitsuntersuchungen bei IT-Investitionen zu berücksichtigen (insbesondere das WiBe-Verfahren).

Eine weitere Erfahrung in Bezug auf Software-Messungen ist, dass viele Unternehmen sich vorzugsweise an Standards orientieren, um entsprechend des Standes der Technik zu handeln und sich so gegen Risiken (insbesondere juristische) abzusichern. Diese Standards berücksichtigen den Einsatzkontext in der Regel jedoch nur völlig ungenügend. Auch ist eine Präferenz für die Anwendung starrer Metriksätze zu erkennen. Das Bewusstsein, dass im Software-Bereich eine wesentlich höhere Anzahl von Variationsparametern als bei Produktions- und Fertigungsprozessen existiert und somit Metriken und Qualitätsmodelle wesentlich größeren Variationen unterliegen, ist oftmals nicht vorhanden.

Herausforderungen in Bezug auf die quantitative Qualitätsdefinition werden unter anderem darin gesehen, dass es für viele Qualitätsaspekte nur ungenaue, implizit definierte Begriffe gibt (z.B. Wartbarkeit, Performanz, Genauigkeit). Selbst vergleichsweise klar erscheinende Begriffe wir Größe, Kosten oder Aufwand können in unterschiedlichen Umgebungen sehr unterschiedliche Definitionen haben und ihre Quantifizierung kann sich als große Herausforderung gestalten. Insgesamt stellt sich die Ableitung geeigneter Metriken und die Minimierung der zu erfassenden Metrikmenge als schwierig da. Hierzu gehört unter anderem die Beantwortung der Fragen, wie man zielorientiert misst, wie man Einflussfaktoren auf Qualität (z.B. Erfahrung, Teamkohärenz) ermittelt und quantifiziert oder wie man weichere Faktoren (z.B. die Güte von Codekommentaren) misst.

Herausforderungen in Bezug auf den Einführungsprozess von Software-Messungen sind unter anderem ungünstige Randbedingungen (z.B. Lieferdruck, fehlende Ressourcen, geringe Organisationsgröße, fehlende Unterstützung des Managements). Der Mangel an nützlichen Standards und Anleitungen führt darüber hinaus dazu, dass es für Unternehmen schwierig ist, die für sie wichtigen Indikatoren zu ermitteln. Es ist oftmals auch nicht klar, wie bestehende Messprogramme genutzt werden können, um neue Messziele zu beantworten. Darüber hinaus fehlen in der Praxis meist Baselines, sodass ein Vergleichswerte für die Beurteilung von Software-Qualität nicht verfügbar sind.

Beispiele für Herausforderungen hinsichtlich der technischen Umsetzung von Software-Messungen sind die kosteneffiziente Erfassung von Daten (ggf. durch automatisierte Verfahren), die Integration von Messwerkzeugen in den Entwicklungsprozess und die Harmonisierung von Messungen im Falle von verteilter Entwicklung über Organisationsgrenzen hinweg.

### 4.2 Empfehlungen

Im Folgenden werden ausgewählte Empfehlungen und Hinweise zur erfolgreichen Einführung von Software-Messungen in der Praxis aufgeführt:

- Es ist wichtig, Kosten und Nutzen von Software-Qualität und deren Vermessung explizit darzulegen. Hierzu gehört die Darstellung der Bedeutung von Software-Qualität im Kontext und in Bezug auf IT- und Geschäftsziele einer Organisation.
- Metriken müssen aus Zielen abgeleitet werden und für den Kontext passen.
- Die Einsicht zur Berücksichtigung von Software-Qualität lässt sich mit Konsequenzen mangelnder Qualität in Bezug auf Mehrkosten, Risiken, oder Haftung im juristischen Sinne motivieren.
- Existierende Messprogramme sollten soweit möglich genutzt werden, um Messziele zu beantworten.
- Messergebnisse sollten in regelmäßigen Feedback-Sitzungen an die Bereitsteller von Daten kommuniziert werden. Analyseresultate sollten in Aktionen umgesetzt werden.
- Auftragnehmer sollten gegenüber ihren Auftraggebern darauf bestehen, dass auch nichtfunktionale Anforderungen frühzeitig und präzise formuliert werden.
- Wichtig für die Akzeptanz von Qualitätsmessungen in der Software- und Systementwicklung ist, dass diese als ein Mittel zum Zweck gesehen wird und nicht als Selbstzweck. Oftmals ist es hilfreich, Software-Messungen im Rahmen von Prozessverbesserungsprogrammen einzuführen.
- Qualität ist in der Regel nicht das einzige Kriterium für Entwicklungsvorhaben. Qualität steht oftmals im Konflikt mit Kosten- und Terminzielen und ein entsprechender Tradeoff ist erforderlich. Für die Software-Qualität bedeutet dies, dass ein „angemessenes“ Qualitätslevel innerhalb einer zu definierenden Toleranzschwelle erzielt werden muss.
- Software-Messungen sollten nicht mit zu ambitionierten Zielen starten (z.B. Vorhersage). Der Aufbau von Messkompetenz in Unternehmen ist in der Regel ein mehrjähriger Prozess. Als Start wird die Ermittlung von Baselines empfohlen.

### 4.3 Forschungsfragen

Aus diesen Herausforderungen und Empfehlungen ergeben sich eine Reihe von Forschungsfragen, die bezüglich der Messung von Software-Qualität relevant sind. Hierzu gehört die Definition und Erprobung von Einführungsprozessen für Software-Messprogramme, die Entwicklung einfach zu bedienender und leicht anpassbarer Messwerkzeuge, die Verbindung von Metriken auf der Engineering-Ebene einer Organisation und höheren, geschäftsrelevanten Metriken (beispielsweise über eine Balanced Scorecard definiert).

Für die praktische Umsetzung stellt sich für Unternehmen die Frage, wie sie zu den für ihre Ziele und Randbedingungen wichtigen Metriken kommen. Dies kann als zentrale Herausforderung bei der Normierung von Qualitätsmodellen angesehen werden: Wie lassen sich abstrakte, empirisch belastbare Industriestandards für Qualitätsmodelle entwickeln, die zum einen hinreichend abstrakt sind und zum anderen systematische Anleitungen zum Anpassen an Firmenkontexte ermöglichen.

## 5 Forschungsagenda

Interessanterweise ist in all diesen Diskussionen ein Punkt unstrittig geblieben: Ein Modell für die Produktqualität von Software ist zentral für den Umgang mit Qualität in der Software-Entwicklung. Das Thema ist zu diffus, um ohne klare Definitionen und Festlegungen auskommen zu können. Aus diesem Grund ist eine zentrale Aufgabe für die Forschung in den nächsten Jahren basierend auf den bestehenden Standards und neueren Ergebnissen ein Qualitätsmodell zu entwickeln, das den aktuellen Herausforderungen gerecht wird. Es müssen Fragen beantwortet werden, wie ein Qualitätsmodell überschneidungsfrei verschiedene Attribute darstellen kann oder wie mit den weitreichenden Informationsmengen umgegangen werden kann, die in einem in der Entwicklung einsetzbaren Modell erfasst werden müssen.

Neben solch eher technischen Fragestellungen, kommt ein nicht zu unterschätzender Teil an sozio-psychologischen Themen hinzu. Ein Qualitätsmodell muss gut im Unternehmen verankert sein, damit es seine voll Wirkung entfalten kann. Hierzu müssen Erfahrungen aus der allgemeinen Prozesseinführung genutzt werden, damit hier eine nachhaltige Verbesserung besteht. Dazu gehört auch die Einführung eines entsprechenden Messprozesses. Auch wenn es sich wahrscheinlich nicht lohnt alle Bereiche eines Qualitätsmodells mit genauen Messungen abzudecken, müssen die wichtigen Teile identifiziert und mit einem entsprechenden Messprogramm unterfüttert werden. Es müssen also Verfahren entwickelt werden, die einerseits Qualitätsmodelle mit Geschäftszielen verknüpft und andererseits systematisch aus Qualitätsmodellen ableiten lässt.

Zu den obigen Herausforderungen gehören noch zwei weitere Gesichtspunkte, die dort aufgetaucht sind. Für die Konzeption und Einführung von Qualitätsmodellen genauso wie für das Messen im Rahmen von Qualitätsmodellen ist es wichtig auch die Kosten/Nutzen-Zusammenhänge mit zu betrachten. Dies hilft einerseits direkt die richtigen Qualitätsaspekte zu analysieren, aber andererseits auch Argumentationshilfen für die Einführung zu liefern. Weiterhin ist eine umfassende und integrierte Werkzeugunterstützung unbedingt notwendig. Es müssen sowohl bei der Erstellung von Modellen, als auch beim Messen von Qualität mit großen Mengen an Daten und Informationen umgegangen werden. Außerdem ist es natürlich wünschenswert, die offensichtlichen Verbindungen zwischen Qualitätsanforderungen, Qualitätsmodellen und Qualitätsmessungen auch in den Werkzeugen abzubilden.

Ein weiterer großer Punkt ist die Anpassung von Qualitätsmodellen auf unternehmens- und projektspezifische Bedürfnisse. Hierzu muss die Variabilität in existierenden und zukünftigen Modellen analysiert und daraus Schlüsse auf die Einflussfaktoren gezogen werden. Insbesondere wäre es wünschenswert im Hinblick auf Wiederverwendung, aber auch Normierung, eine gemeinsame Basismenge in den Qualittsmodell-Varianten zu identifizieren.

## 6 Zusammenfassung

Software-Qualität ist ein entscheidender Faktor für die Konkurrenzfähigkeit von Unternehmen und Wirtschaftsstandorten. Leider konnte sich der gute Ruf von *Made in Germany* bisher nur bedingt auf Softwareprodukte übertragen lassen. Dies ist ein Problem, dem mit verbesserten Methoden zur Software-Qualität entgegnet werden muss.

Qualitätsmodelle sind offensichtlich ein akzeptiertes Mittel zum Umgang mit Softwarequalität. Jedoch gibt es sehr unterschiedliche Arten der Verwendung, wie auch sehr unterschiedliche Ausprägungen in den verschiedenen Domänen und Unternehmen. Für alle diese Ausprägungen stellen sich aber eine Reihe von Forschungsfragen, die beantwortet werden müssen, um auch Softwarequalität in der Praxis handhabbar zu machen.

Dieses Papier liefert hierzu eine Forschungsagenda mit den wichtigsten Punkten, die zukünftig adressiert werden müssen. Die Agenda basiert auf einer ausführlichen Diskussionen mit akademischen und industriellen Experten und spiegelt damit ein umfangreiches Erfahrungsspektrum wider. Ein Eingehen auf die gemachten Vorschläge kann die Chance eröffnen, auch für Software die deutschen Ingenieursleistungen in der Welt beliebt zu machen.

### Danksagung

Wir danken allen Teilnehmern des Workshops SQMB '08, insbesondere denjenigen, die in den Diskussionsgruppen teilgenommen haben und durch ihre wertvollen Beiträge zur Entwicklung dieser Forschungsagenda beigetragen haben.

## Literatur

1. Broy, M., Jarke, M., Nagl, M., Rombach, D.: Manifest: Strategische Bedeutung des Software Engineering in Deutschland. Informatik-Spektrum **29**(3) (2006)
2. : Deutschland schlampt bei Softwarequalität. Computer Zeitung **35**(21) (2005)
3. Garvin, D.A.: What does product quality really mean? MIT Sloan Manage. Rev. **26**(1) (1984) 25–43
4. ISO 9126-1: Software engineering – Product quality – Part 1: Quality model (2001)
5. Deissenboeck, F., Wagner, S., Pizka, M., Teuchert, S., Girard, J.F.: An activity-based quality model for maintainability. In: Proc. 23rd International Conference on Software Maintenance (ICSM '07), IEEE Computer Society Press (2007)
6. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., Macleod, G.J., Merrit, M.J.: Characteristics of Software Quality. North-Holland (1978)
7. Heitlager, I., Kuipers, T., Visser, J.: A practical model for measuring maintainability. In: Proc. Sixth International Conference on the Quality of Information and Communications Technology, IEEE Computer Society Press (2007) 30–39
8. Dromey, R.G.: A model for software product quality. IEEE Transactions on Software Engineering **21**(2) (1995) 146–162
9. Seffah, A., Donyaee, M., Kline, R.B., Padda, H.K.: Usability measurement and metrics: A consolidated model. Software Quality Control **14**(2) (2006) 159–178

10. ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability (1998)
11. ISO 14598: Information technology – software product evaluation (2000)
12. Plösch, R., Gruber, H., Hentschel, A., Körner, C., Pomberger, G., Schiffer, S., Saft, M., Storck, S.: The EMISQ method – expert based evaluation of internal software quality. In: Proc. 31st Annual IEEE/NASA Software Engineering Workshop (SEW-31). (2007) 99–108
13. McCall, J., Walters, G.: Factors in Software Quality. The National Technical Information Service, Springfield, VA, USA (1977)
14. Basili, V.R.: Software modeling and measurement: the Goal/Question/Metric paradigm. Technical Report UMIACS TR-92-96, University of Maryland at College Park (1992)
15. Kitchenham, B., Linkman, S., Pasquini, A., Nanni, V.: The SQUID approach to defining a quality model. Software Quality Control **6**(3) (1997) 211–233
16. Wagner, S., Deissenboeck, F., Winter, S.: Erfassung, Strukturierung und überprüfung von Qualitätsanforderungen durch aktivitätenbasierte Qualitätsmodelle. In: Workshop Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung. (2008)
17. Wagner, S., Deissenboeck, F., Winter, S.: Managing quality requirements using activity-based quality models. In: Proc. 5th Workshop on Software Quality (6-WoSQ), ACM Press (2008) Zur Publikation angenommen.

# Index der Autoren