INSTITUT FÜR INFORMATIK

A Formal Approach to High Quality Software Design and Development

Hannu Jaakkola [1]
Bernhard Thalheim [2]

Bericht Nr. 0903
Januar 2009

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

A Formal Approach to High Quality Software Design and Development

Hannu Jaakkola [1]
Bernhard Thalheim [2]

Bericht Nr. 0903
Januar 2009

e-mail: hannu.jaakkola@tut.fi,
thalheim@is.informatik.uni-kiel.de

[1] Tampere University of Technology, P.O.Box 300, FI-28101 Pori, Finland
[2] Christian-Albrechts-University at Kiel, Department of Computer Science, Kiel, Germany

A Formal Approach to High Quality Software Design and Development

Hannu Jaakkola[1] Bernhard Thalheim[2]

[1] Tampere University of Technology, P.O.Box 300, FI-28101 Pori, Finland
hannu.jaakkola@tut.fi
[2] Christian-Albrechts University Kiel, Olshausenstrasse 40, 24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de

Abstract

Software design and development coexist and co-evolve with quality provision, assessment and enforcement. However, most and also modern research "provides only bread and butter lists of useful properties without giving a systematic structure for evaluating them" [KLS95]. Software engineers have been putting forward several three-score quantities of metrics for software products, processes and resources whereas a theoretical foundation is still missing [PGC02]. This paper proposes a framework to quality property specification, to quality control, the quality utilisation and quality establishment. Our framework has a theoretical basis that is adaptable to all stages of software development.

Table of Contents

Chapter 1

Introduction

In the discussion of software quality two aspects are usually pointed out: software development *process* quality and software *product* quality. The expectation is that a good quality development process is a precondition for a good quality product. There are several frameworks in the area of process quality. The standard [ISO05] and the related ones provide an applicable basis to develop a company based *quality management system*. Software development oriented application of this framework is under development [ISO03]. In addition to ISO 9000 there are several other process oriented quality models, like SPICE [ISO06a] and CMMI [CMM02a, CMM02b].

Product quality is widely discussed in software engineering study books. During the life cycle of a software product, the original *user requirements* derived from the *purposes* of the product are first transformed to a software product *specification*, and further to different levels of *designs* and finally to the *implementation*. The central elements during this transformation are *functional requirements* of the software, *non-functional requirements* of the software, and *constraints* (and other requirements). The main functionality of a software product is derived - naturally - from the functional requirements. In the beginning of the software life cycle (requirements elicitation and analysis, early design) non-functional requirements are specifying the *qualitative* aspects. When the functional requirements specify the *verifiable functionality* of the software, the non-functional requirements play the role of *validity* - suitability of the product to be used in its context.

However, when the development of software forwards until later *design phases* (final architecture, detailed design) the role of *non-functional (qualitative) requirements* must have more focus. When the evolution of functional requirements in the implementation is "straightforward" transformation trough different abstractions and layers, the non-functional requirements expect a designer to *transform them in the functionality* filling the quality expectations of the software. A systematic way to make this transformation is given by [BCK03] in the '*Quality Attribute Driven Software Design Method*'. In the method, a *stimulus* activates the operations specified by the selected *quality tactics* (operations), which will cause *expected response* in the system to fill the requirements of a quality attribute.

Quality characteristics are discussed in ISO 9126 (2001-2004) [ISO01], which

Fig. 1. The SQuaRE initiative

specifies a framework to analyze software product quality from three different viewpoints (quality in use, internal quality, external quality). The standards are further developed by ISO/IEC JTC1/SC7 in a new initiative SQuaRE [ISO06d] (Software Product Quality Requirements and Evaluation quality model). It is based on the series of ISO 9126 (2001-2004) which is augmented by *data quality model* (25012). In addition it clarifies and fills the structure of product / data quality by dividing the standards in different *divisions* (Figure 1). This structure is discussed in Section 3.1 in the extent relevant to this paper.

This paper is structured as follows. In Chapter 2 we shortly introduce a general quality characteristics structure. We also introduce a *Quality Characteristics Conceptual Model* - QCCM. Chapter 3 extends the conceptual model to a management model. It is based on a specification, control, application and establishment levels. Chapter 4 illustrates the management model for one quality criterion. The model has already been validated in a project that is discussed in Chapter 5.

Chapter 2

Quality Characteristics and Concepts

2.1 Quality Model and Quality Characteristics

The SQuaRE initiative (Figure 1) is based on the earlier work and publications of ISO/IEC JTC1/SC7:

- The framework standard ISO/IEC 25010 [ISO06c] is based on [ISO01].
- There exists a new part ISO/IEC 25012 Data Quality Model.
- The Quality Measurement Division includes the development of former measurements reports as a part of ISO/IEC 25022 work. In addition the work is extended by the measurement reference model.

In this work, the detailed review of the standards is not relevant. However, it might be worth of shortly discussing the ideas presented in the CD-version of ISO 25010 [ISO06b].

The standard opens *three views* to the product quality: internal product quality, external product quality, and product in use quality. It also points out that the role of a quality characteristic is changing trough the product life cycle. For example, quality specified as quality requirements at the start of the lifecycle is mostly seen from the external and users view, and it differs from the interim product quality, such as design quality, which is mostly seen from the internal and developers view. The specification and evaluation of quality need to support these diverse points of view. It is necessary to define these perspectives and the associated technologies for quality, in order to manage quality properly at each stage of the lifecycle. This is the *main goal* of this paper.

Internal Quality is the totality of characteristics of the software product from an internal view. Internal quality is measured and evaluated against the internal quality requirements. *External Quality* is the totality of characteristics of the software product from an external view. It is the quality when the software is executed, which is typically measured and evaluated while testing in a simulated environment with simulated data using external metrics. *Quality in Use* is the user's view of the quality of the software product when it is used in a specific environment and a specific context of use.

Every product quality category is specified by a hierarchy of *quality characteristics*. In internal and external quality category the characteristics are the same - view is different. The characteristics specify how the product is expected to fill *functionality, reliability, usability, efficiency, maintainability* and *portability compliance*. All of these categories have sub-characteristics (hierarchy of the characteristics). The quality on use is defined by four characteristics: effectiveness, productivity, safety and satisfaction.

The level to meet the expected quality is specified by metrics. Qualitative characteristics are described by *quality attributes* and measured by attribute bounded *metrics*. One characteristic may have several metrics having different scales - numeric values or classifications (like unsatisfactory / satisfactory; exceeds requirements / target / minimally acceptable / unacceptable).

As discussed above, the values of quality characters change during the software development life cycle. The *dynamics* of quality characteristics may include e.g. the *following changes*:

- New characteristics may appear during the development (*create*).
- Existing characteristics may loose their importance (*delete*).
- Existing characteristics become satisfied or are unsatisfied (*status*).
- Existing characteristics may be divided in parts (*split*).
- Existing characteristics may be joint together (*join*).
- Existing characteristics may be modified - the importance is changing, new aspects will be added, etc. (*modify*).

The changes come from the different goals of the development phases (*time dimension*) and the iterative role of software development work itself (deeper system knowledge is available in the run of the development activity).

2.2 Quality Characteristics Conceptual Model

As seen in the discussion above, product quality is a manifold concept. From the point of view of this paper the characterization of the quality classes are not important. Instead, we are pointing out the bindings and dynamics of the quality characteristics The "Quality Characteristics Conceptual Model - QCCM" (Figure 1) provides a simplified view of the most relevant concepts and their dependencies.

A *software product* consists of *work products*, that are (life cycle) *process* dependent - a certain process "owns" a certain work product, but it may be used by other processes as needed. From work product point of view the life cycle specifies the evolution path (*revisions*) of a work product from the "preliminary" version to the final one. The owner of a work product is the process finalizing it. In time dimension, even the latter processes may cause changes to the finalized work product, but in this case it is question on the improvement work that should already be done in the "owner" process.

Every work product has characteristics defined by different interest groups. These characteristics are specified by *attributes*, which are measurable and interpretable. The characteristics appear in different phases of the software life cycle as explained at the end of Chapter 1: they are dynamic by existence, value and dependencies. The characteristics may have complicated *dependencies*, which can be classified in

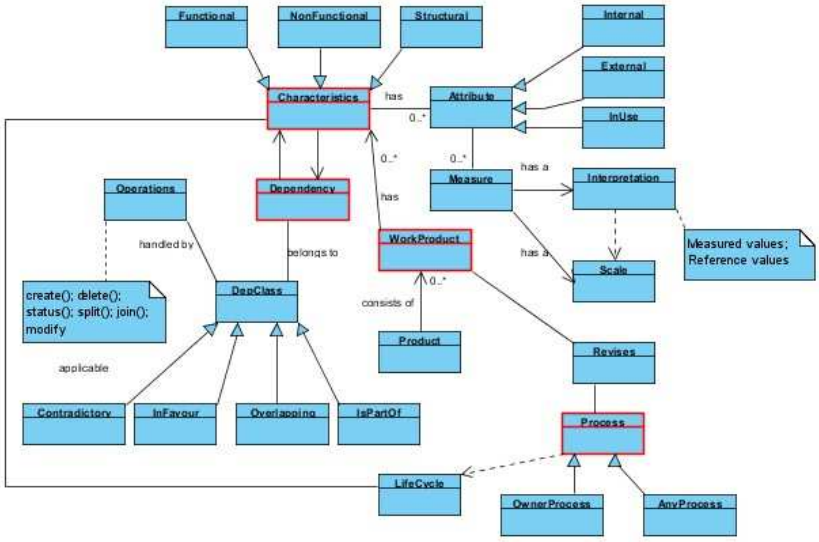

Fig. 1. Quality Characteristics Conceptual Model - QCCM

dependency classes (e.g. contradictory, overlapping, in favor) - a set of similar dependencies.

Chapter 3

From the Concept of Quality to Quality Management

The quality characteristics conceptual model in Figure 1 illustrates the dimensions of a quality model in [ISO01, ISO06d, ISO06b]. This model is not explicitly given there. We have been extracting it after surveying the entire set of documents. Conceptualisation is the first main step and the basis for quality management. In the sequel we propose a framework based on four levels. We separate the description of a quality characteristics from the procedures applied in quality management, from the application of the framework and finally from the impact of finding changes in the software.

There have already been developed a number of frameworks for treatment of data quality. FRISCO bases the framework on linguistic dimensions: syntactical, semantical, pragmatical, social, empirical, and physical view point. NATURE handles three dimensions: specification, representation and agreement. [Kro05] generalises these two approaches by adding an actor dimension. We use the SQuaRE framework, extend this framework by lessons learned in software improvement projects [MVJ99, JMV01, LMV02] and integrate the three frameworks.

We shall use the experience gained with other frameworks and develop a general framework to quality management in Figure 1. Analysing the intentions in Figure 1 and the necessities for management of quality we discover a number of dimensions for each quality property: work product dimension, life cycle dimension, dependencies among quality properties, categorisation dimension, and finally the treatment dimension which is the main topic for this Chapter.

Data quality properties must be based on a formal basis. Similar to [JBM08] we define data quality by explicit specification of the characteristics, by explicit statement of the meaning, by explicit consideration of the purpose aspect, and by providing a basis for trusting in the quality statements. We illustrate these four different aspects in the next Chapter based on one characteristics. The complete specification of all quality characteristics is still an open research question.

Quality is accepted as a multi-dimensional and hierarchical concept [BS06]. Examples of multi-dimensional characteristics are: (i) accessibility, interpretability, usefulness, believability; (ii) intrinsic, contextual, representational, accessibility; (iii)

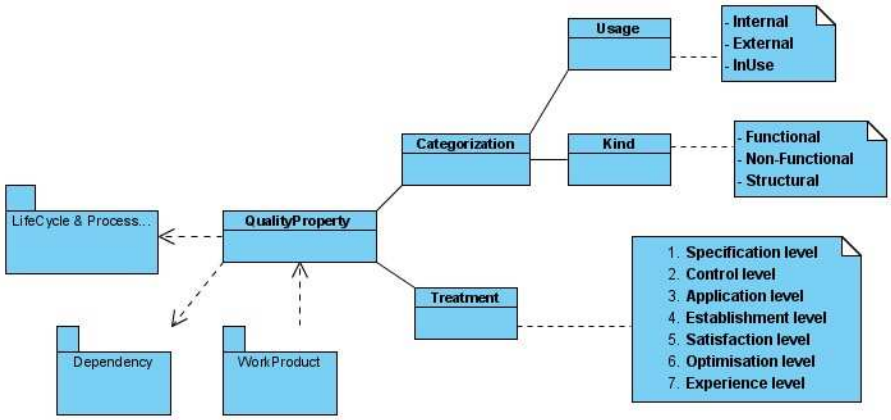

Fig. 1. Dimensions of the Quality Management Framework

mandatory versus desirable, primary versus secondary, direct versus indirect. At the same time, quality management is also difficult due to ambiguity of definitions, subjectiveness in measuring, circularity of definitions within a single classification, and inconsistency across multiple classifications. For instance, credibility is defined as a subattribute of believability and specified as having sufficient evidence to be believed. The definition of believability and completeness is inconsistent since they belong to two disjoint categories while they are related through a specialisation link. Therefore we target in a holistic treatment and management of quality.

3.1 Treatment of Quality Management By Separation into Levels

Our framework is based on a four level model:

1. The *specification level* (Section 3.2) is used for description of the quality. The description consists of a specification of the quality property, the measurement, and the policies for evaluation. It can be extended by specific policies for the various development methods such as agile development, by transformations of quality properties to others, and by associations among quality properties. Finally we may derive constraints for the application of the quality property.
2. The *control* or *technical level* (Section 3.3) treats the application of the quality model. The application of the quality framework is based on the quality property portfolio. We develop techniques and methods for applying quality checks and derive the quality evaluation plan.
3. The *application* or *technology level* (Section 3.4) handles the management of

quality evaluation within software etc. projects based on the technology of development.

4. The *establishment* or *organisational level* (Section 3.5) is based on a methodology and may be supported by a quality maintenance system.

This four-level framework for quality management can be extended by level *five* that provides facilities for handling satisfaction of quality properties and for predicting changes of satisfaction whenever software is evolving. Level *six* integrates the quality management into the optimisation of the software development process. Level *seven* uses experiences gained for innovation and adaptation of other processes and products that have not yet reached this maturity.

Quality properties can be oriented towards products of the software development or towards the processes of the software development. This distinction is reflected by a distinction into product-oriented quality properties and process-oriented quality properties. The properties of the first kind are static one, the properties of the second kind are transitional or dynamic ones. The second kind must be applied to basic or generic activities used within development processes. Most of these processes are refinement steps. Therefore, we may distinguish between faithful (or quality-preserving) refinement activities and those which must be extended by quality maintenance or support activities. This paper concentrates on the first kind of quality properties. Nevertheless, the second kind can be handled in a similar way.

3.2 The Quality Property Specification

Quality property specification may be handled in a variety of ways. We prefer the description in a semi-formal way based on templates. Due to space limitations the description is given here in an informal way and in Chapter 4 in a formal form for an example. The specification is divided into a definitions of the main attributes or parameters, the description of the quality property in an informal or formal form, the measurement component, the data component component, and the evaluation component.

The description of the quality property is based on the proposals of standards such as ISO/IEC 9126-1 [ISO01]. Our framework uses a more structured way. A quality property must be measurable. Therefore, the measurement component specifies the methods used. Measurement uses data and produces a result.

Measurement of quality properties is usually based on elementary properties called attributes. Quality characteristics are given by the above introduced template. We need now to describe the relationship between the description of the quality characteristics and its associated quality characteristics with software product attributes and the corresponding software quality measures.

Measurement functions, quality measure elements, and measurement methods are typically applied if we are able to reason on quality based on the ratio between the good or bad cases against all possible cases. We may consider a number of ratio measures. Recall evaluation relates the number of positive observations with the number of all possible observations. Fallout evaluation measures the negative observations against the number of all possible observations. Precision evaluation

typically measures the relevant observations similar to recall observations. Measurement functions are often using *metrics*. Another kind of measurement uses *model checking* functions that are based on predicates which evaluate certain properties. These properties can be used to derive whether a work product is consistent and can be refined to work products at the implementation layer.

Additionally we need an approach to provide *tolerance* of the results and deviations from the quality requirements.

3.3 Control of Quality Characteristics

The specification is divided into a restriction of the scope of control, the description of the quality control tasks, participants or controllers, the execution context, and restrictions applied to control. The description is given here in an informal way and in Chapter 4 in a formal form.

Quality control is often given on the basis of a number of quality control tasks. These tasks are general transformation steps leading from one initial situation to a targeted one. Quality control is often based on collaboration of participants. The role and the part of participants can be explicitly given or can be assumed from a number of standard participation frames. The execution leads to certain results. Control can only be enacted under starting conditions and must be terminated if closing conditions become valid. Additionally we might specify the execution context and restrictions for control.

The fitting of a quality characteristic can be either measured through a logical formula or through ordinal values. It measures the requirement, making it possible to determine whether a given solution fits the requirement. If a fit criterion cannot be found for a requirement, then the requirement is either ambiguous or poorly understood. All requirements can be measured, and all should carry a fit criterion.

Quality management is typically combined with either eager or lazy enforcement. In eager quality management each of the properties required is constantly verified whenever a change in the specification has been made that might have an impact on the property. Lazy quality management either requires the maintenance of a monitor or uses checkpoints for issuing a quality check for all quality characteristics.

3.4 Application of the Quality Characteristics

The control framework results in a *quality evaluation plan*. This evaluation plan specifies the quality check objective (errors (incorrect result), faults (consequence of a human error), failures), the quality check strategy, the quality check coverage, the quality check cases and suites, the quality check adequacy criteria, the quality check techniques, and the quality check outcomes.

This outcome is used for a development of a general evaluation procedure for any quality characteristics or property The evaluation procedure we used is based on a five step protocol displayed in Figure 2.

The purpose of the quality property drives the enactment of quality control. We determine the analysis model and the decision criteria. The data and functions needed for quality control are obtained and prepared for evaluation. The chosen

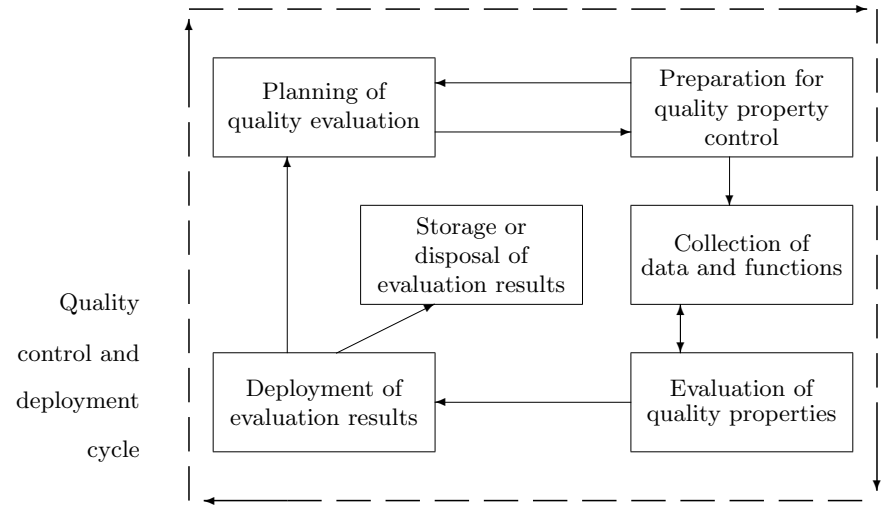

Fig. 2. The General Quality Control and Deployment Cycle of our Application Level

quality control methods are applied. Results are collected and evaluated whether the quality is completely supported, whether they allow a certain tolerance of minor deviations, and whether they fit for the current development stage.

3.5 Establishment of the Quality Characteristics

At the establishment level the framework supports a number of goals such as

- *quality-observing guidance* by coordination of assessment and improvement,
- *quality stewardship* (by alignment of problems, obligations, and satisfaction),
- *support* by clarifying the satisfaction and by analysis of problems,
- *quality delivery* by providing delivery and distribution to all units,
- *data acquisition* by handling back-end activities,
- *quality analytics* through problem tracking, optimisation and forecasting, and
- *support for contracting* by explicit display of the quality level reached.

Due to space limitations this level is going to be described in a sketchy form. The establishment level defines a quality assurance process capable of achieving a coherent quality management. It uses a standard process that will support the deployment of the defined process, includes appropriate tailoring guidelines, manages influences of development activities to change of the *quality portfolio*, defines competencies and roles of actors during development activities, identifies the required infrastructure and work environment for performing a development activity, and determines suitable methods for monitoring the effectiveness and suitability of quality characteristics.

The evaluation results are used for changing or optimising the development process, for extending the scope of quality control, for managing conflicts among the

quality requirements, and for reports of the quality control. Evaluation results support two activities: Drawing conclusions for either changing the development process or for weakening or strengthening requirements to the work products or applying conclusions that may directly lead to eager or lazy enforcement rules. Finally, quality evaluation results can be recorded or directly or later disposed. In the case of storage we keep the log of the evaluation.

Chapter 4

An Example

The framework has been tested and checked[Wal07] for three quality criteria at different abstraction layers, different levels of formal handling and at different life cycle phases: suitability, response time behaviour, and memory resource utilisation. These three quality criteria have been selected as representatives of the large variety of quality criteria proposed in the SQuaRE standard.

Response time is defined in [ISO06d] as the "time interval between user action and system response for specific user operation". In detail, execution time (or observation time) is the interval between user action and system response for specific user operation. Response time is usually a measure of an interactive system's efficiency that tracks the speed with which the system will respond to user's command.

In the sequel we use one of the possible formalisations. We notice however that several other formalisations can also be used.

Response time behaviour is associated to the work products under consideration, the atomar actions or tasks of these products, the life cycle under consideration and the dependencies with other quality properties.

We restrict the discussion to the first two levels due to space limitations.

4.1 Specification of the Response Time Behaviour Quality Characteristics

4.1.1 Definitions.

The response time behaviour is defined on top of the description of the set S of development states of the work product under consideration, the set Fl of all data for this work product, and the set Fc of functions of this work product.

Let $Tk \subseteq Fc \times 2^{Fl}$ the set of time-critical atomar actions or tasks $tsk = (fnc, \widehat{Fl})$ for $\widehat{Fl} \subseteq Fl$ that can be called by a user on the data. Let $\widehat{Tc} \subseteq Tc$.
The function $timeN : Tk \to \mathbb{N}$ defines the number of checks for the task $tsk \in Tk$.
The function $taskTime : Tk \times S \times \mathbb{N} \to \Delta Time$ defines the time necessary for the check at the current state.
The function $taskTimeMean : Tk \times S \to \Delta Time$ defines the average of check time for all states under consideration.

The function $weight : Fc \to W$ assigns a weight to functions. This weight denotes the importance of the function. This function is used for the definition of the derived function $weight : Tk \to W$ which assigns a weight to tasks $tsk = (fnc, \widehat{Fl})$ based on the equation $weight(tsk) = weight(fnc)$. We use additionally a tolerance threshold value $w_{tol} \in W$ for the weight of tasks. The value $c \in \Delta Time$ defines a constant for an acceptable minimal duration.
The value $r \in \frac{\Delta Time}{Filesize}$ declares a maximal threshold for the relative duration of function execution depending on the file size for the set $Filesize$ of the set of all files.
The function $sizeOf : 2^{Fl} \to Filesize$ provides a size value for any set of files.

4.1.2 Quality Property Formula.

The response time behaviour can now be specified in dependence of a set $\widehat{Tk}$ of tasks, of a value for tolerance w_{tol}, the constant value c, the relative duration r and the development state s by a formula

$$\begin{aligned} &ResponseTimeBehaviour(\widehat{Tk}, w_{tol}, c, r, s) = \\ &\quad \forall\, tsk = (fnc, \widehat{Fl}) \in \widehat{Tk} : \\ &\qquad weight(tsk) \geq w_{tol} \Rightarrow \\ &\qquad taskTimeMean(tsk, s) \leq c + r \cdot sizeOf(\widehat{Fl}) \end{aligned} \tag{1}$$

This property specifies that the response time behaviour is appropriate at development state s if the duration of execution is smaller than a specific value for all tasks tsk which have a high importance.

4.1.3 Measurement.

The response time behaviour measurement is based on measuring the duration of the execution of tasks. We use a number of repetitions for identical situations and the average value for these repetitions.

Let $n = timeN(tsk)$ for a given task tsk at a state s.
We use the statistical values mean and deviation for the definition of the $taskTimeMean$:

$$\begin{aligned} mean &= \frac{\sum_{i=1}^{N} taskTime(tsk, s, i)}{N} \\ deviation &= \sqrt{\frac{1}{N-1} \sum_{i01}^{N} (taskTime(tsk, s, i) - mean)^2} \\ maxVal(tsk, s) &= max_{i=0}^{N}(taskTime(tsk, s, u)) \end{aligned} \tag{2}$$

$$\begin{aligned} &\text{do for all } \widehat{s} \in S \\ &\quad \text{if } getTaskState(tsk, s) = getTaskState(tsk, \widehat{s}) \text{ then} \\ &\qquad taskTimeMean(tsk, s) := min(mean + deviation, maxVal) \end{aligned} \tag{3}$$

where $getTaskState : Tk \times S \to S$ reduces the state $s \in S$ to those situations in which the function $fnc \in Fc$ is used for a task $(fnc, \widehat{Fl})$.

The values r and w_{tol} are used for the calibration of the measurement. The first value measure the relation between duration and file size. The second one specifies the importance of functions that are relevant for the result.

4.1.4 Data.

The data part of the specification defines the functions used, the set $\widehat{Tk}$ of tasks, the constants, and the value assignments where the later can also use less detailed granularity. The data part also clarifies the produced content, the scale and the form of the presentation of the measurements. For instance, values from $\{.1\frac{sec}{MB}, .5\frac{sec}{MB} 1\frac{sec}{MB}\}$ have been used in [Wal07] for r. Other constants used in the formulas above are used for configuring the measurement framework. The function $timeN(tsk)$ may either be dynamically configured depending on the state under consideration or be statically assigned. The set $\widehat{Tk}$ is set in dependence on the stage of the development.

The results may use scales for level of tolerance and level of priority. The measurement phase produces content data such as $taskTime$, $taskTimeMean$ and $failedTasks$.

4.1.5 Associations with other Quality Characteristics.

Response time behaviour is a sub-characteristics of time behaviour and does not have other sub-characteristics. This quality characteristics is interrelated with suitability and many other quality characteristics. It also conflicts with suitability, accuracy, security, adaptability, memory resource utilisation, etc. At the same time it supports productivity, attractiveness, satisfaction etc. Therefore, we get a *space of gains and tradeoffs*. This space may be used for optimising quality criteria satisfaction and explicit discussion of the choices made during software development.

4.1.6 Evaluation of Quality Characteristics.

We use for the evaluation a distinction into products which quality is satisfying, into products which quality is not satisfying, and orthogonally into products under consideration and products that are not of interest. To make this distinction explicit we use precision, recall and fallout values similarly to approaches used in information retrieval.

$$Precision(\widehat{Tk}, s) = \frac{|fullfillingSet(\widehat{Tk})|}{|\widehat{Tk}|} \quad \texttt{where}\ fullfillingSet(\widehat{Tk}) = \{task | tsk \in \widehat{Tk} \wedge tsk \notin FailedTasks_{s,r}\} \tag{4}$$

$$Recall(\widehat{Tk}) = \frac{|\widehat{Fc}|}{|Fc|} \qquad \texttt{where}\ \widehat{Fc} = \{fnc | \exists \widehat{Fl} : (fnc, \widehat{Fl}) \in \widehat{Tk}\} \tag{5}$$

$$Fallout(\widehat{Tk}, s) = \frac{|unsatisfied(\widehat{Tk})|}{|\widehat{Tk}|} \quad \text{where}\ unsatisfied(\widehat{Tk}) = \{tsk | tsk \in \widehat{Tk} \wedge tsk \in FailedTasks_{s,r}\} \tag{6}$$

and where $FailedTasks_{s,r}$ denotes the set of tasks that do not satisfy the property.

4.2 Control of the Response Time Behaviour Quality Characteristics

4.2.1 Setting of Control Management.

Control management is based on the *profile* of the computational environment and the *task portfolio* of the management. The profile of the computational environment can be specified in a form similar to the setting of database benchmarks [TPC08]. The profile uses the specification of the computer, the computational profile of the quality assessment software, and the concurrent computational profile as mandatory parameters.

4.2.2 Scope of Control.

Scope of control is based on a selection of the main parameters for the specification. We might restrict our attention to crucial functions or to functions that require high bandwidth due to the data exchange. The scope of control is often restricted by the hardware and software within the application. Control tasks may also be restricted by aspects such as structuring, functionality, interactivity or distribution. The scope for the given quality criterion is based on functionality, interactivity and distribution. Scope of control is based on explicit selection of $\widehat{Tk}, w_{tol}$, and r.

The result of the control is a new set of assignments for the function $taskTime$. Additionally, the function $failedTasks : S \times \frac{\Delta Time}{Filesize} \to 2^{Tk}$ allows to judge which actions do not satisfy the quality criterion in dependence on r and s. We use an auxiliary module for the computation of failing parts:

$$\begin{aligned} &GetFailedTasks(\widehat{Tk}, w_{tol}, c, r, s) = \\ &\quad \forall\, tsk = (fnc, \widehat{Fl}) \in \widehat{Tk} : \\ &\qquad \texttt{if } taskTimeMean(tsk, s) \leq c + r \cdot sizeOf(\widehat{Fl}) \texttt{ then} \\ &\qquad \texttt{add } (fnc, \widehat{Fl}) \texttt{ to } failedTasks(s, r) \end{aligned} \tag{7}$$

We additionally use rely-conditions and guarantee-conditions by both pre- and post-conditions. Rely conditions state what can be tolerated by the users. Guarantee-conditions record the interference that other quality characteristics will have to put up with. We envision in this paper that these conditions can be generalized to a specific style of *assumption-commitment specification*.

4.2.3 Quality Control Tasks.

The quality criterion in this example is based only on one control task. Typically control tasks are scheduled. We assign to each control task $ctlTsk$ an event $ev(ctlTsk)$, a condition $cond(ctlTsk)$ and an control action $act(ctlTsk)$ and require that if the event is observed and the control condition is valid then the control action is fired. The event or the control condition may be trivial. Typical events are timestamps.

Let s be the actual development state of the program and $\widehat{Tk} \subseteq Tk$. We assume that the program function $fnc \in Fc$ is in s implemented and testable if an action $(fnc, \widehat{Fl}) \in \widehat{Tk}$ exists. We may assume that events are controlled by a time function

$getTime : Tk \rightarrow \Delta Time$. We also may simplify $cond(ctlTsk)$ to a dynamic marking function $marked : Tk \rightarrow Bool$. Actions can be based on counting the the time for execution of a function or for measurement of tasks.

The task model we use is based on an explicit specification of the initial and target states, starting situations for initiating the control, and closing conditions for finalising the control. The starting situation may either be based on automatic check of validity of control conditions and of observations of events or may use a manual start.

$$StartingSituation(\widehat{Tk}, s) \quad :\Leftrightarrow \quad manualStart \tag{8}$$

The initial state of a control task consists of all actions for which values are existing.

$$\begin{aligned} PrepareInitialState(\widehat{Tk}, s) = \ & \texttt{if } StartingSituation(\widehat{Tk}, s) \texttt{ do} \\ & \forall\, tsk \in \widehat{Tk} \text{ with } marked(tsk) = FALSE \text{ do} \\ & \quad \texttt{if } taskTimeMean(tsk, s) = UNDEF \texttt{ then } marked(tsk) := TRUE \end{aligned} \tag{9}$$

The target state does not have any marked actions.

$$\begin{aligned} & TargetState(\widehat{Tk}, s) \Leftrightarrow \\ & \quad \forall\, tsk \in \widehat{Tk} \ : \\ & \qquad taskTimeMean(tsk, s) \neq UNDEF \ \wedge \ marked(tsk) = FALSE \end{aligned} \tag{10}$$

4.2.4 Participants.

Participants of the quality control are developers, implementers and users.

4.2.5 Execution Context and Restrictions.

An execution context and/or a set of restrictions for this quality criterion is considered in dependence on the profile of the computational environment. Typical restrictions are given by the memory and throughput of the computational environment.

4.3 Application of the Response Time Behaviour Quality Characteristics

The application level of quality characteristics has been briefly discussed in Section 3.4. The quality control and deployment cycle given in Figure 2 has been applied to the given quality criterion by explicit definition of the quality maintenance plan, by explicit description of checking modes and levels, by plans for scope conditions and control parameters, and by matching conditions depending on the development state.

We distinguish between quality characteristics that are affected by a development action and that remain to be stable. Negatively affected characteristics require a specific treatment. Five different reactions may be initiated: Either the action is going to be revised until the characteristics is becoming fulfilled or another set of actions are initiated for an explicit repair or revision of the characteristics or the quality characteristics is going to be revised or the quality control is delayed until other actions have been completed.

For automatic support of quality control a number of triggers have been developed for the response time quality characteristics. These triggers automatically track the changes in the function set Fc or in the implementations of the functions.

4.4 Establishment of the Response Time Behaviour Quality Characteristics

Section 3.5 has defined the general program for establishment. The quality characteristics considered in this Chapter is supported by functions such as $GetFailedTasks$ and $failedTasks$ for a development state s and for a threshold value r. We use the triggers sketched in the previous Section for automatic display of of development problems and for enabling one of the possible reactions. If a revision is requested then the completion of the action is blocked until the quality characteristics is fulfilled. If the quality characteristics is revised on the basis of changes in the constants then the effect of these changes is shown before the characteristics is going to be changed.

The quality characteristics response time behaviour has a small set of cause that might influence negative change during a development action. We can revise the action, change the level of tolerance by increase of the constants or of the ratio threshold, increase the random test sample for an improvement of the deviation or improve the hardware or software for a better response.

These potential changes can be monitored and thus discussed with the developer. Some of the changes are crucial. Therefore, we plan to develop an advisory system that allow to show the correspondence of the causes and of the improvement actions.

Chapter 5

Application of the Framework in Practice

The framework is currently applied and tested in a software re-engineering project for a large German company. The software under review, assessment and improvement supports the complete logistics of the the railway system in Germany and of public transportation in some regions in Germany. The proof of practicality has led to the requirement of quality tracking and continuous quality improvement. Our framework has proven to be sufficient to support this requirement.

The three quality characteristics have not been randomly chosen but were the main quality properties that have been requested by the contract of this German company with the transport corporation. The quality improvement is currently stated on request. It can however be automatised and combined with the testing framework. The company is currently extending this framework to all its major software products.

Chapter 6

Conclusion

We presented a formal framework to quality management that allows to handle any quality property at the specification and at the control level. In a similar form, the application and the establishment level are going to be handled. The framework can be extended by a satisfaction level, an optimisation level, and by an experience level. The framework has been used in a large project that was aiming in quality management for three quality characteristics that have been shown to be crucial for the project.

The formal framework has been illustrated for one of these quality characteristics: response time behaviour. This characteristics can be formally defined and integrated into continuous software engineering.

A large variety of quality characteristics has been standardised. The SQuaRE standard [ISO06b] distinguishes between internal quality, external quality and quality in use. These three usage categories must be combined with the explicit specification of the application area, by an integration into the lifecycle models of software development, and finally by an explicit treatment of conflicts among quality characteristics.

Acknowledgement. The authors are very thankful to Marc Walker for the application of the framework in a real application and for his evaluation of the framework.

Bibliography

[BCK03] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 2003.

[BS06] C. Batini and M. Scannapieco. *Data quality: Concepts, methodologies, and techniques*. Springer, Berlin, 2006.

[CMM02a] CMMI. Capability maturity model integration, version 1.1. CMMI for systems engineering, software engineering, integrated product and process development, and supplier sourcing. staged representation. CMU/SEI-2002-TR-012, March 2002.

[CMM02b] CMMI. CMMI for systems engineering/software engiineering, Staged representation. CMU/SEI-2002-TR-002, ESC-TR-2002-002, Version 1.1, 2002.

[ISO01] ISO/IEC. 9126-1 (Software engineering - product quality - part 1: Quality model). ISO/IEC JTC1/SC7 N2519, 2001.

[ISO03] ISO/IEC. Software engineering - Guidelines for the application of iso 9001:2000 to computer software. TC JTC1/SC7, ISO/IEC FDIS 9000-3:2003, 2003.

[ISO05] ISO/IEC. Quality management systems - Fundamentals and vocabulary. IS 9000 (2005), 2005.

[ISO06a] ISO/IEC. Information technology - process assesment. parts 1-5. IS 15504, 2003-2006.

[ISO06b] ISO/IEC. 25010 (Software engineering . software product quality requirements and evaluation (SQuaRE)quality model). ISO/IEC JTC1/SC7 N3426, 2006.

[ISO06c] ISO/IEC. Software engineering - Software product quality requirements and evaluation (SQuaRE) - Quality model. ISO/IEC WD 25010, 2006.

[ISO06d] ISO/IEC. WG6 SQuaRE project status and report for SC7 Bangkok plenary meeting. ISO/IEC JTC1/SC7 N3520, 2006.

[JBM08] L. Jiang, A. Borgida, and J. Mylopoulos. Towards a compositional semantic account of data quality attributes. In *Proc. ER 2008*, volume 5231 of *LNCS*, pages 55–68, Berlin, 2008. Springer.

[JMV01] H. Jaakkola, T. Mäkinen, and T. Varkoi. Assessment of a software process assessment process. In *PICMET 01*, Portland, Oregon, July 2001.

[KLS95] J. Krogstie, O. I. Lindland, and G. Sindre. Towards a deeper understanding of quality in requirements engineering. In J. Iivari, K. Lyytinen, and M. Rossi, editors, *CAiSE*, volume 932 of *Lecture Notes in Computer Science*, pages 82–95. Springer, 1995.

[Kro05] J. Krogstie. Quality of UML. In Mehdi Khosrow-Pour, editor, *Encyclopedia of Information Science and Technology (IV)*, pages 2387–2391. Idea Group, 2005.

[LMV02] M. Lepasaar, T. Mäkinen, and T. Varkoi. Structural comparison of SPICE and continuous CMMI. In Fabbrini F. Fusani M. & Rout T. Dorling, A., editor, *SPICE 2002*, Venice, Italy, 2002.

[MVJ99] T. Mäkinen, T. Varkoi, and H. Jaakkola. Database implementation for a software process assessment model. In D.F. Kocaoglu and T.r. anderson, editors, *PICMET'99, Portland International Conference on Management of Engineering and Technology*, Portland, Oregon, USA, July 25-29, 1999.

[PGC02] M. Piattini, M. Genero, and C. Calero. Data model metrics. In *Handbook of Software Engineering and Knowledge Engineering*, volume II, pages 215–229. World Scientific Pub. Co., 2002.

[TPC08] TPC. TPC benchmarks C, D, H/R, W. http://www.tpc.org (Transaction Processing Performance Council), 2008.

[Wal07] M. Walker. Ein Framework zum durchgehenden Qualitätsmanagement. Master's thesis, Christian-Albrechts-University at Kiel, Computer Science Department, 2007. (In German).