Software Qual J (2010) 18:469–490
DOI 10.1007/s11219-010-9101-z

Quality specification and metrication, results from a case-study in a mission-critical software domain

Jos J. M. Trienekens · Rob J. Kusters · Dennis C. Brussel

Published online: 16 June 2010

Abstract Software quality is of increasing importance in mission-critical embedded software systems. Due to the fast growing complexity and accompanying risks of failures of these systems, software quality needs to be addressed explicitly by software developers, preferably with a systematic method for an optimal implementation of software qualities, such as reliability, time-behavior and usability. At the Centre of Automation of Mission-critical Systems (CAMS) of the Dutch Royal Navy, a new approach has been defined for software developers to improve the way that they deal with software quality in the process of mission-critical systems engineering. The stepwise approach is based on both an international quality standard for software product quality, i.e. ISO9126, and on Multi-Criteria Decision Making techniques, i.e. analytical hierarchy process (AHP). The stepwise approach has been validated in a case study. In particular, the tailoring of the ISO9126 standard toward the specific CAMS development situation, and the applicability of AHP techniques, from the perspective of software developers, has been investigated. The case study is carried out in a representative software development project, i.e. the software for combat management systems (CMS) of warships. Results of the case study show that software developers can explicitly deal with quality on the basis of both the ISO9126 standard and the AHP techniques, respectively regarding the specification, prioritization and metrication of software product quality.

Keywords Software quality · Specification · ISO 9126 · AHP · Prioritization · Metrication

J. J. M. Trienekens (✉) · R. J. Kusters
University of Technology Eindhoven, 2 5600 MB Eindhoven, Den Dolech, The Netherlands
e-mail: j.j.m.trienekens@tue.nl

R. J. Kusters
e-mail: r.j.kusters@tue.nl

D. C. Brussel
Centre for Automation of Mission-Critical Systems (CAMS), Royal Navy, De Ruyghweg 200, Den Helder, The Netherlands

1 Introduction

Air-defense and command frigates (Dutch acronym LCF) of the Royal Dutch Navy are warships that are used in national and international task forces to support political decisions. An important aspect of the power of these frigates is the so-called guided missile. The heart of the command function of such warships is supported by a combat management system (CMS). These highly advanced mission-critical software systems integrate sensor and missile applications and form the central operating systems of the frigates. More than 20 operators on a warship are working in the kernel of the command function on powerful working stations to identify air attacks and to determine defense actions. The information that is needed to identify the foreign objects is provided by different types of sensors, e.g. from radar systems. The various signals are being collected by the sensors and are tracked and analyzed by the kernel software functionality of the CMS, called multi sensor tracking (MST). Both the development and the operational usage of the software require specialist expertise and advanced skills of software developers and end-users of the software.

The development of the mission-critical systems is carried out by the Centre of Automation of Mission Critical Systems (CAMS). CAMS is the knowledge and development centre of the Dutch Royal Navy.

The mission-critical systems are characterized by extreme complexity, and a need for a high level of real-time performance and reliability. End-users, i.e. the operational users of the CMS, are collaborating closely with software developers in accordance with the CAMS software development process model. At CAMS, software development has gone through many changes over the last years. The management at CAMS has been challenged with various opportunities for improving the development of software applications. Advanced analysis and design methods and tools have been introduced such as rapid application development (RAD), e.g. to improve the efficiency of the development processes. RAD has replaced the classic waterfall model approach (Royce 1970). To reduce the complexity of the software applications, incremental development techniques have been introduced by the quality assurance staff in collaboration with the software developers and in accordance with DSDM. Software is developed in well-delineated increments. In early stages of a development project, increments with kernel functionalities are defined, which are verified and validated in each phase of a project. In-line with incremental development, component reuse projects have been initiated. Regarding methods and techniques to be used, an organization-wide standardization program has been initiated on the higher management level, and is being executed by the quality assurance staff in collaboration with the software developers. The objective of standardization is the improvement of the consistency and the transparency in the software development processes and their products. Another challenge that has been identified by the software developers in collaboration with the quality assurance staff is the improvement of technical decision-making in the development processes, e.g. regarding optimization of functionalities and non-functionalities. Several multi-criteria decision-making (MCDM) techniques have been investigated.

Regarding decision making, different groups of stakeholders have to be recognized. From the perspective of the software developers, at CAMS, two groups of stake-holders can be distinguished, respectively, the end-users of the software and the quality assurance staff. The end-users are the operational experts, i.e. military specialists, in the usage of CMS; the quality assurance staff offers support regarding the selection and application of (international) standards and methods, techniques and tools. The software developers are

classified, in accordance with their expertise, in different software development domains, e.g. requirements engineering, analysis, design, implementation; and project management. The input of the end-users in the system development process is recognized, by both software developers and quality assurance staff, as a critical success factor for the successful development of the software.

The paper is structured as follows. In Sect. 2, the problem situation of the case study will be discussed. Section 3 addresses related research on software quality specification, prioritization and metrication. Section 4 presents the systematic software quality approach for the CAMS software developers. In Sect. 5, the quality approach is applied and validated in a realistic case study. Subsequently the lessons learned by the software developers are presented, respectively regarding the specification of software quality characteristics, the determination of their relative importance, and the metrication of software quality. Section 6 finalizes the paper with conclusions.

2 Problem situation

At CAMS, the management has serious doubts regarding the quality of the software applications of the frigates. This is in particular based on the experiences of CMS end-users. Although satisfied with the functionalities of the software, they complain about weak quality aspects, such as the sometimes unpredictable time-behavior and the restricted reliability of the systems. The software developers, being confronted with the complaints, confirmed that they are not addressing, in a systematic way, the specification and implementation of the so-called software qualities, such as reliability, usability, and time-behavior. Implementation and optimization of software quality is carried out in an ad-hoc way, on the basis of the expertise of individual software developers. No specific methods, techniques and tools are used to address software quality in an explicit way. Current methods in use such as RAD (Royce 1970) and DSDM only provide very restricted support. Recently, the CAMS management has stressed the need for a research and improvement project in a realistic case study. Representative stakeholders had to be involved in the research project. The main goal of the research project has initially been defined as: the development of a systematic approach for software developers for the specification and implementation of software product quality. Important sub goals are to derive best practices or lessons learned regarding the optimization, i.e. prioritization, of the different software qualities and the quantification, i.e. the metrication, of software qualities.

This paper focuses on two problem areas of software development at CAMS. The first problem area is the high degree of subjectivity in software quality specification In particular, the operational domain experts, i.e. the end-users, have difficulties in expressing their needs and wishes regarding software quality. The developers lack methods and techniques to respectively interpret and elaborate the (often vague) needs and wishes of the end-users. To solve this type of problem, the benefits of applying software quality standards has already been reported a number of times in literature, see e.g. (Franch and Carvallo 2002; Botella et al. 2003, 2004; Behkamal et al. 2009). To strive at solutions at CAMS the first research subject has been defined as:

> Investigate the applicability of a software product conceptual framework and terminology, i.e. a quality standard, in a realistic case study, to improve the identification of end-users' quality requirements and the specification of software quality characteristics.

The second problem area is that only after the implementation of the software in the command system on board of a frigate, it becomes clear that the 'offered' quality is much less than the expected and needed quality. This is caused by the large extent of subjectivity and unstructuredness in the development process regarding the specification and implementation of software quality characteristics. In particular, the decision-making processes with respect to trade-offs between the different software qualities (such as reliability, usability and time-behavior), is based on ad-hoc activities of individual software developers. To solve this type of problem, the benefits of MCDM techniques have already been addressed several times in the software industry, see e.g. (Karlsson et al. 1998, 2004), Zhu et al. 2005). To strive at a solution for the problems at CAMS the second research subject has been defined as:

> Select and apply a MCDM technique in a real-life case study and investigate its applicability for software developers regarding decision-making in complex trade-off development situations and metrication of software product qualities.

3 Related research

In order to analyze the two mentioned problem areas, related research has been investigated. Regarding the identification and specification of software product quality, different quality models and terminologies can be distinguished, e.g. (Boehm 1978; Musa, 1993; Maiden and Rugg 1996; Kitchenham et al. 1997). On the basis of this work, a software product quality standard has emerged (ISO/IEC 9126 2001). This standard is currently one of the most widespread quality standards. The standard defines quality as 'the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs' (ISO/IEC 9126 2001). It offers a valuable conceptual framework for software quality, a set of well-elaborated definitions for software quality characteristics, and metrics regarding the quantification of quality attributes.

The standard has been applied in different domains and has served different purposes. In particular, the development of ISO9126-based quality models for specific software domains has been addressed over the last 10 years. Examples are the development of ISO9126-based quality models for respectively mail servers (Franch and Carvallo 2002), ERP packages (Botella et al. 2003), COTS selection (Botella et al. 2004), and B2B applications (Behkamal et al. 2009). These authors stress on the one hand the importance of an internationally accepted generic framework for software quality to avoid confusion and subjectivity, but on the other hand also the need for tailoring the generic ISO9126 quality model to specific software domains. Tailoring focuses on the hierarchical structure of the quality model, the definitions of the quality characteristics and attributes, and the metrication of these attributes.

At CAMS, it was decided to investigate the applicability of the ISO9126 standard. In the following, the main aspects of this standard and the opportunities for tailoring it to a specific software domain will be addressed.

ISO9126 makes in its conceptual framework a distinction between 'quality in use', 'external' and 'internal quality', see Fig. 1. 'Internal quality' is expressed in terms of quality attributes and reflects the static aspects, e.g. code structure and programming language, of a software application.

'External quality' characteristics reflect the dynamic aspects of a software application. 'Quality in use' is the user's view of the quality of a software application, and is expressed

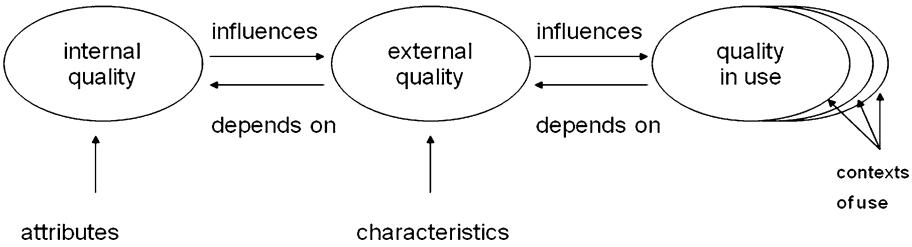


Fig. 1 Quality concepts (ISO9126)

in terms of the results of using the software, rather than properties of the software application itself. Four 'quality in use' characteristics are defined, i.e. 'effectiveness', 'productivity', 'safety' and 'satisfaction', see Fig. 2.

Although definitions are given in the standard, these four characteristics are not elaborated further toward sub characteristics and attributes, i.e. divided into quantifiable elements. Also the interrelations between the 'quality in use' characteristics and the 'external quality' characteristics are not elaborated. 'External quality' is subdivided into six quality characteristics which are reflecting the behavior of the software when executed, see Fig. 3. 'Internal quality' is in Fig. 3 reflected by a subdivision of the six external quality characteristics into internal quality attributes. This means that the interrelations between 'external' and 'internal quality' have been elaborated explicitly. Regarding the quantifiability of the 'internal quality' attributes, metrics and measurement scales are presented in the ISO9126 standard.

Together with the clear hierarchical structure of the framework, the ISO 9126 standard (part 1) offers also definitions of the software qualities, in terms of quality (sub) characteristics and attributes. However, these definitions are given in a generic sense, i.e. they are

Fig. 2 'Quality in use' (ISO9126)

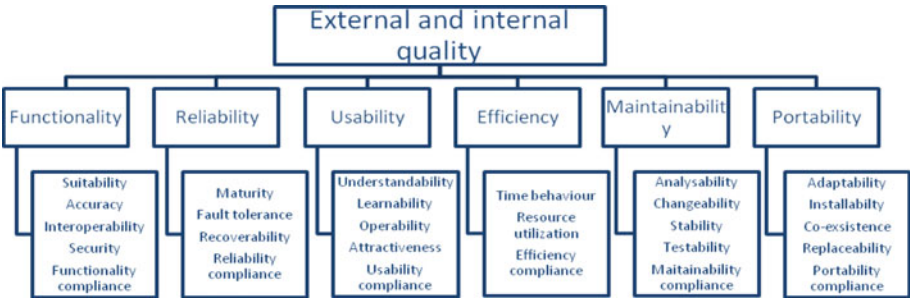


Fig. 3 'External' and 'internal quality (ISO9126)

defined independently, both from an application domain, see e.g. (Franch and Carvallo, 2002; Botella et al. 2003; Botella et al. 2004), and from purposes, such as specification, evaluation or assessment, see (Behkamal et al. 2009). The mentioned authors in the foregoing state that tailoring the generic ISO9126 standard toward a specific quality model can consist of respectively: a redefinition or refinement of definitions of quality characteristics, a splitting up of quality characteristics into two or more characteristics, and the definition of additionally needed quality (sub) characteristics and attributes. Not only the quality characteristics and attributes should be defined but also metrics have to be defined and selected (Franch and Carvallo 2002). The latter authors recommend strongly the use of mathematical concepts for describing precisely the meaning of a metric, and recommend that besides ISO9126 (part 2) the general theory of metrics should be followed, e.g. (Zuse 1998), (Fenton and Pfleeger 1998).

Regarding the process of quality specification, the ISO 9126 standard offers only limited guidance. This means that software developers have to develop themselves stepwise procedures, decision criteria and prioritization mechanisms, see e.g. (Trienekens and Kusters 1999). The different views on software quality and the many quality (sub) characteristics and attributes lead to the necessity of making trade-offs between the different software qualities (Wohlin and Mattson 2005). These complex software development tasks need to be supported by good methods, techniques, and tools, in order to avoid subjectivity and random decision making. E.g. in (Karlsson et al. 1998) it is stated that prioritization of competing quality characteristics and attributes must be elicited by software developers. The goal is to determine priorities in a process of negotiation, which should lead to consensus about an optimal solution. Regarding complex prioritization problems, MCDM techniques can provide formal and quantitative support (Roy, 1996; Svahnberg et al. 2002). The objective of MCDM is to provide practitioners with a rationale for the ordering and rating (i.e. prioritization) of different aspects of a product by using different types of criteria. Analytical hierarchy process (AHP) is an MCDM technique that offers support to reduce the complexity of a prioritization problem and to determine the relative importance of particular aspects of a system (Saaty 2001). Over the years, AHP has been applied and validated in a variety of application domains, both in general business domains, such as BPR decision-making (Mansar et al. 2009), the prioritization of infrastructural projects (Karydas and Gifun 2006), and also in the software application domain, e.g. in an evaluation of methods for prioritizing software requirements (Karlsson et al. 1998), and an experiment on the comparison of prioritization techniques (Karlsson et al. 2004). In these publications, AHP is mentioned as the most promising and trustworthy approach for establishing priorities in software development, although AHP is time-consuming. Applying AHP in a standard manner can provide priority weights of design alternatives, e.g. reflecting the relative importance of quality attributes to be implemented. AHP offers developers the opportunity to clarify and motivate their decisions in these complex prioritization processes. AHP offers developers a hierarchical method of pairwise comparisons of each quality (sub) characteristic or attribute against one another with respect to their relative importance regarding the contribution that they offer to a higher level quality (sub) characteristic. Preferences among quality (sub) characteristics and attributes are converted into numerical weights; see e.g. (Weil and Apostolakis 2001). After having determined the relative importance of the software quality characteristics and attributes, it has to be decided to what extent each quality attribute has to be implemented in a software application, and in what way. Therefore, measurement scales have to be defined for each quality attribute. These scales contain different categories of design decisions which are relevant to the implementation of a particular quality attribute. Different measurement

scales can be distinguished, e.g. natural (e.g. time, distance) and constructed scales. An advantage of constructed scales can be that no exact numbers have to be determined. For each category, a good description can be given. For further advantages of constructed scales, see (Weil and Apostolakis 2001). For each category on a scale, a contribution factor has to be determined that reflects the contribution of a particular design decision (of that category) to a particular attribute (with a certain weight). Finally, a so-called performance index is calculated as the summation of the weights of attributes multiplied by their contribution factors. The higher the performance index of a quality (sub) characteristic, the higher its contribution to the overall quality level of a software application; see (Karydas and Gifun 2006). The calculation of performance indexes will be addressed in more detail in the presentation of the case study results in Sect. 5.

In the next section, the theoretical basis offered by the ISO9126 standard, i.e. the hierarchical quality model and definitions, and the AHP techniques, i.e. regarding prioritization, have been incorporated into an approach for the stepwise quality specification, prioritization and metrication at CAMS.

4 Software quality specification and metrication at CAMS: the approach

One of the most important projects at CAMS is the project on the development of the MST functionality of the CMS. This functionality integrates the information that is received from the different sensors on board of the frigate. This information is transformed into so-called 'operational representations'. These representations reflect the kinematical aspects (i.e. the position, velocity and direction) of foreign objects, for instance enemy airplanes. These foreign objects are called 'tracks' and are detected with different types of radars at specific points in time. The quality of the information of the operational representations is of utmost importance for the decision processes at the command level of the frigate, in particular with regard to the activation of the defense system. Of course both the (operational) end-users and the decision makers need to have confidence regarding the quality of the operational representations, e.g. with regard to their reliability, time-behavior, and usability.

4.1 The stepwise software quality approach

The approach for quality specification has to be integrated into the way of working of the software developers at CAMS, which is based on the incremental development approach of DSDM. The case study focuses at a particular phase of DSDM, i.e. the RA phase. This phase has to result in a structured and consistent specification of software increments. Regarding the verification and validation of the functionality of each increment, different types of specialists, both from the development domain and from the end-user, or operational domain, are involved. Increments are developed in a so-called time boxes. A time box covers a period in time between 2 and 6 weeks in that well defined objectives have to be reached. In each time box, continuous iterative processes take place to identify, specify and prioritize the requirements, i.e. regarding the functionalities of the software application. The RA phase consists at CAMS of two main sub phases, respectively Joint Requirements Definition (JRD; striving at a complete set of requirements), and Joint Requirement Planning (JRP; defining increments and prioritizing requirements).

To improve the way software product quality is dealt with at CAMS, it was decided to develop a stepwise approach that would fit in the two RA sub phases of DSDM at CAMS. In the first RA sub phase JRD, the ISO9126 standard will be used as a basis of reference for the specification of software quality. The objective is to investigate the applicability of the ISO9126 standard to software development at CAMS, in particular the tailoring of the quality model toward the specific software development situation. In the second sub phase JRP, the objective is to find out how AHP techniques can be applied at CAMS regarding the prioritization and the metrication of software quality attributes.

To validate the approach at CAMS, a number of steps have been identified for each of the RA sub phases of software development. These steps are derived from the hierarchical structure of the ISO9126 standard, i.e. the distinction between 'quality in use', 'external' and 'internal' quality, and from the steps offered by AHP. The two RA sub phases, i.e. JRD and JRP, will be described in the following and subsequently some resource and organizational aspects of the two sub phases will be addressed.

4.2 RA sub phase 1: JRD and the specification of software product quality

In this first sub phase, the applicability of the ISO9126 quality model and in particular its different views on quality (i.e. the 'quality in use', the 'external quality' and the 'internal quality' view) are investigated. As input to this sub phase, a number of internal documents of the particular software system is used, i.e. the Combat System Specification (LCF), and different types of development and quality assurance documents, such as review and testing reports. Two steps are recognized; respectively step 1a on the specification of quality characteristics, making use of the hierarchical conceptual framework of ISO 9126, and step 1b on the refinement of the specification, following the structured subdivision of the 'overall' software product quality into (sub) characteristics and software attributes. In this sub phase, the focus is in particular on the tailoring of the ISO9126 quality model toward the specific CAMS situation, as needed by both the software developers and the end-users.

4.3 RA sub phase 2: JRP and the prioritization and metrication of software quality

In this second sub phase, the relative importance of the quality (sub) characteristics and attributes will be determined on the basis of AHP. Both the software developers and the end-users will play a role in this process. Three steps are recognized. In step 2a, the relative importance, i.e. the weights of the quality (sub) characteristics and attributes are determined. Step 2b focuses on the development of measurement scales, and step 2c addresses the calculation of performance indexes (PI) of the quality attributes. These measurement scales and PI's are needed to determine to what extent a quality attribute will contribute to the overall quality of a software application.

In the three steps of this sub phase consistency checks will also be carried out regarding the determination of the weights, the development of the measurement scales, and the calculation of the performance indexes, see (Karlsson et al. 1998; Weil and Apostolakis 2001).

4.4 Resource and organizational aspects of the software quality approach

In the stepwise approach, different domain experts are collaborating in the complex processes of specification, prioritization and metrication. In this case study, two domain

experts on the incremental development of software applications have been selected, and two domain experts on the operational aspects, i.e. the usage of the software in practice. For each step at least two workshop sessions have been organized. These sessions are coordinated and guided by a chairman/mediator. This chairman/mediator selects and collects the documents from different sources to be used during the sessions, such as quality assurance manuals, but also review reports on software quality from relevant actual and past projects. As chairman of the sessions, the mediator structures the discussions, e.g. by making use of Delphi principles and techniques to reach consensus on solutions of the various multi-criteria decision problems (Linstone and Turoff 2002). In the first workshop session, after that the objectives have been presented, the applicability of the ISO9126 framework and terminology is discussed. In order to focus the discussions, short questionnaires are used to address subjects such as the main software application characteristics, the particular needs for software quality, the delineation of the increments, the relevant context aspects of the software application, and the availability of information on software quality from previous software projects.

In the second sub phase JRD, the AHP techniques have been applied to support multi-criteria decision making on software quality prioritization and metrication. Also in this phase, in an introduction session, AHP is first presented and its applicability is discussed with the involved domain experts. In this way, confidence in AHP and its applicability has to be established. To ease the adoption of AHP and to support its applicability, it was decided to make use of a free trial version of the tool Expert Choice which supports various AHP techniques. Expert Choice is a commercially available AHP application (Expert Choice® 2000, Expert Choice, Inc.). The following section presents the results of the application of the stepwise approach in a case study at CAMS.

5 Quality specification and prioritization: a case study

The case study has been executed on a particular part of the MST functionality of the software of the CMS of a frigate. To limit complexity in the case study, a selection has been made of two increments. These two increments cover the kernel functionality of the software, respectively the functionality '*localizing contacts* and the data objects '*set of contacts*' and '*location of contact data*'. This kernel functionality stores and transforms data collected by the sensors of the radar systems of a frigate. The functionality 'localizing contacts' covers the identification of foreign objects, e.g. airplanes and their kinematical aspects (i.e. velocity, direction, and location), and the subsequent construction and presentation of the operational representations of the movements of the foreign objects. The information from the operational representations serves as a basis for the determination of particular defense actions of the frigate.

5.1 RA sub phase 1: JRD and the specification of software quality characteristics

In this first sub phase 1 of the case study, and in-line with the quality approach as defined in Sect. 4, the following research questions will be addressed:

- Which views on software quality in the ISO9126 quality model, as described in Sect. 2, are being recognized, and by which particular domain expert?
- For each view: which definitions of ISO9126 quality (sub) characteristics and attributes are recognized as being relevant for a particular domain expert?

- For each ISO9126 quality characteristic definition: is the definition useful for a particular domain expert? And if not: in what way should the definition be redefined or refined?
- Is there a need for defining additional quality (sub) characteristics and/or attributes?

5.2 Views used on software quality

In the introduction of the first workshop session, the different views on quality of ISO 9126 have been presented and discussed. The operations domain experts recognized in particular the 'quality in use' and the 'external' view on software quality; see Figs. 2 and 3. This is not surprising because 'quality in use' represents the user's view on software quality in terms of the results of using the software, rather than in terms of the properties of the software itself. The development domain experts recognized both the 'external' and the 'internal' view. Also this is understandable because of a focus of these specialists on specific 'software internal' design decisions, e.g. to implement particular quality attributes into a software application.

5.3 'Quality in use' software characteristics

In Table 1, the result from the workshop sessions is given regarding the determination of the quality characteristics from the 'quality in use' point of view. In the ISO9126 standard, the 'quality in use' characteristic 'effectiveness' is defined as: 'the extent to which the software product enables the user to achieve specified goals with accuracy and completeness in a specified context of use', (ISO/IEC 9126 2001). Regarding 'effectiveness', three sub characteristics have been identified for the MST functionality, respectively 'completeness of the operational representation', the 'validity of the operational representation', and the 'integrity of the operational representation'; see Table 1. Regarding the 'quality in use' characteristic 'satisfaction' only one characteristic was identified, i.e. 'trust in the operational representation'. Table 1 shows that by the operational domain experts two out of four of the ISO 'quality in use' characteristics have been determined as being relevant.

Subsequently, the MST 'quality in use' characteristics have been specified in more detail, see Fig. 4. As stated before, regarding this further decomposition the 'quality in use' view of ISO9126 does not offer any support. The four 'quality in use' characteristics of ISO 9126 are not further elaborated or hierarchically subdivided in the standard. However, in the definitions provided by the standard, particular quality aspects are stressed, such as

Table 1 'Quality in use' characteristics of the MST

ISO/IEC 9126	MST
Effectiveness	Completeness
	Operational representation
	Integrity operational representation (accuracy)
	Validity of the operational representation
Productivity	–
Safety	–
Satisfaction	Trust in the operational representation

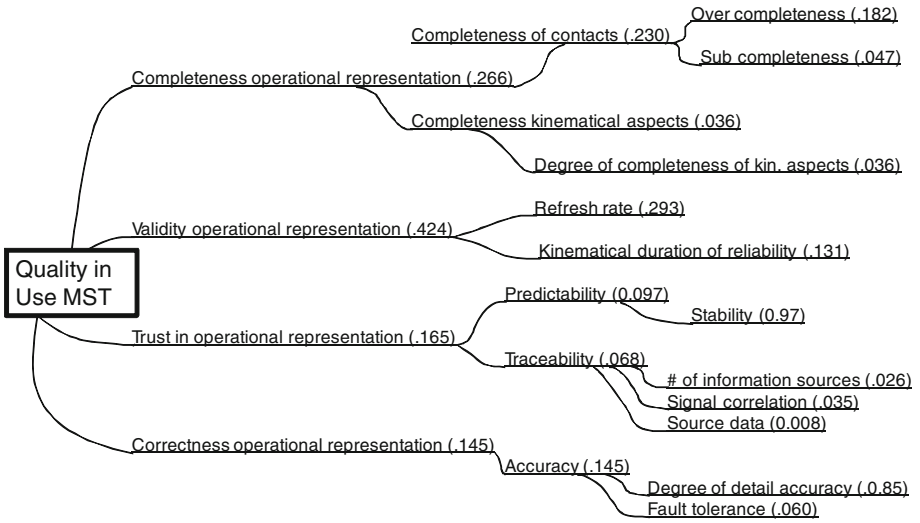


Fig. 4 The MST 'quality in use' view

'completeness' and 'accuracy' in the definition of 'effectiveness'. The hierarchical structure in Fig. 4 has been established in two subsequent workshop sessions. Figure 4 shows that the 'quality in use' characteristic 'completeness of the operational representation', is subdivided into two sub characteristics, respectively 'completeness of the (number of) contacts', and 'completeness of the kinematical aspects of the contacts'. Further, regarding the sub characteristic 'completeness of the contacts' two attributes have been recognized, i.e. 'over completeness' and 'sub completeness'. 'Over completeness' is defined as the extent to which more contacts are given in an operational representation than there are in reality (e.g. as a consequence of dual tracking and/or false alarms). 'Sub completeness' is defined as the extent to which fewer contacts are given in the operational representation than there are in reality, e.g. caused by missing contact detections, or by a too strong information filtering.

Regarding the 'quality in use' characteristic 'validity of the operational presentation', instead of sub characteristics only two attributes have been identified, respectively the 'refresh rate', i.e. the time between two subsequent refreshes of the kinematical aspects of a track, and the 'validity' of the kinematical aspects, i.e. the extent (in seconds) that the offered operational representation is behind reality.

The 'quality in use' characteristic 'satisfaction' is defined as the extent to which the software product satisfies users in a specified context of use (ISO/IEC 9126 2001). Regarding the 'satisfaction' of the operational domain experts, i.e. the end-users, in particular 'confidence in the operational representation' was identified. This quality characteristic is subdivided into two sub characteristics, respectively 'traceability' and 'predictability'. 'Traceability' is defined as the extent to which the information in the operational representation is traceable, i.e. regarding the determination of the sensors which provide information on the tracks, and the certainty with which videos of the radars can confirm the represented position of the tracks (e.g. foreign airplanes). The 'predictability' is defined as the extent to which the represented kinematical aspects can be predicted by an operational expert.

Regarding the research questions of this sub phase 1, the foregoing shows that the 'quality in use' view of the ISO9126 standard supported only to a small extent the particular domain experts, i.e. both developers and end-users, in the identification and specification of software quality (sub)characteristics and attributes. The redefined 'quality in use' characteristics (Table 1) and the refinement of these quality characteristics into (sub) characteristics and the definition of new attributes (Fig. 4) are all based on consensus building between the domain experts in the workshop sessions. It appeared from these sessions that the hierarchical structure in Fig. 4 is in fact the result of a complexity reduction and a further elaboration and refinement of the 'quality in use' view. N.b.: the weight factors in Fig. 4 will be addressed in the RA sub phase 2 in this section.

5.4 'External and internal quality' characteristics

Table 2 shows that regarding the 'external quality' view three quality characteristics have been identified, respectively 'time behavior', 'reliability' and 'usability'. Regarding 'efficiency' explicitly 'time behavior' was stressed, although in the ISO9126 standard also 'resource utilization' is mentioned as a sub characteristic.

Both 'maintainability' and 'portability' were considered to be out of scope regarding the involved operational domain and development domain experts. Functionality was not addressed in the workshop sessions because of the emphasis that was put on software quality as being a set of non-functional characteristics.

Also the MST 'external quality' characteristics have been specified in more detail, and have been refined or subdivided in terms of sub characteristics and internal attributes. Figure 6 shows the resulting MST software quality hierarchy from the 'external and internal quality' point of view. Some of the ISO9126 sub characteristics appeared to be useful for the domain experts, but it was also necessary to define new sub characteristics. Regarding 'reliability', the three sub characteristics of ISO9126, i.e. 'maturity', 'fault tolerance' and 'recoverability' and their definitions could be used, but not in exactly the same way as given in the standard. Two main sub characteristics of 'reliability' were identified during the sessions, respectively 'availability' and 'fault tolerance'. 'Availability' is defined in ISO9126 as a combination of the three sub characteristics 'maturity' (which governs the frequency of failures), 'fault tolerance' (maintaining performance in cases of software faults), and 'recoverability' (which governs the length of downtime following a failure). However, the domain experts decided to make use of the 'combined' quality characteristic 'availability'. Regarding 'availability' two attributes, different from ISO9126, have been identified: 'break down avoidance' and 'break down duration'. 'Break down avoidance' is defined as the extent of redundancy in the functionality, to avoid a breakdown in case a serious failure occurs. 'Break down duration' deals with the duration

Table 2 'External quality' of the MST software

ISO/IEC 9126	MST software
Functionality	–
Reliability	Reliability
Usability	Usability
Efficiency	Time behavior
Maintainability	–
Portability	–

of recovering the software. Regarding 'fault tolerance', the same definition as in ISO9126 was used, i.e. the capability of the software to maintain a specified level of performance in cases of software faults or of infringement of its specified interface. The two attributes that have been defined for 'fault tolerance' are respectively 'overload prevention' and 'failure rate'. 'Overload prevention' is defined as the extent to which the software can prevent the maximal number of tracks which can be handled by the software being reached (graceful degradation is then applied). 'Failure rate' is defined as the extent to which the software is providing the requested functionality. 'Overload prevention' is a very specific MST context-dependent attribute, while 'failure rate', as defined in ISO9126, could be adopted directly.

Regarding 'usability', the ISO9126 framework and terminology could be used more extensively. In ISO9126, 'usability' is defined as the capability of the software to be understood, learned, used and liked by the user, when used under specified conditions. During the workshop sessions, four sub characteristics have been identified, respectively 'understandability', 'learnability', 'operability' and 'attractiveness'. These four sub characteristics are completely in-line with the definitions in the ISO9126 standard. However, the attributes that have been defined differ from the attributes as given by ISO9126. Regarding 'understandability', an attribute 'intuitive understanding' is defined, i.e. the extent to that the software can be understood intuitively. Regarding the ISO9126 sub characteristic 'learnability' for the MST software, the sub characteristic 'easiness of learning' is used. Subsequently two attributes have been defined, respectively 'availability of help functions', and 'availability of training facilities'. The latter two attributes are not defined in the ISO9126 standard but some interrelations can be distinguished between these (new) attributes and the 'operability' sub characteristic of ISO9126, e.g. with respect to attributes such as 'interactive guidance'.

Regarding the quality characteristic 'time behavior', in the workshop sessions three sub characteristics have been identified, respectively 'response behavior', 'storage behavior' and 'frequency of providing information'. 'Response behavior' is defined in the same way as 'response time' in ISO9126, however three different attributes are identified, respectively 'collection time' (i.e. to collect information from the sensors), 'transfer time' (i.e. the algorithmic calculations) and 'presentation time' (i.e. of the operational representations to the operator). 'Storage behavior' is a context-dependent MST quality sub characteristic (not mentioned in ISO9126) and has as an attribute the 'duration of validity' of the kinematical aspects of a track. 'Frequency of providing information' is an attribute that represents the frequency with which the kinematical aspects of the tracks can be changed.

Summarizing it can be stated that the ISO9126 standard offers a useful framework and terminology for the MST software quality specification. However, regarding the research questions formulated in the beginning of sub phase 1 of the case study, it has become clear that the standard cannot be used or applied by software developers in a straightforward way. This means that in a particular software development situation, software developers have to make selections from the provided set of quality characteristics and attributes (both in the 'quality in use' view, the 'external quality' and the 'internal quality' views), that definitions of both sub characteristics and attributes have to be refined, and that 'new' situation-dependent attributes (or even sub characteristics) have to be defined. In particular in the well-elaborated 'external and internal quality' views, many deviations from ISO9126 appeared to be necessary to reach agreements on the specification of quality sub characteristics and attributes of the MST software. N.b.: the weight factors in Fig. 6 will be addressed in the RA sub phase 2 in the following.

5.5 RA sub phase 2: JRP and the prioritization and metrication of software quality

In this sub phase, the following research questions will be addressed in three subsequent steps (2a, 2b and 2c), following the approach as defined in Sect. 4:

- Can the relative importance of the 'quality in use' and the 'external and internal quality' sub characteristics and attributes be determined by using the AHP techniques (as described in Sect. 3)?
- Can measurement scales, as described in Sect. 3, be developed to quantify the quality attributes?
- Can software developers clarify their design decisions and/or calculate the effects of alternative decisions on the overall quality or on particular quality characteristics, e.g. by making use of PI's (see Sect. 3)?

In the first step 2a of this phase, AHP has been applied to determine the relative importance of the 'quality in use' and the 'external and internal quality' (sub) characteristics and attributes. The relative importance is expressed as weight factors reflecting the importance of a particular quality (sub) characteristic or attribute, to a particular (sub) characteristic at a higher level. A weight factor represents the maximum contribution that a (sub) characteristic or an attribute can deliver to a higher level in a quality hierarchy. Note that in Fig. 4 the 'quality in use' characteristic 'validity of the operational representation' has the highest weight factor (i.e. 0.424), and as such can contribute to the highest to 'quality in use'. The attribute 'refresh rate' has the highest weight factor of the attributes of 'validity of the operational representation' and contributes to the highest to this quality characteristic. Figure 5 gives an overview of the weight factors of the 'quality in use' attributes in a bar chart.

Figure 6 shows that the 'external quality' characteristic 'reliability' has received the highest weight, or the highest level of importance regarding 'overall' external software quality. Note also that three attributes of 'reliability', respectively 'overload prevention', 'redundancy' and 'failure rate' have received the highest weights in the whole 'external' and 'internal quality' hierarchical structure. From the top toward the lower levels in both quality hierarchies the quantifiability of the identified attributes increases. Note that regarding the quality characteristic 'reliability' it is not possible to determine a single metric. In fact a complex compound set of metrics is needed to measure 'reliability'. However for the identified 'lower' reliability sub characteristics and attributes, such as 'failure rate' and 'overload prevention', measurement is possible on the basis of single metrics.

In step 2b for each of the attributes measurement scales have to be determined. This can be considered as the metrication of software attributes (Hall and Fenton 1994). These scales are needed to make clear to the different involved parties in what way a quality attribute can contribute to a particular quality (sub) characteristic, and ultimately to the overall software quality. Developers can decide what type of design decisions and development activities are needed, depending on the norm that has been defined on the scale, to realize or implement a particular quality attribute. The norm, i.e. the target value on the scale has to be based on consensus between the different involved parties, e.g. the operational domain experts, the development domain experts and project management. The measurement scales and the agreed norms of course also play a role in the verification and validation of the software to determine whether the agreed quality levels have been reached.

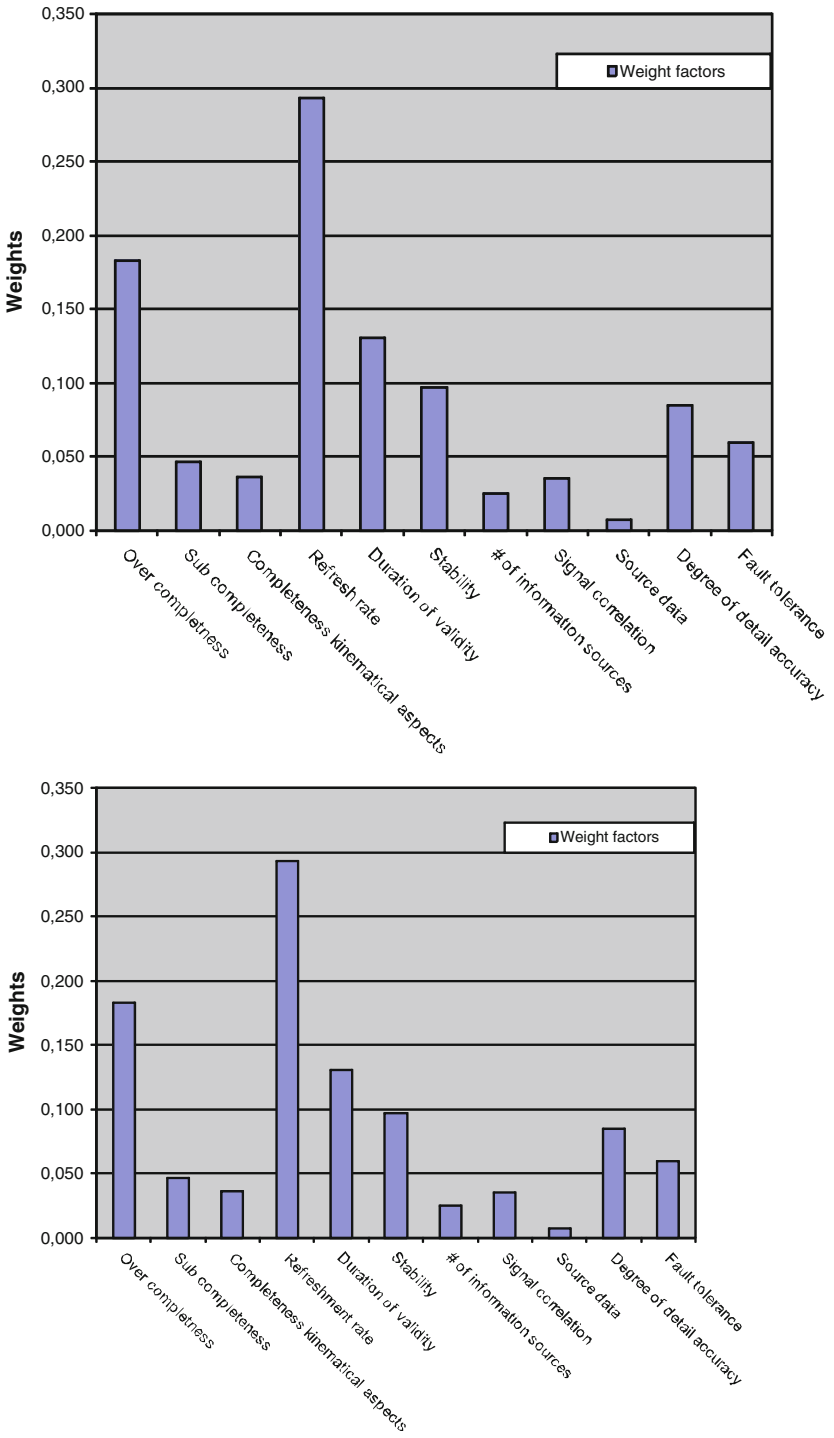


Fig. 5 The relative importance of quality attributes in the 'quality in use' view

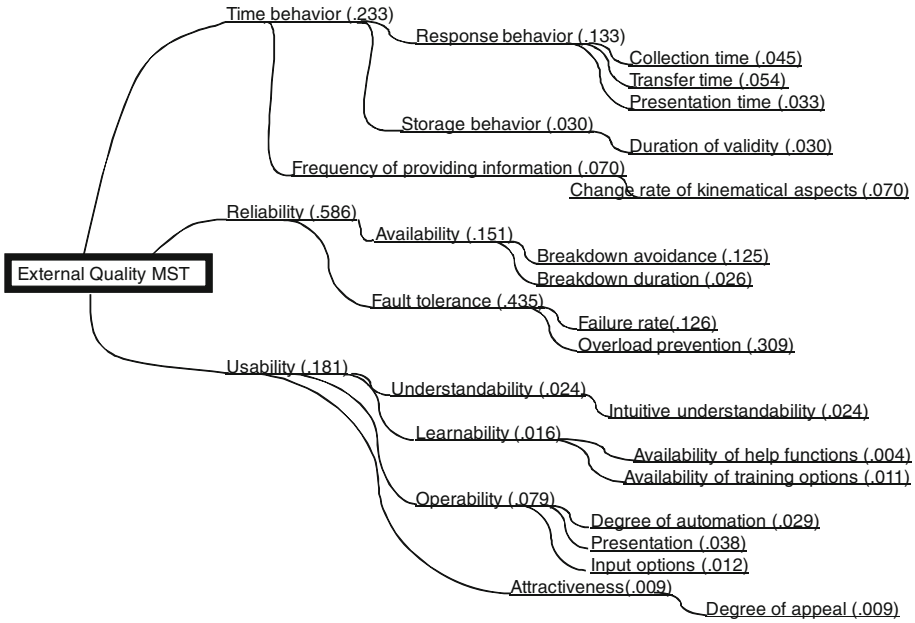


Fig. 6 The MST 'external and internal quality' view

In this case study, two types of measurement scales have been used, respectively a natural and a constructed scale. The natural measurement scale is a so-called continuous scale, with as dimension the time. The constructed scale is divided in disjunctive categories. The measurement scales have defined numbers of categories. Each category represents a design decision which contributes to a certain extent to a particular quality (sub) characteristic.

The measurement scales have been developed in collaboration with the development domain and the operational domain experts. The categories and the extent to which they contribute to a particular quality characteristic have been determined in two workshop sessions. Regarding the determination of the contribution factors, of the different categories of a scale, AHP techniques have been used. In the following, we give two examples of the determination of measurement scales, their categories and accompanying contribution factors. The first example addresses a natural scale for the quality attribute 'refresh rate' of the 'quality in use' characteristic 'validity of the operational presentation'. This attribute scores the highest weight (i.e. 0.293) in the 'quality in use' view, see Fig. 5. The second example addresses a constructed scale for the quality attribute 'overload prevention' of the sub characteristic 'fault tolerance' of the 'external quality' characteristic 'reliability'. 'Overload prevention' scores the highest weight (i.e. 0.309) in the 'external quality' view, see Fig. 7.

5.6 Example 1: metrication of 'refresh rate'

'Refresh rate' is defined as the time in seconds that passes between two subsequent refreshes of operational representations of the kinematical aspects of a track. On the measurement scale, five 'continuous' categories have been defined, see Table 3. Category 5 is the highest level on the scale, i.e. a refresh takes place within 1-s. At the lowest level 1, a refresh takes place only after 4–5-s. For each of the five categories so-called contribution

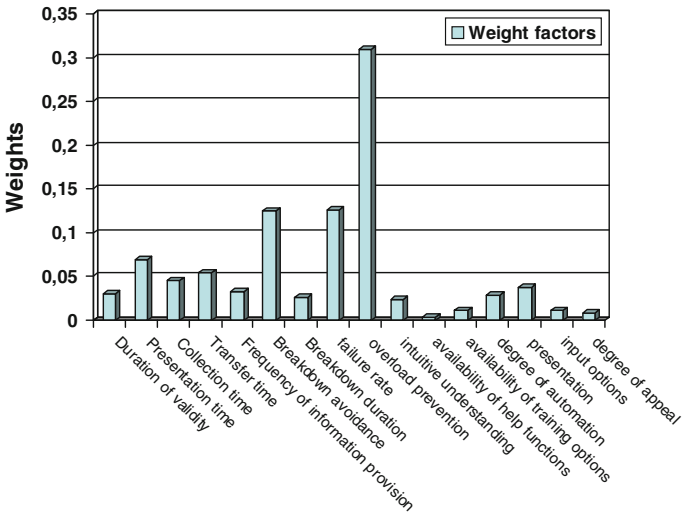


Fig. 7 The relative importance of quality attributes in the 'external and internal quality' view

Table 3 Natural measurement scale of the quality attribute 'refresh rate'

Category	Refresh time of the kinematical aspects in the operational representation	Contribution factor
1	Between 4 and 5 s	0.07
2	Between 3 and 4 s	0.19
3	Between 2 and 3 s	0.38
4	Between 1 and 2 s	0.65
5	Between 0 and 1 s	1

factors have been determined, again by making use of AHP techniques. The result is shown in the rightmost column in Table 3. E.g. a contribution factor of 0.65 means that if a 'refresh rate' is implemented between 1 and 2 s, this implementation results in a contribution of 0.65 multiplied with the weight factor (i.e. 0.293), to the 'quality in use' characteristic 'validity of the operational representation'.

5.7 Example 2: metrification of 'overload prevention'

The quality attribute 'overload prevention' has the highest weight of the four attributes of the quality characteristic 'reliability'. 'Overload prevention' is defined as the degree to which the software prevents the maximum number of tracks being reached (that can be handled by the software). The objective of 'overload prevention' is to determine whether so-called *graceful degradation* has to be applied. This means that in the case of an overload threat, the tracks with a lower priority will not be handled by the software, but that the software application continues operating in a normal way for the other tracks. 'Overload prevention' has two different aspects. The first one is the detection of an overload, and the second one is the prevention of an overload by decreasing the extent to which the software functionality is used. Detection and prevention can be carried out in different ways, respectively: not carried out, carried out by an operator or carried out by the software

Table 4 Possible detection and prevention of overload situations

Prevention detection	Not	Operator	Automatically
Not	1	X	X
Operator	2	3	X
Automatically	X	4	5

automatically, see Table 4. The numbers in Table 4 represent the identified categories. The crosses (X) represent the situations that are considered to be not realistic, e.g. for an overload that cannot be detected, it is also not possible to prevent it. For the five categories, contribution factors have been determined by making use of AHP techniques, see Table 5. E.g. the contribution factor 0.94 means that if an overload is prevented in accordance with category 4, the contribution to the 'operational reliability' is 0.94 multiplied with the weight of the 'overload prevention' (i.e. 0.309). If the implementation of 'overload prevention' is carried out in accordance with category 5, the contribution of 'overload prevention' to 'operational reliability' is 1 multiplied with the weight factor 0.309.

5.8 Step 2c: calculation of the performance index (PI) of the quality attributes

Based on the weights and contribution factors, the so-called PI's can be calculated for each quality characteristic. The quality characteristic with the highest PI gets the highest priority in the process of software development. A PI is defined as the sum of the multiplications of the weights of the attributes with the contribution factors (as determined on the scales). The PI of each quality characteristic expresses the relative contribution to the overall quality, e.g. the 'quality in use' or the 'external quality' of the software. Figure 8 presents a schematic overview of the calculation of the performance index.

The calculation of the PI is based on the following formula and definitions:

$$\mathrm{PI} - Cj = \left(\sum_{i=1}^{n} (\mathrm{WA}ij * Uij) \right) \Big/ \mathrm{WC}j * 100\%$$

Table 5 Constructed measurement scale of the quality attribute 'overload prevention'

Category	Description	Contribution factors
1	An overload threat will not be detected and will not be prevented. As a consequence no track is being processed by the software	0.07
2	An overload threat will be detected by an operator, but cannot be prevented. Consequently all tracks are being processed by the software with quite some delay	0.17
3	An overload threat will be detected by an operator but is not prevented in an automated way. The consequence is that all low priority tracks are being deleted by the operator and all high priority tracks are processed by the functionality	0.4
4	An overload threat will be detected automatically but will not be prevented automatically. Consequently, all low priority tracks have to be deleted by the operator and all high-priority tracks can be processed by the software	0.94
5	An overload threat will be detected automatically and prevented automatically so that all high-priority tracks are being processed by the software	1

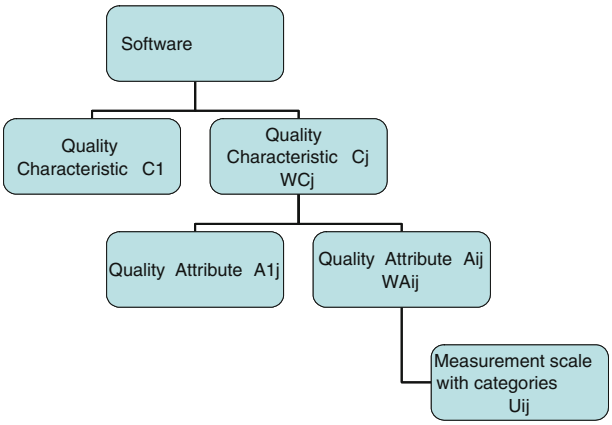


Fig. 8 Schematic overview of the calculation of performance indexes

n = number of attributes per quality characteristic, PI-Cj = performance Index of quality characteristic Cj, WCj = weight factor of quality characteristic Cj, WAij = weight factor of quality attribute Ai of quality characteristic Cj, Uij = contribution factor of the particular category of the measurement scale of quality attribute Aij.

In the following, an example is given of a calculation of a PI for the 'quality in use' characteristic 'validity of the operational representation' (Cj). In the formula given above the weight factor (WCj) of the quality characteristic Cj is 0.424. The Weight factors (WAij) of the 'quality attributes' Aij, which are respectively 0.293 for 'refresh rate' and 0.131 for 'time validity', have to be multiplied by their contribution factors. If we assume that the contribution factor for 'refresh rate' is chosen as 0.38, and the contribution factor for 'time validity' is chosen as 1 then the outcome of the formula is a PI of 57%. This means that by choosing these contribution factors for these two attributes of the quality characteristic 'validity of the operational representation' only 57% of the (maximum) quality level of the quality characteristic will be reached. Based on this PI information, software developers can clarify their design decisions, calculate the effects of alternative decisions on the overall quality, or on particular quality characteristics, and discuss with different stake-holders the impact of different design scenarios.

RA sub phase 2 of the case study shows that AHP prioritization techniques could be applied successfully by software developers, respectively to determine on the one hand the weight factors, i.e. the relative importance of the quality (sub) characteristics and attributes, and on the other hand the contribution factors of the particular design decisions to implement software quality. The case study showed that software quality attributes can be metricated by software developers, and the effectiveness of particular design decisions with respect to quality attributes can be determined quantitatively.

6 Conclusions

The CAMS department of the Dutch Department of Defense has investigated an approach for software developers to deal in a formal and systematic way with software product quality, in particular the specification, the prioritization and the metrication of software quality. The approach has been developed on the theoretical basis of on the one hand the

ISO9126 conceptual framework and terminology, and on the other hand the AHP prioritization and metrication techniques.

The case study demonstrates that the ISO9126 standard offers software developers a useful and an applicable conceptual framework and terminology for the specification of software product quality. The hierarchical structure of the ISO9126 standard could be applied to reduce complexity in the 'overall' quality concept of a software application. However, a number of adaptations had to be made. First of all, it appeared to be necessary to (re)define (new) quality characteristics, both regarding the 'quality in use' view and the 'external quality' view. Some of the ISO9126 quality characteristics appeared to be not relevant in the case study. Also the subdivision of the quality characteristics into sub characteristics and attributes toward lower levels in the quality hierarchies shows many differences with respect to the ISO9126 standard. In the case study, many quality attribute definitions had to be reinterpreted and/or translated into the specific technical context of the particular software application. The overall lesson learned with regard to ISO9126 is that software developers need to tailor the standard in their specific software development situation. The objective is to reach a context-specific quality model that fits with the needs of the software developers with regard to the specification of the required software quality.

The case study results show also that AHP techniques can be applied successfully to implement the required software quality. Although ISO9126 offers a useful framework and appeared to be not too time-consuming in the specification of software quality, the application of the AHP techniques required quite some time and effort, despite the fact that the automated tool expert choice could be used. This problem of applying AHP in practice has also been recognized in previous research; see e.g. (Weil and Apostolakis 2001; Karlsson et al. 2004). However, it has also been emphasized that in high-tech industrial projects, in which larger numbers of requirements have to be prioritized, the application of AHP could be valuable and should be investigated further in case studies (Karlsson et al. 2004). The overall lesson learned with regard to AHP is that priorities regarding software quality attributes can be made explicitly and quantitatively by the software developers. Also the implementation of these quality attributes can be based on well-defined measurement scales. The objective is that software developers ultimately deliver software products of which there is no confusion about both the quality from the end-user point of view and the developer point of view.

Finally, based on the positive experiences and the lessons learned from the case study, it has been decided to adopt the stepwise software quality approach for the software developers and to elaborate the approach further toward an operational method with accompanying techniques and tools, well-integrated in the existing DSDM at CAMS. Regarding the specification of software product quality, it was decided to adopt the ISO9126 standard as the theoretical basis for software quality specification. However, based on the experiences, guidelines have been developed to adapt or tune the ISO9126 standard toward particular software application domain. Regarding the prioritization and metrication it was decided to implement AHP techniques in the stepwise approach to support de decision-making processes. To support and to speed up the application of AHP techniques, CAMS-specific guidelines and procedures have been developed, in particular to structure the discussions in the workshop sessions, and to strive at earlier consensus on AHP outcomes such as the weighting factors, the measurement scales and the contribution factors.

To improve the efficiency of dealing with software quality further, it was decided to start a project on the development of a knowledge base on software quality aspects, such as software quality profiles for the various types of software applications (their quality

characteristics and attributes), and the accompanying measurement scales and performance indexes. Based on these results, the authors believe that their research objectives have been met and that further steps can be made toward a professional methodology for software developers regarding the specification, prioritization and metrication of software quality.

References

Behkamal, B., Kahani, M., & Kazem Akbari, M. (2009). Customizing ISO9126 quality model for evaluation of B2B applications. *Information and Software Technology, 51*, 599–609.

Boehm, B. W. (1978). *Characteristics of software quality*. Amsterdam: North-Holland.

Botella, P. X., Burgues, J. P., Carvallo, X., & Franch, C. (2004). Quer, Using quality models for assessing COTS selection. *In Proceedings of MPEC'04 and ICSE'04*.

Botella, P., Burgues, X., Carvallo, J. P., Franch, X., Pastor, J. A., & Quer, C. (2003). Towards a quality model for the selection of ERP systems. In A. Cechich, M. Piattini, A. Vallecillo (Eds.), *Component-based software quality: methods and techniques* (pp. 225–246). Springer, LNCS 2693.

Franch, X., & Carvallo, J. (2002). A quality-model based approach for describing and evaluating software packages. In *Proceedings IEEE joint international conference*.

Hall, T., & Fenton, N. (1994). Implementing software metrics—the critical success factors. *Software Quality Journal, 3*(4), 195–208.

Fenton, N. E., & Pfleeger, S. L. (1998). *Software metrics, a rigorous and practical approach*. Boston, USA: PWS Publishing.

ISO/IEC 9126 (2001). Software engineering—software product quality, parts 1, 2 and 3, international organization for standardization, Geneve, 2001 (part 1), 2003 (parts 2 and 3).

Linstone, H., & Turoff, M. (Eds.) (2002). The delphi method: Techniques and applications. Web edition available at http//:www.is.njit.edu/pubs/delphibook/.

Karlsson, L., Berander, P., Regnell, B., & Wohlin, C. (2004). Requirements prioritization: An experiment, on exhaustive pair-wise comparisons versus planning game partitioning. *Proceedings 8th Conference on Empirical Assessment in Software Engineering*, *Edinburgh, UK*.

Karlsson, J., Wohlin, C., & Regnell, B. (1998). An evaluation of methods for prioritization software requirements. *Information and Software Technology, 39*, 939–947.

Karydas, D. M., & Gifun, J. F. (2006). A method for the efficient prioritization of infrastructure renewal projects. *Reliability Engineering and System Safety, 91*, 84–99.

Kitchenham, B. A., et al. (1997). The SQUID approach to defining a quality model. *Software Quality Journal, 6*(3), 211–233.

Maiden, N. A. M., & Rugg, G. (1996). ACRE: Selecting methods for requirements acquisition. *Software Engineering Journal, 11*(3), 183–192.

Mansar, S. L., et al. (2009). Development of a decision-making strategy to improve the efficiency of BPR. *Expert Systems with Applications, 36*(3), part 2, 3248–3262.

Musa, J. D. (1993). Operational profiles in Software-reliability engineering. *IEEE Software*.

Roy, B. (1996). *Multicriteria methodology for decision aiding*. Dordrecht: Kluwer Academic Publishers.

Royce, W. (1970). Managing the development of large software systems. *Proceedings IEEE WESCON*.

Saaty, T. L. (2001). *Models, methods concepts and applications of the analytic hierarchy process*. Dordrecht: Kluwer Academic Publishers.

Svahnberg, M., Wohlin, C., Lundberg, L., & Mattsson, M. (2002). A method for understanding quality attributes in software architecture structures. In *Proceedings of the 14th international conference on Software Engineering and Knowledge Engineering, SEKE*.

Trienekens, J. J. M., & Kusters, R. J. (1999). *Identifying embedded software quality, two approaches*. Addison Wesley: Quality and Reliability Journal.

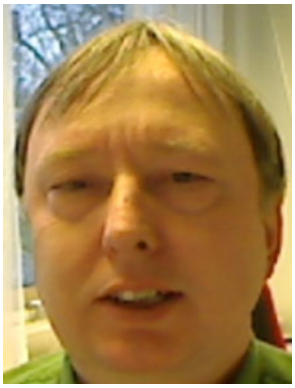
Weil R., & Apostolakis, G. E. (2001). A methodology for the prioritization of operating experience in nuclear power plants. *Reliability Engineering and System Safety, 74*, 23–42.

Wohlin, C., & Mattson, M. (2005). Guest editorial on the special issue: trade-off analysis of software quality attributes. *Software Quality Journal, 13*(4), 327–328.

Zhu, L., Aurum, A., Gorton, I., & Jeffrey, R. (2005). Tradeoff and sensitivity analysis in software architecture evaluation using analytic hierarchy. *Process, Software Quality Journal, 13*, 357–375.
Zuse, H. (1998). *A framework of software measurement*. Berlin, New York: De Gruyter Publishing.

Author Biographies

Dr. Jos J. M. Trienekens (1952) is an Associate Professor at TU Eindhoven (University of Technology—Eindhoven) in the area of ICT systems development. He is responsible for a research program on ICT-driven business performance and is an associate member of the research school BETA at TUE that focuses at operations management issues. Jos Trienekens published over the last 10 years various books, papers in international journals and conference proceedings in the domains of software quality and software process improvement. He joined several international conferences as PC member and member of the organization committee. He is also an experienced project partner in various European projects.



Prof. Dr. Rob J. Kusters (1957) obtained his master degree in econometrics at the Catholic University of Brabant in 1982 and his Ph.D. in operations management at Eindhoven University of Technology in 1988. He is professor of 'ICT and Business Processes' at the Dutch Open University in Heerlen where he is responsible for the master program 'Business process Management and IT'. He is also an associate professor of 'IT-enabled business process redesign' at Eindhoven University of Technology where he is responsible of a section of the program in management engineering. He published over 90 papers in international journals and conference proceedings and co-authored six books. Research interests include process performance, enterprise modeling, software quality and software management.

Dennis Brussel received a M.Sc. in Industrial Engineering and Management Science from the Technical University Eindhoven, The Netherlands in 2006. After his career in the Royal Netherlands Navy he acted as management consultant in the finance industry. He is now Chief Information Officer at WestlandUtrecht Bank.