Software quality requirements

Požadavky na jakost softwaru

J. Vaníček

Czech University of Agriculture, Prague, Czech Republic

Abstract: At the present time, the international standards and technical reports for system and software product quality are dispersed in several series of normative documents (ISO/IEC 9126, ISO/IEC 14598, ISO/IEC 12119 etc.). These documents are not purely consistent and do not contain a tools for exact requirements set-ups. As quality is defined as a degree to which the set of inherent characteristic fulfils requirements, the exact requirement formulation is the key point for the quality measurement evaluation. This paper presents the framework for quality requirements for software, which is recommendable to use in the new international standard series ISO/IEC 250xx developed on the *SQuaRE* (*S*oftware *Qua*lity *R*equirements and *E*valuation) standardisation research project. The main part of this contribution was presented on the conference Agrarian Perspectives XIV, organised by the Czech University of Agriculture in Prague, September 20 to 21, 2005.

Key words: software quality; quality requirements; requirements for requirements formulation; international standardization; project SQuaRE

Abstrakt: V současnosti jsou mezinárodní normy a technické zprávy pro jakost systémů a softwaru rozptýleny do několika řad normativních dokumentů (ISO/IEC 9126, ISO/IEC 14598, ISO/IEC 12119 a další). Tyto dokumenty nejsou zcela navzájem konsistentní a neobsahují nástroje pro přesné stanovení požadavků. Protože jakost je definována jako stupeň splnění požadavků souborem inherentních znaků, je přesná formulace požadavků klíčovým bodem v procesu měření a hodnocení jakosti. Tento článek představuje rámec pro formulaci požadavků na jakost softwaru, který by měl být užit v nové řadě mezinárodních norem ISO/IEC 250xx, vyvíjených v rámci mezinárodního vědeckého normalizačního projektu *SQuaRE* (Požadavky na jakost software a jejich hodnocení). Podstatná část tohoto příspěvku byla přednesena na konferenci Agrární perspektivy XIV, uspořádané Českou zemědělskou univerzitou v Praze 20.–21. září 2005.

Klíčová slova: jakost softwaru; požadavky na jakost; požadavky na formulaci požadavků; mezinárodní normalizace; projekt SQuaRE

In the current state, software product quality international standards consist of several standards, which are not fully consistent. The international research standardisation project *SQuaRE* (*S*oftware *Qua*lity *R*equirements and *E*valuation) is running with the aim to develop a new consistent standard series ISO/IEC 250xx for the software product quality, evaluated from the users and stakeholders point of view. More details on this project are reported in the article (Vaníček 2005) in the proceedings of this conference. In Vaníček (2005), there is also listed a more extensive bibliography concerning our problems.

Quality is generally defined as a degree to which a set of inherent characteristic fulfils the requirements. Requirements are needs or an expectation that is stated, generally implied, or obligatory. One of the weakest spots of the today's stage of software product quality standards is the absence of the guidelines for the exact quality requirements formulation. For the unbiased quality evaluation such requirements have to be clear, complete, consistent and measurable.

The present contribution tries to draw the line how to define the stack holders requirements for software product to facilitate the objective quality evaluation and will be provided as an input for the requirement part ISO/IEC 2503x of standards in SQuaRE standard series.

Supported by the Ministry of Education, Youth and Sports of the Czech Republic (Grant No. MSM 6046070904 – Information and knowledge support of strategic control).

SOFTWARE AND SYSTEMS

The topic of this contribution is the software quality requirements framework. However, software usually appears as a part of a larger system. Therefore, it can be useful to take a system view. A system is defined as a combination of interacting elements organised to achieve one or more stated purposes. This definition allows a high degree of freedom to decide what constitutes a system, and the boundaries of a system will depend very much on the point of view. In this way, the system of interest is the system whose life cycle is under consideration in the context of this contribution.

The boundary of a system depends on the point of view as illustrated by the following three examples. One example is the control system of an aircraft engine, the second example is the complete engine of an aircraft, and a third example is the complete aircraft. An aircraft can be considered as a combination of elements (the engines, the wings, etc.). These elements can also be considered systems on their own.

What constitutes the elements of a system depends son the point of view and there may be several different appropriate ways of defining the elements of a system. Software may be considered one of the elements of a system. Figure 1 illustrates one way of looking at the elements constituting a system. The illustration does not show the interaction between elements.

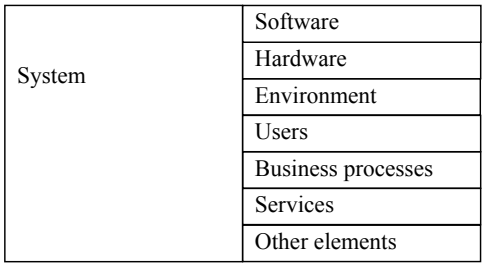

Figure 1. Elements of a system

As illustrated in Figure 1, software can be viewed as an element of a system, and it is often appropriate to view software as a part of a larger system. However, software can also be viewed in isolation. The most appropriate view depends on the purpose.

STAKEHOLDERS AND STAKEHOLDER REQUIREMENTS

The stakeholders of a system include all persons, organizations and bodies having a legitimate interest in the system in question. Systems have a variety of stakeholders who have an interest in the system throughout its life cycle. Stakeholders have different needs and expectations to the system. Their needs and expectations may change throughout the systems life cycle. Stakeholders include individuals (for example end users), organisations (for example end user organisations or development organisations), and society (for example the statutory and regulatory authorities or the general public).

Stakeholder needs can be explicitly stated or only implied. In some cases, stakeholders are not aware of some of their needs. In some cases, the real needs of some stakeholders are different from what they express. Implied needs are often implied by the context where the software product is to be used and represent expectations based on similar software products or existing work routines, normal working procedures and operations of business, laws and regulations, etc. In many situations, stakeholder needs only become evident when the software product and the related business processes or tasks can be tried out. Scenarios use cases, and prototypes represent methods to identify needs at an early stage of a development project.

The stakeholders' needs and expectations are identified through a requirements elicitation and definition process as illustrated in Figure 2. The process takes all stakeholders' needs, wants, desires, and expectations into consideration. This includes the

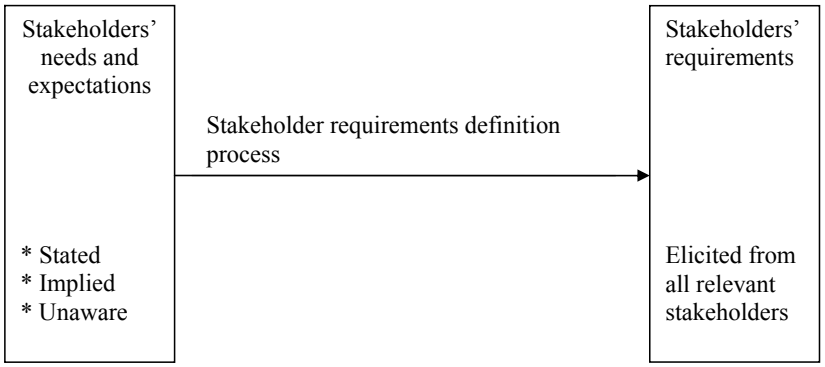

Figure 2. Stakeholder requirements definition

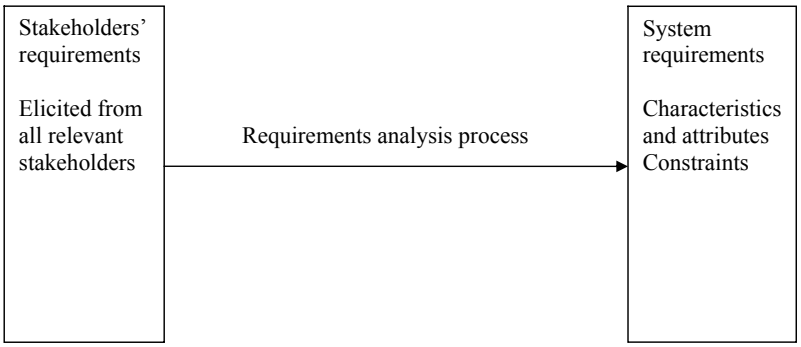

Figure 3. Requirements analysis

needs and requirements imposed by society, the constraints imposed by the acquirer, and the needs from the end users.

Different categories of stakeholders often have different requirements. In some situations, stakeholders have conflicting requirements. Conflicts between stakeholder requirements could for example be between different end user perspectives, or between acquirer needs and available skills, experiences or resources in the developing organisation.

The result of the definition process is called stakeholders requirements.

An analysis process aims at making a transformation of stakeholder requirements into a technical view of system requirements that can be used to realise the desired system, see Figure 3. The technical view of requirements is called system requirements. System requirements are verifiable and will state which characteristics the system is to possess in order to satisfy stakeholder requirements.

A system will often be composed of different elements, each with specific characteristics and serving different purposes in the whole system. In order to be operational, system requirements have to be formulated as requirements to the different system elements. As different elements interact to offer the system capabilities, requirements to different system elements cannot be seen in isolation, but only in a broader view including requirements to other system elements. System requirements may imply requirements to for example software, but it is not always clear whether or not a system requirement implies a software requirement. System requirements can be implemented in different ways, for example either in hardware or in software or as a business process (i.e. a manual process).

SOFTWARE QUALITY

Software quality is the degree to which the software is capable to provide and maintain a specified level of service. The required level of service is specified according to a quality model. The software product quality model provided in ISO/IEC 9126-1 and also prepared ISO/IEC 25010 defines six quality characteristics: functionality, reliability, usability, maintainability, portability, and efficiency. In addition, the quality model defines quality in use in terms of four characteristics: effectiveness, productivity, safety and satisfaction. The quality characteristics have defined sub-characteristics and the standard allows for user-defined sub-sub-characteristics in a hierarchical structure. The intention is that the defined quality characteristics cover all quality aspects of interest for most software products and as such can be used as a checklist for ensuring the complete coverage of quality.

The quality model suggests three different views on quality: internal quality, external quality, and quality in use. Internal quality provides a 'white box' view of software and addresses properties of the software product that typically are available during the development. External quality provides a 'black box' view of the software and addresses properties related to the execution of the software. The quality in use view is related to application of the software in its operational environment, for carrying out the specified tasks by specified users. Internal quality has an impact on external quality, which again has an impact on quality in use. These impact relations are shown in Figure 4.

Internal quality is mainly related to static properties of the software, external quality is mainly related to dynamic properties of the software, and quality in use is mainly related to dynamic properties of the system of which the software is an element.

The quality model serves as a framework to ensure that all aspects of quality are considered, both from the internal, external, and quality in use point of view.

SOFTWARE PROPERTIES

The capabilities of a software product are determined by its properties. Some software properties are

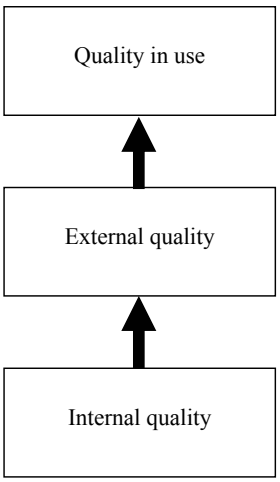

Figure 4. Software product quality

inherent to the software product; some are assigned to the software product. "Inherent", as opposed to "assigned", means existing in something, especially as a permanent characteristic or feature. Examples of inherent properties are number of lines of code and the accuracy of a numeric calculation provided by the software. Examples of the assigned properties are the owner of a software product and the price of a software product.

Inherent properties can be classified as either functional properties or quality properties. Functional properties determine what the software is able to do. These properties relate to the environment, in which the software can be executed (for example operating system), the interfaces to other software, and the functions the software is able to perform. Quality properties show how well the software performs. In other words, the quality properties show the degree to which the software is able to provide and maintain its specified services. Figure 5 illustrates this classification of software properties.

Quality is something that is inherent to a software product. An assigned property is therefore not considered to be a quality characteristic of the software, since it can be changed without changing the software.

The software quality properties are classified according to the characteristics and sub-characteristics defined in the ISO/IEC 25010 software quality model, i.e. functionality, reliability, etc.

SOFTWARE QUALITY MEASUREMENT MODEL

A statement about the quality of a software product is an expression about the capability of the software to perform and maintain a specified level of service. Software quality can therefore be formulated as the degree to which the software is capable to perform and maintain a specified level of service. The approach to software quality follows a measurement approach, i.e. it is based on measuring the degree to which software properties fulfils specified requirements.

Inherent software properties, that can be distinguished quantitatively or qualitatively, are called attributes. Quality attributes are measured as the degree to which the software product is capable of providing and maintaining its specified level of service. Quality attributes belong to one or more (sub)-characteristics.

Quality attributes are measured by applying a measurement method. A measurement method is a logical sequence of operations used to quantify an attribute with respect to a specified scale. The result of applying a measurement method is called a base measure. The quality characteristics and sub-characteristics can be quantified by applying measurement functions. A measurement function is an algorithm used to combine base measures. The result of applying a measurement function is called a quality measure. In this way, quality measures become quantifications of the quality characteristics and sub-characteristics. More than one quality measure may be used to describe a quality characteristic or sub-characteristic. Not all software properties are measurable. This is in particular the case for functional properties.

Figure 6 shows the relations between the ISO/IEC 25010 quality model and the measurement model suggested in ISO/IEC 15939.

<table>
<tr><td rowspan="3">Software properties</td><td rowspan="2">Inherent properties</td><td>Functional properties</td></tr>
<tr><td>Quality properties</td></tr>
<tr><td>Assigned properties</td><td>For example price, delivery date, product future, product supplier</td></tr>
</table>

Figure 5. Software properties

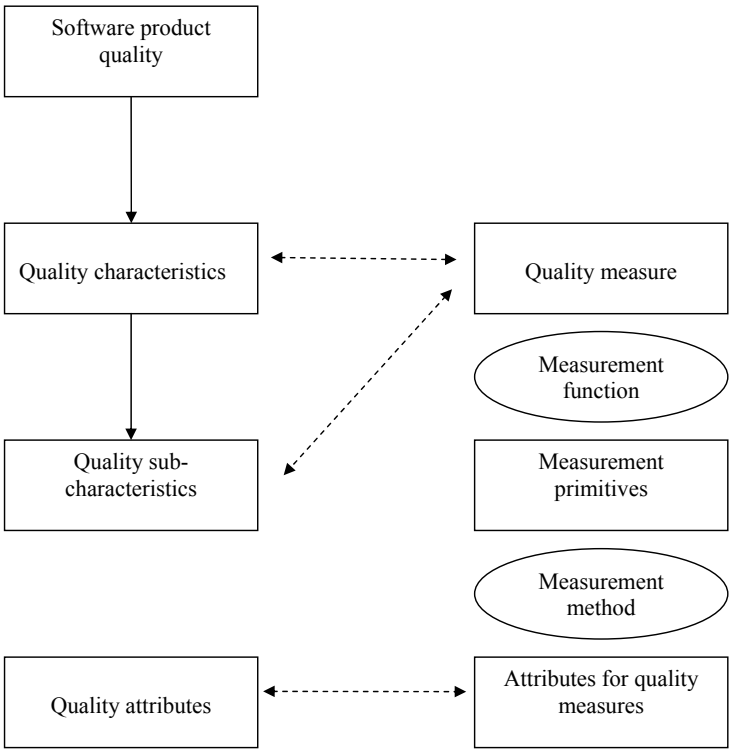

Figure 6. Quality model and measurement model

SOFTWARE QUALITY REQUIREMENTS

A measurement function represents an interpretation of a software quality property and a target value of the quality measure represents a quality requirement. Similarly, the actual value of the quality measurement represents the observed quality of the software.

Software quality requirements as well as all other requirements cannot be seen in isolation, but must be view in a broader context. Software quality requirements have a particular close relation to functional requirements. Functional requirements play an important role for specifying software quality requirements. Functionality is one of the six characteristics for internal and external quality in ISO/IEC 9126-1 and ISO/IEC 25010. Functionality requirements should not be confused with functional requirements. Functionality is the capability of the software to provide functions, which meet its functional requirements. Functionality requirements are refined into requirements for the software product to be suitable, accurate, interoperable, secure and compliant with the relevant functional standards and regulations.

In some situations, it is meaningful to specify a quality requirement for a software product, whereas in other situations a quality requirement only applies to a portion of the software product. For example, some functions are only relevant for specific users and have specific quality requirements, which are different from quality requirements for other functions intended for other purposes and other users. It is therefore important to specify which portion of a software product is relevant for a software quality requirement. In other words, a quality requirement is linked to a portion of the software product (a set of functions).

For example, some functions may be intended for general end users and may hence require low error tolerance, whereas another group of functions may be intended for specialists and thus permit a greater error tolerance. In both cases, the error tolerance mechanism and the degree of error tolerance required should be rigorously specified.

Quality in use is defined as the extent to which a product meets the needs of specified users to achieve specified goals with effectiveness, productivity, safety and satisfaction in a specified context of use. Therefore quality in use requirements are closely related to other system requirements such as hardware requirements, business requirements, and requirements to end-user.

SYSTEM REQUIREMENTS CATEGORISATION

A system consists of a number of interacting elements. They can be categorised in different ways. Figure 1 provides one possible categorisation. Figure 7 provides a similar categorisation of system requirements. System requirements can for example include requirements for software, hardware, organisation,

and may come from a variety of stakeholders including end users and organisations.

Software requirements address, on one hand, the software product or the software development process. There will often be dependencies between software development requirements and software product requirements.

Software development requirements may for example include requirements for artefacts, processes, project, development organisation, and developers. Requirements to the software development process are often based on a hypothesis that a 'good' development process will ensure a 'good' software product.

Quality requirements life cycle model

Stakeholder requirements are defined from stakeholder needs. Stakeholder needs come from many sources. When the system in question is completely new, i.e. no similar systems exist; it may be difficult to identify the real needs of stakeholders. In other situations a system exists, but it is realised in a different way, for example as a manual business process. In that case stakeholders will usually be able to express many needs and expectations.

Figure 8 illustrates the situation where a similar system already exists and is in use. In that case the users of the existing system will be a major source of input to the requirements of the new system. Stakeholder needs can largely be identified based on experiences from actual use of the existing system. Stakeholder requirements are defined from the stakeholder needs as shown in Figure 3. An analysis of stakeholder requirements will produce system requirements as shown in Figure 4. Software quality requirements are derived as part of the system requirements. There are three views of software quality as shown in Figure 5. These different views give rise to three types of software quality requirements:

– Quality in use requirements
– External quality requirements
– Internal quality requirements

Quality in use requirements are typically derived from stakeholder requirements such as a) business requirements (company policy, competitors, etc.), b) functional requirements, and c) application domain specific requirements.

External quality requirements are typically derived from a number of sources including a) stakeholder requirements, b) legal requirements, c) standards and guidelines for the relevant application, d) quality in use requirements, e) functional requirements, f) application domain specific requirements, and g) security requirements, which may be derived from risk analysis.

Internal quality requirements are typically derived from a number of sources including a) external quality requirements, b) company policy, c) development policy and limitations, and d) best practice guidelines.

Quality in use requirements may imply external quality requirements and similarly external quality requirements may imply internal quality requirements. The software implementation process realises the software quality requirements. The quality of the new system can be used as input to another new system again, thereby completing the cycle indicated in Figure 8.

Software requirements can be specified as a part of an iterative development process, where one version of the software is used as a basis for deciding the requirements for the next version. In general, it is the functional requirements that evolve in this way, whereas the quality requirements are fixed. However,

<table>
<tr><td rowspan="6">System requirements</td><td rowspan="5">Software requirements</td><td rowspan="3">Software product requirements</td><td rowspan="2">Inherent property requirements</td><td colspan="2">Functional requirements</td></tr>
<tr><td rowspan="3">Quality requirements</td><td>Quality in use</td></tr>
<tr><td>External quality</td></tr>
<tr><td>Internal quality</td><td colspan="2" rowspan="2">Include for example requirements for price, delivery date, product future, and product supplier</td></tr>
<tr><td>Assigned property requirements</td></tr>
<tr><td rowspan="2">Software development requirements</td><td colspan="3">Development process requirements</td></tr>
<tr><td colspan="3">Development organisation requirements</td></tr>
<tr><td>Other system requirements</td><td colspan="4">Include for example requirements for hardware, businesses, business processes, and end users</td></tr>
</table>

Figure 7. System requirements categorisation

it is not exclude that quality requirements evolve from one version to the next.

Requirements for quality requirements

Software is usually a part of a larger system. Architectural decisions made at a higher level of system hierarchical structure define boundaries and interfaces to the software. Architectural decisions about which parts of a system will actually be implemented in software may not or only partly be made at the time when qua-lity requirements are specified. Therefore, it may be impossible to decide at an early time in the development process whether or not a quality requirement is concerned with the software.

Software quality requirements may be a part of a contractual agreement between acquirer and developer or may be input for a product quality evaluation. No specific development model or measures are prescribed or assumed.

System consides for stakeholder requirements for software quality

The intended purpose of the software shall be documented. The description should take a system perspective when the software is part of a larger system.

The system solution constraints on the stakeholder software quality requirements shall be documented. The rationale and sources for system solution constraints should be documented as far as possible.

All identified stakeholders shall be listed. The identified stakeholders roles and interests shall be documented. It shall be documented whether or not a stakeholder is taken into consideration when identifying stakeholder software quality requirements. If a stakeholder is not taken into consideration, the rationale should be documented.

A commercially successful software product may have excluded specific stakeholder requirements from its requirements. For example, it may be decided to fulfil requirements of decision makers and to neglect some end user requirements. It is important to be conscious about such decisions. A consequence is that some stakeholders may not be satisfied with the software product.

Each stakeholder software quality requirement shall be uniquely identified. Each stakeholder software quality requirement shall be traceable to the individual stakeholders or classes of stakeholders. Stakeholder software quality requirements may be

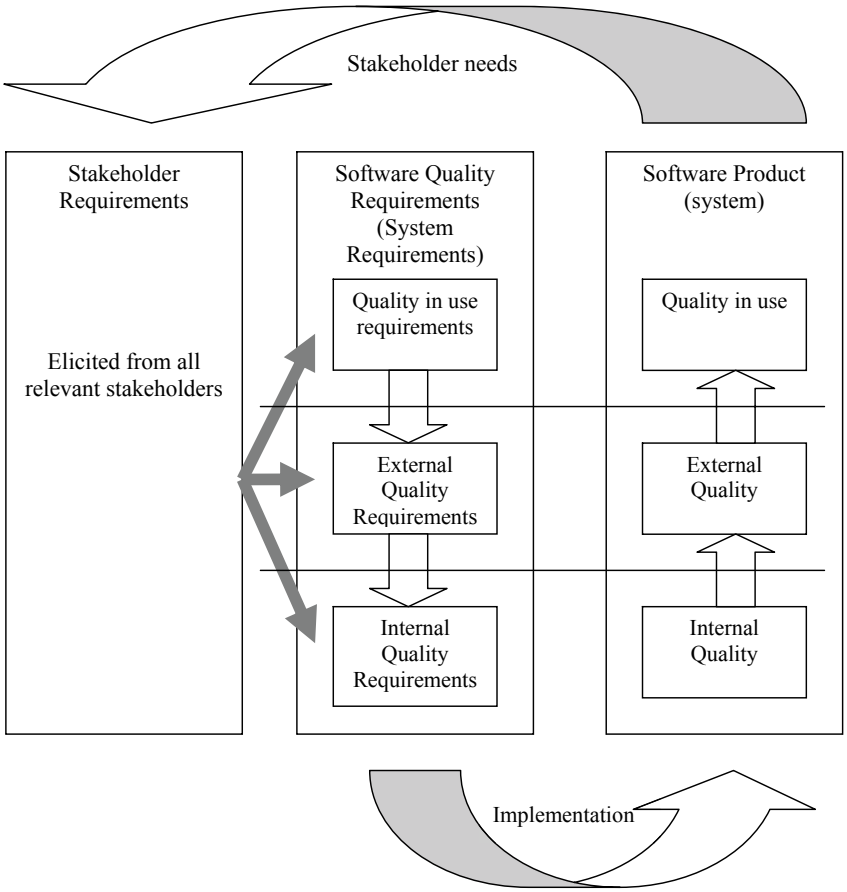

Figure 8. Quality requirements life cycle

documented in any informal, semiformal or formal notation. Stakeholder requirements are often expressed in plain text (informal notation) and concentrate on the system purpose and behaviour and are described in the context of the expected operational environment and the conditions of use. Specific concepts and terms used should be defined or explained, in order to avoid misunderstandings of the quality requirements.

Scenarios and user interactions can be helpful to identify stakeholder software quality requirements. Scenarios should include a representative set of activity sequences to identify all required services that correspond to the anticipated operational and support scenarios and environments. Also the interaction between users and the system should be identified. If possible, stakeholder software quality requirements should be traceable to scenarios and user interaction descriptions.

Verification and validation considerations for stakeholder requirements for software quality

Stakeholder software quality requirements shall be prioritised. The quality model can be used as a checklist to ensure the coverage of all quality aspects.

Incomplete, ambiguous, and unverifiable stakeholder software quality requirements shall be identified. The identified problems shall be documented. Conflicts, inconsistencies, and incongruence between the stakeholder software quality requirements shall be identified. The identified issues shall be documented. Additional stakeholder requirements solving the identified problems with stakeholder software quality requirements shall be documented according to requirements for stakeholder software quality requirements. Additional stakeholder requirements shall be traceable to original stakeholder software quality requirements. Stakeholder requirements that have been replaced shall be marked as such. Stakeholder software quality requirements should be validated. If the validation process identifies new stakeholder requirements or modifies the existing requirements, this shall be stated and the reason shall be documented.

System and quality model considerations for technical requirements for software quality

The functional boundaries of the software in terms of functional behaviour and properties to be provided shall be documented. The documentation should include usage, the benefits, system specified lifetime, system criticality and risk dependencies like safety or hazard issues. The documentation should also include architectural decisions that are allocated from design at higher levels in the structure of the system about which functions will be implemented as part of the software. Implementation constraints relevant for the technical requirements for software quality shall be documented.

Technical software quality requirements shall be categorised according to the quality model as one of the following:
– Quality in use requirement
– External quality requirement
– Internal quality requirement

A technical software quality requirement shall be specified in terms of a measurement function and associated target value. Measurement functions shall be defined according to requirements in ISO/IEC 25020. It is recommended to develop and maintain a list of quality measures for use. Applying the same set of measures for different projects makes it possible to create an experience base as described in ISO/IEC 14598-3 (ISO/IEC 25042). This approach makes estimation more reliable. Alternatively the GQM (Goal-Question-Metric) or similar approaches may be applied to identify appropriate measures for specific situations. The target values for the measure are the values, which are acceptable for fulfilling the quality requirement. The target values may be a single value or a range of values.

Specific concepts and terms used should be defined or explained, in order to avoid misunderstandings of the technical software quality requirements. The operational profile for a technical software quality requirement shall be specified when relevant. The operational profile provides information about the workload and types of transactions under which the software must be able to maintain a specified level of service. Different operational profiles may result in different measurement results and hence without a specification of the operational profile, the requirement is not uniquely specified.

Verification and validation considerations for technical requirements for software quality

Conflicts between technical software quality requirements shall be identified and documented.

In order to understand whether two or more quality requirements are conflicting, a deep understanding

of the underlying quality model is often mandatory. Practical experiences indicate that for example a high reliability requirement, a high maintainability requirement and a high efficiency requirement may be difficult to fulfil simultaneously. However, the ISO/IEC 9126-1 quality model provides no information suggesting such relationships (conflict). As long as there are no formal reasons to claim requirements to be conflicting, there is nothing preventing an evaluation or certification of these requirements. However, it may be extremely difficult to develop the software according to such requirements. Requirements may be consistent from the theoretical point of view, but almost impossible to fulfil in practise. What is possible depends largely on the abilities of the software developers, and also on the methods, tools and techniques utilised.

Conflicts between technical stakeholder requirements and implementation constraints shall be identified and documented. Additional technical requirements solving identified problems with technical software quality requirements shall be documented according to requirements for technical software quality requirements. Traceability to original technical software quality requirements shall be ensured. If implementation constraints are changed, this shall be documented. Traceability to original implementation constraint shall be ensured. The technical software quality requirements shall be reviewed and approved. It shall be stated who has reviewed and approved the technical software quality requirements.

CONCLUSION

The key moment from the author point of view is the concrete quality attributes and concrete measures. Without an agreement which attributes are essential and how to measure these attributes, all quality framework will be only an empty box, without the utilization possibility in the concrete situation. The problem of attributes and measures is the general problem of software product quality. From the authors point of view, the possible accomplishment of the SQuaRE project depends strongly on the ability of the software standardization community to share the experience in the software quality measures used by the vanward softwarehouses and software quality experts and on the agreement which measures will be selected for the quality requirement formulation, the consecutive software quality measurement and quality evaluation and rating.

REFERENCES

Vaníček J. (2005): Software and Data Quality. In: Proceedings conference Agricultural perspectives XIV. Czech University of Agriculture in Prague, September 20–21.

ISO/IEC 9126-1:2001, Software Engineering – Product quality – Part 1: Quality model.

Other related resources are referenced on the paper: Vaníček J.: Software and Data Quality, published on issue 3/2006 of this journal pp. 138–146.

Arrived on 1st February 2006

Contact address:

Jiří Vaníček, Czech University of Agriculture Prague, Kamýcká 129, 165 21 Prague 6-Suchdol, Czech Republic
tel.: +420 224 382 362, e-mail: vanicek@pef.czu.cz