Journal of Software Engineering and Applications, 2016, 9, 112-127
Published Online April 2016 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2016.94010

The Analysis and Proposed Modifications to ISO/IEC 25030—Software Engineering—Software Quality Requirements and Evaluation—Quality Requirements

Karen Mou Kui, Khaled Ben Ali, Witold Suryn

École de Technologie Supérieure, Université du Québec, Montréal, Canada
Email: {karen.mou-kui, khaled.ben-ali}.1@ens.etsmtl.ca, witold.suryn@etsmtl.ca

Received 23 February 2016; accepted 19 April 2016; published 22 April 2016

Abstract

The quality of the software product is a crucial factor that contributes to its success. Therefore, it is important to specify the right software quality requirements that will establish the basis for desired quality of the final system/software product. There are several known methodologies/processes that support the specification of the system/software functional requirements starting from the user needs to finally obtain the system requirements that the developers can implement through their development process. System/software quality requirements are interdependent with functional requirements, which means that the system/software quality requirements are meant to be specified in parallel with the latter. The ISO/IEC 25000 [1] SQuaRE series of standards include the standard ISO/IEC 25030—Software engineering—Software Quality Requirements and Evaluation—Quality requirements [2], which has as main goal to help specify software quality requirements. As to date, this standard does not offer clear and concise steps that a software quality engineer could follow in order to specify them. This article presents modifications recommended for ISO/IEC 25030 standard, with, among the others, a new requirements definition process that allows for specifying the system/software quality requirements taking into account the existing published system and software quality model ISO/IEC 25010 [3] as well as all the stakeholders of the project.

Keywords

System/Software Quality, System/Software Quality Requirements, Software Quality Engineer, Specification Process, ISO/IEC 25030, ISO/IEC 25000 SQuaRE

How to cite this paper: Mou Kui, K., Ben Ali, K. and Suryn, W. (2016) The Analysis and Proposed Modifications to ISO/IEC 25030—Software Engineering—Software Quality Requirements and Evaluation—Quality Requirements. *Journal of Software Engineering and Applications*, **9**, 112-127. http://dx.doi.org/10.4236/jsea.2016.94010

1. Introduction

Usually the popularity of a system or software is characterized by the set of functionalities it offers and its quality attributes such as, for example, performance, level of security or usability. In order to achieve this goal, in each step of the life cycle of a system or a software product the quality should be considered and actively implemented. To help IT industry in this effort the ISO/IEC Joint Technical Committee 1 (JTC1) Subcommittee 7 (SC7) has developed the series of quality-dedicated standards known as ISO/IEC 25000 SQuaRE [1], with ISO/IEC 25030 [2]—Software engineering—Software product Quality Requirements and Evaluation (SquaRE)—Quality requirements in a prominent place.

However, as ISO/IEC 25030, which should describe the process helping to specify the system/software quality requirements, in its actual form is not exhaustive enough to fulfill its basic objective of applicability in real projects, it would benefit from some modifications.

In this paper the authors analyze the actual content of the standard, propose modifications and identify several possible improvements, which, when applied, would render the standard more useful to the IT industry.

The rest of the paper is organized as follows: Section 2 describes the methodology of the presented research. Section 3 discusses the proposed modifications to the new version of the standard and particularly the new process to specify the software quality requirements. Section 4 presents the conclusion and outlines the possible continuation of this research. Finally, Section 5 presents the table of contents of the new standard.

2. Methodology

In its actual apprach to quality, the industry recognizes its importance in developing software products, thus, the quality must be present throughout the life cycle of a software product. That is the reason for the development of SQuaRE standards-*System and Software Quality Requirements and Evaluation* (*SQuaRE*).

The standards ISO/IEC 25010—System and software quality models [3] and ISO/IEC 25020—Measurement reference model and guide [4] are included in this series. ISO/IEC 25010 presents the software quality models containing the characteristics and sub-characteristics for software quality in use and software product quality. It provides the description of these models:

1) A quality in use model composed of five characteristics (some of which are further subdivided into subcharacteristics) that relate to the outcome of interaction when a product is used in a particular context of use. This system model is applicable to the complete human-computer system, including both computer systems in use and software products in use.

2) A product quality model composed of eight characteristics (which are further subdivided into subcharacteristics) that relate to static properties of software and dynamic properties of the computer system. The model is applicable to both computer systems and software products.

The standard ISO/IEC 25030 uses ISO/IEC 25010 to define the software quality requirements.

The standard ISO/IEC 25020 provides a reference model for the measures and a guide to use them with the characteristics defined in the ISO/IEC 25010 quality model. Indeed, it presents introductory explanation and a reference model that is common to quality measure elements, measures of software product quality and quality in use. It also provides guidance to users for selecting or developing, and applying measures and it contains informative annexes addressing the following topics: criteria for selecting software quality measures and quality measure elements, demonstrating predictive validity and assessing measurement reliability, and an example format for documenting software quality measures.

The standard ISO/IEC 25030 uses the standard ISO/IEC 25020 to define which mesures should be adopted for each characteristic and subcharacteristic identified in the standard ISO/IEC 25010 in otder to specify the software quality requirements.

The main objective of this research project was to analyze the ISO/IEC 25030 standard and propose the modifications to make it more explicit by providing clear, easy to follow steps helping to define the system or software quality requirements. The additional objective was to ensure that the new version of the standard can be understood by the different stakeholders of a software development process and that it fits properly within the ISO 25000 SQuaRE series of standards.

The standard ISO/IEC 25030 is the only ISO standard dedicated to specifying the system/software quality requirements, however, in its actual version the standard does not fulfill its main objective, as there is no process or method available to the readers allowing them to effectively identify and define quality requirements in real

projects.

The methodology used in this research is built of the following steps:

1) First round of analysis based only on the standard ISO 25030:

a) Identification of the elements to be preserved, reformulated, added or removed.

b) Provide for each of the elements found a justification and an explanation of the change to be made.

2) Second round of analysis taking into account interrelations with other standards of the 25000 series:

a) Analysis the other standards and identification the points of convergence with ISO 25030

b) Analysis of the application and applicability of these standards

c) Refine the results of the first round of ISO 25030 analysis.

3) Development of an ISO 25030-dedicated and precise process applying other relevant ISO 25000 series standards that a software engineer can use in order to specify the software quality requirements of a product.

4) Review and propose a new structure of ISO/IEC 25030 standard.

5) Propose a new version of the standard ISO/IEC 25030 document.

At the same time, the modified version of the standard should still fit into ISO 25000 SQuaRE series being also elaborated enough to be applicable in different software projects. Finally, the process of specifying system/software quality requirements should meet the needs of all involved stakeholders.

3. Proposed Modifications

This section presents the analysis of the standard and the resulting proposed modifications, where each modification is matched with the corresponding part in the current standard.

It is also important to clarify that this article discusses only the content of the modifications to relevant sections of the ISO 25030 standard, not their linguistic form (the language of ISO standards is specific and does not make a part of the presented research).

3.1. Structure of the Standard

The actual structure of the standard lacks a subclause that describes basic concepts used in quality requirements definition knowledge area and the clause dedicated to the process for identifying and defining quality requirements.

The most notable changes in the structure of the standard would be:

- The addition of several subclauses in Clause 5 “Fundamental concepts for quality requirements”. As the result this clause should provide general but complete set of concepts and/or references applicable in system/software quality requirements definition and quality measurement knowledge area. It should enable the readers unfamiliar with the subject of software quality to better understand the relationship between system/software quality and the identification and definition of quality requirements.
- The addition of a separate clause presenting the process of specifying quality requirements.

See Section 5 for the proposed structure of the modified version of the standard.

3.2. Modifications of Clause 5 Fundamental Concepts for Quality Requirements

The following sections contain the additions or modifications of the content of the clause with only some effort of keeping the specificity of ISO language and terms. If the proposed recommendations are accepted by ISO/IEC JTC1 SC7, their language will be revised by the ISO editors before implementation.

3.2.1. New Subclause in Clause 5—Software Quality Engineer

It is recommended that the standard define the role of software quality engineer and the expertise he/she should have in order to correctly conduct the specification of software quality requirements. The proposed new subclause “Software quality engineer” describes the role of the software quality engineer and its (role’s) importance in the specification of software quality requirements.

Proposed new text

The full process of creating a system or a software product requires the intervention of several different specialists, like business analysts, architects, developers or testers. The process of engineering the quality into this product requires a singular role, a software quality engineer. The role of the software quality engineer is

crucial to achieving the level of quality of the final product specified in the contract with the customer.

Considering that the software quality engineer should be effective throughout the whole product development cycle, he/she should have the expertise allowing for participation in all relevant activities of the cycle (like analysis, design, development and testing).

In order to make this task less complex, it may be useful to separate the required expertise into two areas: specifying system/software quality requirements and the implementation of these requirements.

The first area would require a specialist who should be responsible for identifying and defining the customer's needs and translating them into precise and doable system/software quality requirements. The responsibility of such a specialist would also cover negotiating quality-related parts of contracts and the analysis and design of system/software quality requirements.

The second area would cover:

- translating system/software quality requirements into engineering "to-dos", an equivalent of system/ software requirements specification and communicating them to the development team,
- cooperating with developers in order to facilitate the application of required quality-related engineering activities, and
- cooperating with testers in verifying (measuring and evaluating) the actual level of achieved quality against the earlier defined requirements.

Considering the fact that the specialist responsible for this area should be able to collaborate with the developers and the testers, it would be preferable that he/she be sufficiently familiar with development and test techniques and technologies.

3.2.2. Modifications to Subclause 5.2—Stakeholders and Stakeholder Requirements

The proposed new subclause "Categories of stakeholders" in Subclause 5.2 "Stakeholders and stakeholder requirements" explains the importance of the categorization of stakeholders in the specification of quality requirements. Such a clause would help the quality engineer identify more easily the key persons of the project and get them involved in the process of specifying quality requirements.

Proposed new text

Considering the stakeholders of a project, it may be useful to categorize them to help the quality engineer identify more easily the key persons of the project, their influences on it or their specific interests in it.

One possible categorization would be to define two big categories of stakeholders: customer and supplier.

The first category (customer) can be decomposed into three sub-categories as defined in Clause 3.6 of ISO 25010.

The second category represents the entire development team. The quality engineer should work closely with its major stakeholder subcategories in each of the stages of the product development process (like analyst, architect, developer and tester).

The reason for such additional categorization comes from the fact that even within the development team the internal subcategories of stakeholders may have different objectives and views on the developed product.

3.2.3. Modifications to Subclause 5.4—Software Quality Model

The standard shall not only explicitly state the importance of using the models defined in the standards ISO/IEC 25010 and ISO/IEC 25020 [4] in the process of system/specifying quality requirements, but, more importantly, it should demonstrate how these models can be used as part of the process.

Proposed new text

The quality of a system or software is the result of the quality of its elements and their interaction. This International Standard focus on the quality of both the software and the system. System/software quality is the capability of the product to satisfy stated and implied needs when used under specified conditions.

The system/software product quality model provided in ISO/IEC 25010 defines the characteristics and the sub-characteristics, which "cover all quality aspects of interest for most system/software products and as such can be used as a checklist for ensuring a complete coverage of quality". (ISO/IEC 25010).

The quality model defines three different views of quality:

- System/software quality in use.
- External/dynamic system/software quality.
- Internal/static system/software quality.

The quality in use view is related to the application of the system/software in its operational environment for carrying out specific tasks by specific users. External/dynamic quality provides a black box view of the system/software and addresses properties related to its execution on computer hardware and applied operating system. Internal/static quality provides a white box view of system/software and addresses its properties that typically are available during the development.

Note: See ISO/IEC 25010 [3] for more information about the quality models.

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

The quality characteristics and sub-characteristics defined in the ISO 25010 quality model should be used as the basic tool to specify system/software quality. Once the quality engineer has specified and validated with the customer all the quality requirements, he/she may produce the personalized quality model of the product.

Note: Personalized quality model contains only those quality characteristics and sub-characteristics that apply to a given system/software product for its intended context of use.

3.2.4. Modifications to Subclause 5.6—Software Quality Measurement Model

The small modification to subclause 5.6 has as an objective to indicate to the user the basic source of measurement-related information within ISO 25000 series.

Proposed new text

The standard ISO/IEC 25020 represents the general guide of the measurement models associated to the quality models defined in the standard ISO/IEC 25010.

Note: See ISO/IEC 25020 for more information about the guide of the measurement models.

3.2.5. Modifications to Subclause 5.9—Quality Requirements Life Cycle Model

The proposed modifications to Subclause 5.9 Quality requirements life cycle model, have as an objective to explain in which phase of the life cycle of system/software development the specification of quality requirements takes place and describes the different types of quality requirements that are relevant to each phase.

Proposed new text

Quality in use requirements are typically derived from stakeholder requirements such as 1) business requirements (company policy, competitors, etc.); 2) functional requirements; and 3) application domain specific requirements.

Quality in use requirements are normally used for system/software validation (is the software fit for its intended purpose?). Typically quality in use requirements are obtained in the first phase of analysis (requirements gathering and analysis) of the life cycle of the system/software product. The specification process presented in clause "N" provides the steps to follow in order to obtain the system/software quality requirements during this phase.

Note: The process defined in clause "N" helps obtain a majority of quality in use requirements, some of the external requirements and sometimes (rarely) internal requirements.

Note 2: As the recommended changes to the original text of ISO 25030 may influence its structure, "N" indicates the number of the new clause without imposing its physical position within this structure.

External/dynamic system/software quality requirements are typically derived from a number of sources including 1) stakeholder requirements; 2) legal requirements; 3) standards and guidelines for the relevant application; 4) quality in use requirements; 5) functional requirements; 6) application domain specific requirements; and 7) security requirements, which may be derived from risk analysis. External/dynamic system/software quality requirements are used for software validation and verification (is the software built according to specifications?). External/dynamic system/software quality requirements should be completely specified in the design phase, where the architecture of the system is obtained. The collaboration with the architect allows the quality engineer to check if the external/dynamic requirements obtained to support quality in userequirements are feasible and complete. In practice, quality in use requirements do not translate semi-automatically to external/dynamic quality requirements through the shared model (like it is in case of external and internal quality) but rather through the analysis of technical, technological or budgetary constraints. Additionally, the architecture of the system obtained in this phase may demonstrate that some external/dynamic requirements are incomplete and should be reviewed with the customer. Thus, at the end of this phase, the quality engineer should obtain a comprehensive list of external/dynamic system/software quality requirements that are feasible and

complete.

Internal/static system/software quality requirements are typically derived from a number of sources including 1) external/dynamic system/software quality requirements; 2) company policy; 3) development policy and limitations; 4) best practice guidelines and 5) rarely from quality in use requirements. Internal/static system/software quality requirements are normally used for quality monitoring and control during development. These requirements are more easily obtained from external/dynamic quality requirements because they share the same quality of models (product quality model). However, they may still remain incomplete and it is only at the beginning of the implementation phase that the software quality engineer in collaboration with the developer can verify and complete the list of internal/static requirements that are feasible.

3.3. New Process to Specify Software Quality Requirements

This section presents the dedicated process of specifying quality requirements intended to be followed by the software quality engineer. Following the steps (see **Figure 1**) of this process, the quality engineer will be able to specify a complete set of quality requirements beginning with the quality needs of the stakeholders.

The process to specify software quality requirements includes 3 phases:

1) Phase 1—Define the project context.

During this phase, the software quality engineer should identify the characteristics and constraints related to the project. The domain of the project will have a considerable influence on its characteristics. For example, the

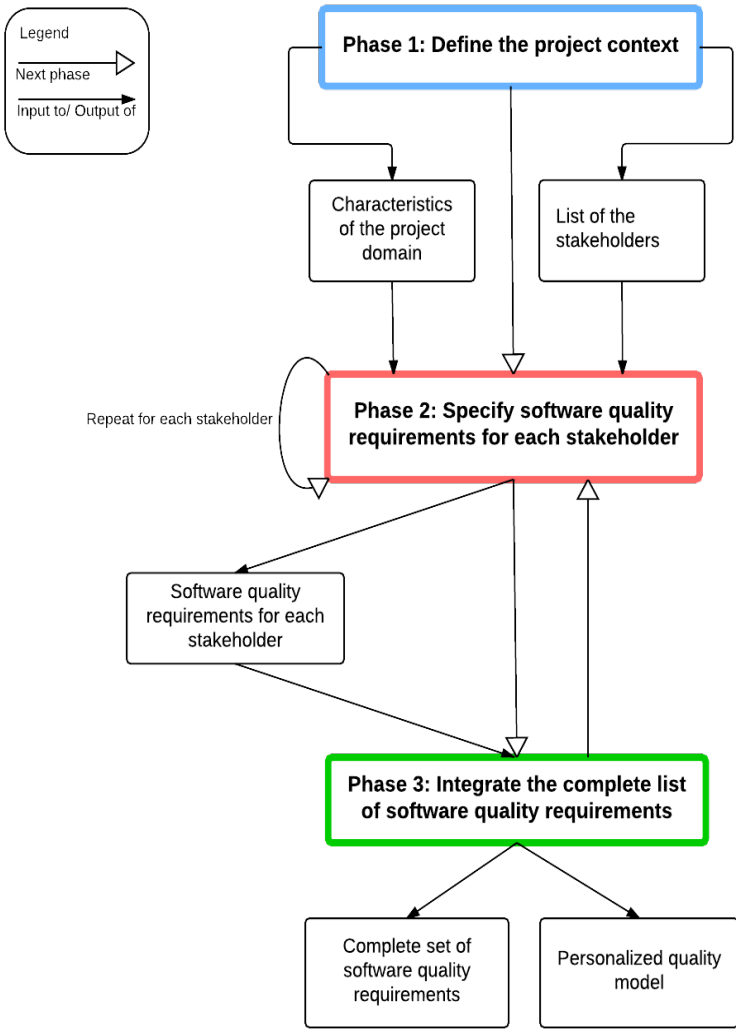

Figure 1. Process of specifying software quality requirements.

security will represent a main aspect of the project for life-critical systems. In addition, the software engineer shall identify the constraints of the project as the time or the budget allocated to it. Finally, he should identify all the stakeholders involved in the project.

2) Phase 2—Specify software quality requirements for each stakeholder.

Once the general context of the project is identified, the software engineer shall iterate with each identified stakeholder in order to specify his software quality requirements. As a result of this phase, he should be able to specify an exhaustive list of quality requirements related to each stakeholder.

3) Phase 3—Integrate the complete list of software quality requirements.

During this phase, the software engineer should resolve the conflicts that may exist among the different stakeholders' quality requirements. After that, he should set the priority of the resulting list of quality requirements. Finally, this list should be validated with the stakeholders of the project. As a final result, the software quality engineer should produce a complete set of software quality requirements and a personalized quality model.

Note: It is possible that the software engineer goes back to the previous phase in order to resolve some conflicts between quality requirements.

In the process described above and discussed more in detail further in the article, the software quality engineer should use the models and tools provided by the standards ISO/IEC 25010 and ISO/IEC 25020 in order to execute its different steps of this process and specify the software quality requirements. For each step, the input and the output are clearly specified and the characteristics of each step of this process are specified in more details.

Note: This process is proposed to be used to specify the quality in use, external and internal software quality requirements. As for the data quality requirements, the software quality engineer should check the standards ISO/IEC 25012 [5] and ISO/IEC 25024 [6].

Phase 1—Define the project context (see Figure 2)

Step 1.1 List the general assumptions of the project

Input: None.

Output: General assumptions.

The first step of this phase is to consider the following assumptions in the specific context of the project:

- The customer is a specialist in his area of business.
- The customer may not be familiar with the concepts of software quality.
- The software quality engineer is specialized in the specification of software quality requirements and may not be expert in the area of business of the customer.
- Any other necessary assumption from the perspective of software quality that is relevant to the project.

The software quality engineer should consider all the above factors before starting the process and should adapt to them in order to adequately specify the right software quality requirements. The process of specifying software quality requirements is based on the determination of the quality needs that support the business needs of the different stakeholders identified. The better the software quality engineer elicit the necessary assumptions

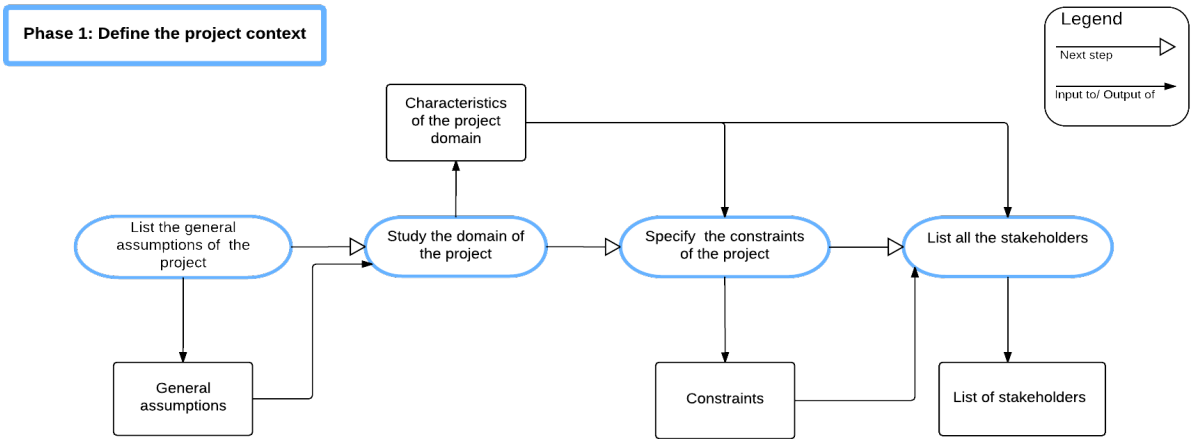

Figure 2. Process to define the project context

of the project the better he can collaborate with the customer and define the quality needs of the different stakeholders.

Step 1.2 Study the domain of the project

Input: General assumptions.

Output: Characteristics of the project domain.

This step is essentially used to obtain a better understanding of the project domain and the general context in which the software product will have to be deployed. It allows the software quality engineer to gain more expertise in the project domain and to better understand the customer's business needs. In addition, this step is essential to determine the feasibility of the various aspects of the project as it gives a clear picture of the resources (financial, capital and technology infrastructure) and skills (staff and knowledge) necessary to achieve the software quality required by the customer.

The execution of this step will allow the engineer to better understand the different aspects of the area in which the project will occur and to facilitate the communication with the client and the different stakeholders, which is essential to specify the software quality requirements.

In order to get a proper comprehension of the domain, the engineer should use the international standards that exist in that specific domain.

Step 1.3 Specify the constraints of the project

Input: Characteristics of the project domain.

Output: Constraints.

During this step, the software quality engineer should use his understanding of the project domain in order to specify the constraints of the project. This step should involve all the relevant stakeholders of the project.

This step is used to define three main types of constraints:

- Budget constraints: budget constraints will depend essentially on the financial resources allocated for the purpose of the quality of the software product.
- Technical constraints: the technical constraints are initially stated by the development team. Moreover, in the case where the customer has to perform the software maintenance after its deployment, it is important to involve his technical team in this step. In addition, the technical infrastructure of the customer's environment will necessarily impose some technical limits to the development team.
- Organizational constraints: these constraints are defined primarily by discussing with the client. He provides the information on the structure of his company and the various interactions that exist within the company.

Finally, the software quality engineer may identify other types of constraints that are relevant to a specific project with the help of the customer or the development team. The complete list of constraints is decisive to evaluate the feasibility of software quality requirements specified by the engineer. The engineer should use the data that is available from previous projects so that he does not miss a crucial point at this stage.

Step 1.4 List all the stakeholders

Input: Characteristics of the project domain, constraints.

Output: List of stakeholders.

The final step of the first phase of the process will be useful to establish the list of the stakeholders of the project. Once the software quality engineer has a better understanding of the project and its constraints, it is very important to specify all stakeholders from both sides: the customer and the supplier. In fact, if the engineer leaves any relevant stakeholder unidentified, it may negatively impact the quality of the final software product, which in turn may eventually not meet all the quality needs of the customer. The engineer should conduct interviews with the different stakeholders that he identifies to get a better idea of their involvement in the project and to ensure that he does not miss a category of stakeholders.

Phase 2—Specify software quality requirements of each stakeholder (see Figure 3)

The purpose of this phase is to specify the software quality requirements of each stakeholder identified in the previous phase. The first steps in this phase will extract the quality needs of each stakeholder. Subsequently, these needs will be used in the process of specifying software quality requirements and the measures. Finally, all the requirements of each stakeholder will be analyzed to resolve conflicts, to perform a risk analysis and to validate them with each stakeholder.

Therefore, the steps of this phase (2.1 to 2.7) shall be performed individually for each stakeholder determined in the previous phase of the process.

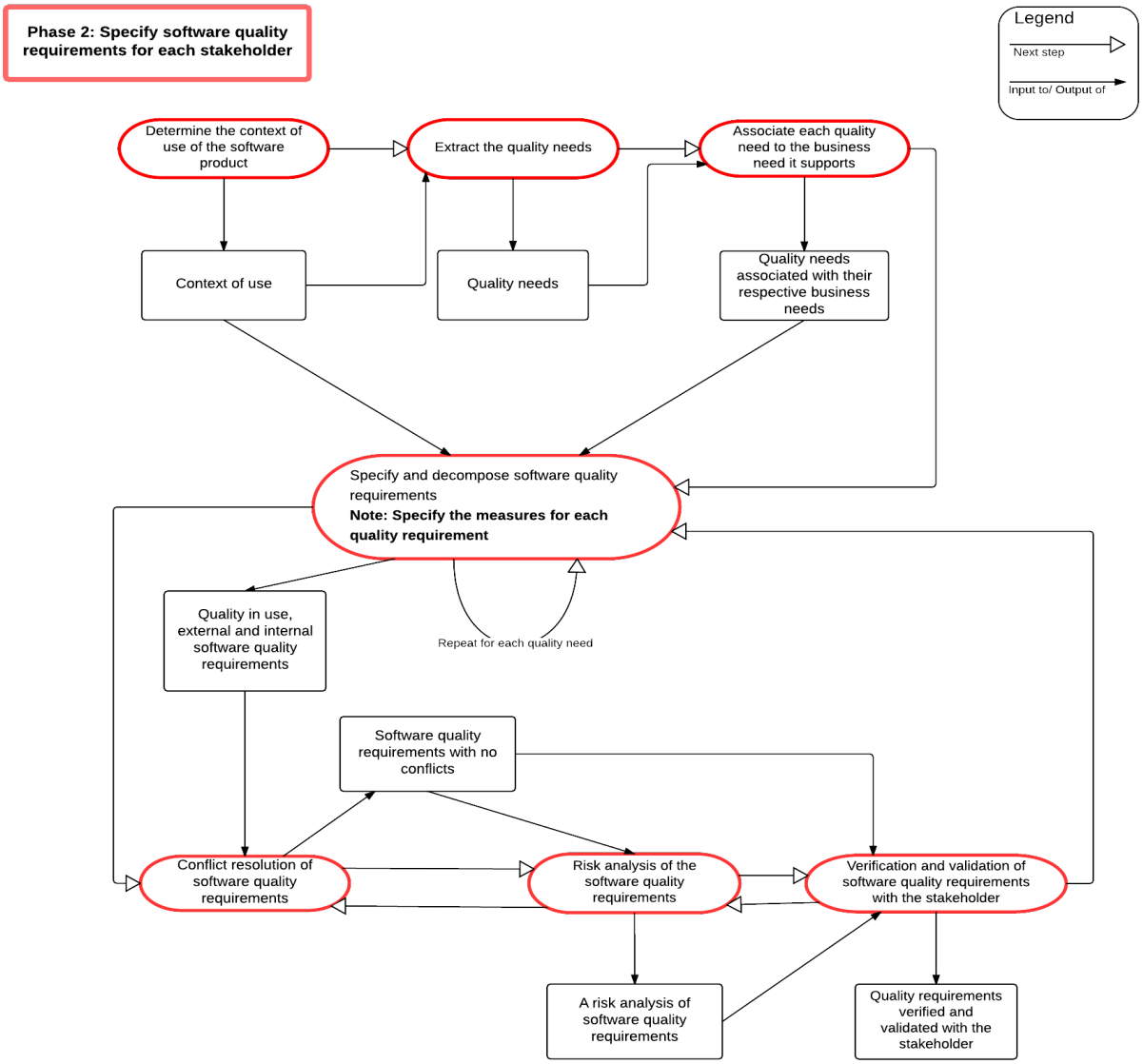

Figure 3. Process to specify software quality requirements for each stakeholder.

Step 2.1 Determine the context of use of the software product

Input: List of stakeholders, Characteristics of the project domain.

Output: Context of use.

This first step should be executed in collaboration with the stakeholder. To properly execute this step, the software quality engineer should determine:

- The goal that the user wants to achieve through the use of this software.
- The tasks that the user will perform in order to achieve his goal.
- The environment of use: technical, physical and organizational.

The context of use of the stakeholder can help the software quality engineer to determine his quality needs later in the process. In order to define the context of use, the software quality engineer may use different techniques that allow him to obtain this information, for example:

- survey;
- observation;
- interview;
- any other required tools.

It is possible to combine several techniques in order to obtain more information and have a stronger basis for

specifying software quality requirements.

Finally, the software quality engineer may use standard ISO/IEC 25063 [7] to document the context of use of each stakeholder to ensure the traceability and to adopt a common industrial format.

Step 2.2 Extract quality needs

Input: Context of use.

Output: Quality needs.

The software quality engineer should use the information obtained from the context of use for each stakeholder in order to identify his quality needs. The stakeholder's context of use should provide a clear idea of his software quality needs and also limit the scope of these needs. Therefore, in collaboration with the stakeholder, the engineer should list all his quality needs and validate them with the stakeholder to ensure that they represent the real needs.

The engineer shall document all the software quality needs in a format that allows him to easily trace them to the different stakeholders.

Step 2.3 Associate each quality need to the business need it supports

Input: Quality needs.

Output: Quality needs associated with their respective business needs.

This step is not necessarily decisive for the conduct of the rest of the process but it is fundamental in order to justify the software quality requirements that will be identified. A direct relationship between each quality need and its business need allows the software quality engineer to justify its relative importance. Once this step is completed, the software quality engineer should be able to initiate the next step, which will allow him to specify the software quality requirements for each stakeholder using the earlier defined software quality needs.

Step 2.4 Specify and decompose software quality requirements

Input: Quality needs associated with their respective business needs, Context of use.

Output: Quality in use, external and internal software quality requirements.

This part of the process is in itself a sub-process that will allow the software quality engineer to specify software quality requirements based on software quality needs of the stakeholder, starting with the quality in use requirements. In addition, the engineer should specify the measures for each identified software quality requirement (**Figure 4**).

Note: This process is executed for each quality need of the stakeholder

Step 2.4.1 Define quality in use characteristics

Input: Quality need, Context of use, Quality in use model ISO/IEC 25010.

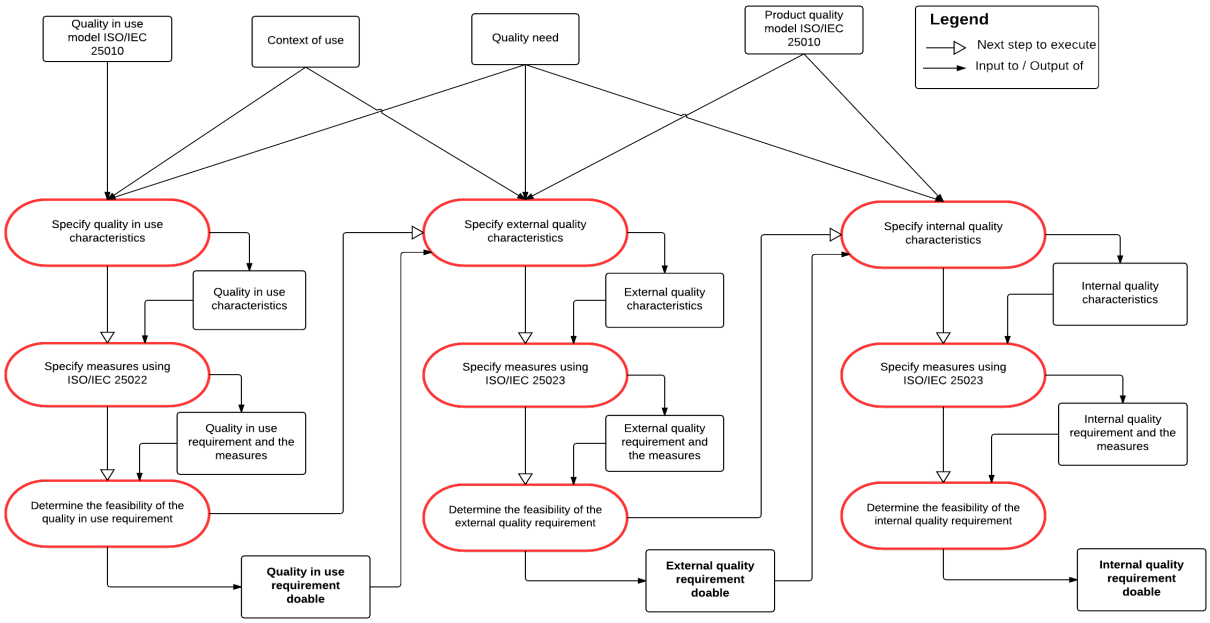

Figure 4. Process to specify and decompose software quality requirements.

Output: Quality in use characteristics.

During this step, the software quality engineer should consider the identified software quality need and the context of use of the stakeholder. In this step the engineer should use the quality in use model defined in the standard ISO/IEC 25010 to identify the characteristics that can meet the quality need of the stakeholder.

These quality characteristics represent the basis to specify the measures that will be used. The standard ISO/IEC 25010 provides some recommendations and examples of the use of the quality model.

Step 2.4.2 Specify quality in use measures using ISO/IEC 25022 [8]

Input: Quality in use characteristics.

Output: Quality in use requirements and the measures.

To perform this step, the software quality engineer should consider all the quality characteristics that he identified in the previous step. Each characteristic is built upon measure(s) defined in the standard ISO/IEC 25022 and the engineer should review all the characteristics and assign the measures that correspond to them.

As indicated in the standard ISO/IEC 25040 [9], the software quality engineer may specify the criteria for validating the software quality requirement if he has in his possession the necessary information to do so.

Step 2.4.3 Determine the feasibility of the quality in use requirements

Input: Quality in use requirement and the measures.

Output: Quality in use requirement doable.

This step allows for checking whether the quality in use requirement is feasible or not. The feasibility should be determined in collaboration with the technical team of the provider because the software quality engineer may not have all the necessary knowledge to make such a decision. The quality engineer should consider the advice of the development team in order to make his decision on the feasibility of the software quality requirement. Every feasible requirement should be used in the next step.

Step 2.4.4 Define external/dynamic quality characteristics

Input: Quality need, Context of use, Product quality model ISO/IEC 25010, feasible Quality in use requirements.

Output: External quality characteristics.

During this step, the software quality engineer should consider the software quality needs, the context of use of the stakeholder and the quality in use requirements he specified in the previous step. The engineer should use the product quality model defined in the standard ISO/IEC 25010 to identify the external (or *dynamic*) characteristics that meet the quality needs of the stakeholder. In addition, the engineer should verify that these characteristics allow achieving the quality in use characteristic specified earlier.

These external quality characteristics are the basis to specify the measures that will be used for quality verification purposes. The standard ISO/IEC 25010 provides some recommendations and examples of the use of the quality model. The engineer should be able to specify some of the external quality characteristics at this phase. The rest of the characteristics could be identified during the design of the software product.

Step 2.4.5 Specify external/dynamic quality measures using ISO/IEC 25023

Input: External quality characteristics

Output: External quality requirement and the measures

To perform this step, the software quality engineer should consider all the external quality characteristics that he identified in the previous step. Each characteristic corresponds to set of measures defined in the standard ISO/IEC 25023 [10]. The quality engineer should review all the characteristics and assign the measures that correspond to them.

As indicated in the standard ISO/IEC 25040, the software quality engineer may specify the criteria for validating the software quality requirement if he has in his possession the necessary information to do so.

Step 2.4.6 Determine the feasibility of the external/dynamic quality requirements

Input: External quality requirement

Output: External quality requirement feasible

This step allows the quality engineer to check whether the external quality requirement is feasible or not. The feasibility should be determined in collaboration with the technical team of the provider because the software quality engineer may not have all the necessary knowledge to make such a decision. The quality engineer should consider the advice of the development team in order to make his decision on the feasibility of the software quality requirement. Every feasible requirement should be used in the next step.

Step 2.4.7 Define internal/static quality characteristics

Input: Quality model 25010, External quality requirement associated with the measures

Output: Internal quality requirement

During this step, the software quality engineer should consider the external quality requirement he specified in the previous step. The quality engineer should use the product quality model defined in the standard ISO/IEC 25010 to identify the internal (or *static*) characteristics that allow achieving the external quality characteristics specified earlier.

These internal quality characteristics are the basis to specify the measures that will be used. The software quality engineer should consider all the internal quality characteristics of the model. The standard ISO/IEC 25010 provides some recommendations and examples of the use of the quality model.

The engineer should be able to specify a few of the internal quality characteristics in this step. He should be able to specify them completely during the phase of the construction of the software product.

Step 2.4.8 Specify internal/static quality measures using ISO/IEC 25023

Input: Internal quality characteristics

Output: Internal quality requirement and the measures

In order to perform this step, the software quality engineer should consider all the internal/static quality characteristics that he identified in the previous step. Each characteristic corresponds to a set of measures defined in the standard ISO/IEC 25023. The engineer should review all the characteristics and assign the measures that correspond to them.

As indicated in the standard ISO/IEC 25040, the software quality engineer may specify the criteria for validating the software quality requirement if he has in his possession the necessary information to do so.

Step 2.4.9 Determine the feasibility of the internal quality requirement

Input: Internal quality requirement and the measures

Output: Internal quality requirement doable

This step allows the quality engineer to check whether the internal quality requirement is feasible or not. The feasibility should be determined in collaboration with the technical team of the provider because the software quality engineer may not have all the necessary knowledge to make such a decision. The quality engineer should consider the advice of the development team in order to make his decision on the feasibility of the software quality requirement.

Step 2.5 Requirements conflicts resolution

Input: Quality in use, external and internal software quality requirements

Output: Software quality requirements with no conflicts

Once the software quality engineer has finished specifying the software quality requirements of a given stakeholder, he should resolve the conflicts that may exist among these different requirements. For example, there may be a requirement that specifies some degree of performance that is in conflict with a requirement of reliability.

In such cases the engineer should work closely with the stakeholder to find the resolution of the conflict. The engineer should also consider the conflicts that may exist among the requirements at the technical level. In those cases, to find the resolution of the conflict the engineer should collaborate with the development team.

Once this step is completed, the software quality engineer should produce a list of software quality requirements without conflicts for a given stakeholder. This step should be repeated for every identified stakeholder.

Step 2.6 Risk analysis of the software quality requirements

Input: Software quality requirements with no conflicts

Output: A risk analysis of every software quality requirement

For this step, the software quality engineer should perform a risk analysis of each software quality requirement. This task requires close co-operation with the stakeholder in order to identify business risks that are specific to each requirement of software quality. The objective is to determine the possible consequences if a given quality requirement is ignored or, what will be the cost of *missing* quality.

In addition, the engineer should collaborate with the supplier to specify the technical risks that are specific to all previously identified software quality requirements.

This risk analysis will be essential to assign the priorities to the requirements later in the process.

Note: It is possible to return to the previous step 2.5 to resolve some conflicts between software quality requirements.

Step 2.7 Verification and validation of software quality requirements with the stakeholder

Input: Software quality requirements with no conflicts, a risk analysis of every software quality requirement
Output: Quality requirements verified and validated with the stakeholder

This is the final step of this phase that provides a list of software quality requirements verified and validated by the stakeholder. At the end of this process, the software quality engineer should produce an accepted list of software quality requirements that meet the identified needs of the stakeholder (**Figure 5**).

Note: It is possible to return to the previous step (2.6) to review the risk analysis of some requirements. It is also possible to go back to step 2.4 to review a given software quality requirement in particular.

Phase 3—Integrate the complete list of software quality requirements

This phase is the last of the proposed new process and is performed once the software quality engineer has finished specifying all the software quality requirements of all stakeholders. This phase should help:

- globally resolve the conflicts among all the software quality requirements;
- globally prioritize the full set of identified software quality requirements;
- globally verify and validate the complete list of software quality requirements with all the stakeholders;
- produce the personalized software quality model of the project.

Step 3.1 Conflict resolution of all software quality requirements

Input: The complete set of software quality requirements
Output: Software quality requirements with no conflicts

During this step, the software quality engineer should consider all the software quality requirements of all the stakeholders he identified during the process. In order to complete this step, the engineer should work closely with the customer and the development team. To solve the possible conflicts the engineer may need the help from the customer who ultimately can decide what are the most important requirements for him, which may eliminate a significant number of conflicts. Besides that, other conflicts may affect the technical feasibility of the software quality requirements, what may require the collaboration with the software supplier team in order to find the right solutions to these conflicts.

Step 3.2 Set the priority of each software quality requirement

Input: Software quality requirements with no conflicts
Output: Software quality requirements with their respective priorities

In this step the software quality engineer should assign a priority to each software quality requirement that he specified. These priorities will be crucial for the development team in course of the project, to, for instance, adjust the implementation plan of the various requirements or correctly allocate the necessary resources to achieve their goals.

The software quality engineer should work with the customer and the development team to prioritize software quality requirements what would be helpful to reduce and control the risks that could affect the quality of the final software product.

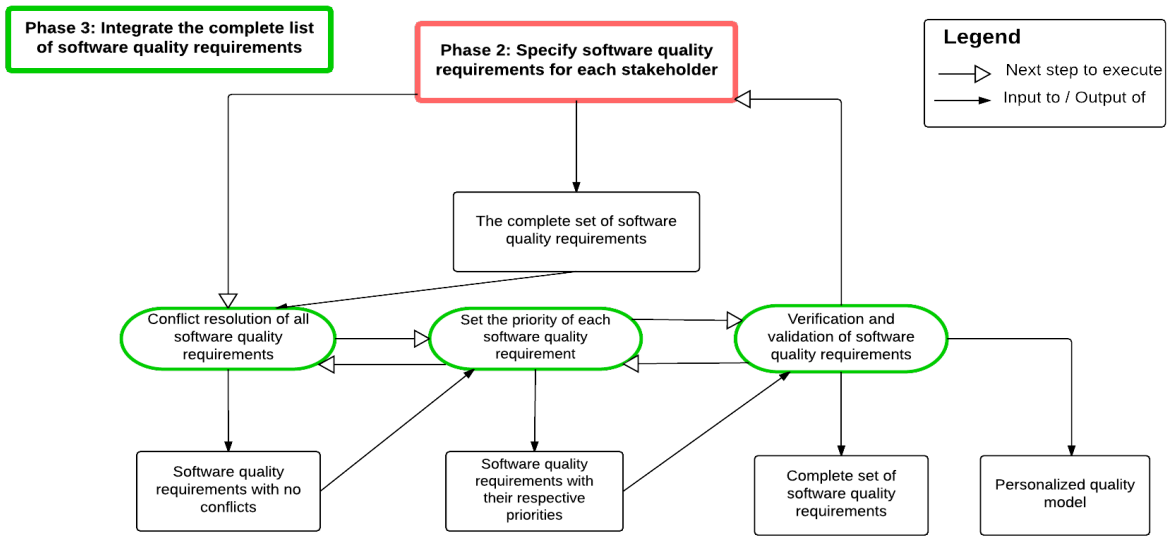

Figure 5. Process to integrate the complete list of software quality requirements.

Note: It is possible to return to the previous step (3.1) to resolve some conflicts if the software quality engineer thinks that this is needed.

Step 3.3 Verification and validation of software quality requirements

Input: Software quality requirements with their respective priorities

Output: Final set of software quality requirements + Personalized quality model

This is the final step of the process. It allows the software quality engineer to verify and validate the final set of software quality requirements specified in the process. At the end of this process, the software quality engineer should produce a list of software quality requirements that meet all identified quality needs of the different stakeholders he has identified. He should also be able to validate these requirements with all stakeholders and with the customer in particular. This validation should review all of the requirements also considering the priority that has been assigned to each requirement.

Note: Refer to the section 6.2 on the life cycle that explains the evolution of software quality requirements.

The software quality engineer may use the list of software quality requirements validated with the customer to produce the project-personalized quality model of the project. This model contains only characteristics, subcharacteristics and measures applicable in the given project and allows the engineer to quickly verify the characteristics that should be present in the final product of the project. This model may evolve during the project as the result of the evolution of software quality requirements.

Note: It is possible to return to the previous step (3.2) to review the priorities of the software quality requirements. It is also possible to return to the Phase 2 to review some software quality requirements.

4. Conclusion and the Future Work

In this paper, the authors analyzed the actual version of the standard ISO/IEC 25030 in order to propose modifications to the new version (actually under revision in ISO/IEC JTC1 SC7) of the standard that should make it more applicable in the context of the IT industry. These modifications, if accepted by ISO/IEC JTC1 SC7, could allow for restructuring of the standard so it becomes understandable by the majority of the possible stakeholders involved in the development cycle of a system/software product.

This revision aims at enhancing quality requirements division of SQuaRE series aligned with the other divisions, and providing a more practical guide for definition and use of quality requirements. Major proposed modifications are:

- Extension of the view from software to system,
- Enhancement and deployment of quality requirements, and
- Clarification of quality requirements definition steps:
  - Stating them exhaustively by using the quality models as a checklist,
  - Specifying them with the quality measures with criteria for evaluation.

The main proposed addition to the new version of the standard is the dedicated process for specifying the system/software quality requirements and the details describing the steps of each phase of this process.

For future research, the reader could analyze the process for specifying software quality requirements as proposed by the authors. In addition, it would be interesting to consider using the standard ISO/IEC 25012 that describes a model for data quality. Indeed, the quality of data used in a system/software is crucial to its operation. Besides that, the standard ISO/IEC 25024 provides a list of measures that can be used with the data quality model.

Finally, once the process proposed by the authors is analyzed and improved it should be applied in different projects to prove its reliability and that it is useful to specify the right software quality requirements correctly.

5. Proposed Structure of the Modified Version of ISO/IEC25030

The proposed changes and modifications to the table of content are underlined:

Foreword

Introduction

1 Scope

2 Conformance

3 Normative references

4 Terms and definitions

5 Fundamental concepts for quality requirements
5.1 Software quality engineer
5.2 Software and system
5.3 Stakeholders and stakeholder requirements
5.3.1 Categories of stakeholders
5.3.2 Stakeholder requirements
5.4 Stakeholder requirements and system requirements
5.5 Software quality model
5.6 Software properties
5.7 Software quality measurement model
5.8 Software quality requirement
5.9 System requirements categorization
5.10 Quality requirements life cycle model
6 Process of specifying software quality requirements
6.1 Defining the project context
6.1.1 List the general assumptions of the project
6.1.2 Study the domain of the project
6.1.3 Specify the constraints of the project
6.1.4 List all the stakeholders
6.2 Specifying quality requirements of each stakeholder
6.2.1 Determine the context of use of the software product
6.2.2 Extract quality needs
6.2.3 Associate each quality need to the business need it supports
6.2.4 Specify and decompose software quality requirements
6.2.5 Conflict resolution for requirements
6.2.6 Risk analysis of the software quality requirements
6.2.7 Verification and validation of software quality requirements with the stakeholder
6.3 Complete specification of software quality requirements
6.3.1 Conflict resolution of all software quality requirements
6.3.2 Set the priority of each software quality requirement
6.3.3 Verification and validation of software quality requirements
7 Requirements for quality requirements
7.1 General requirements and assumptions
7.2 Stakeholder requirements
7.2.1 System boundaries
7.2.2 Stakeholder quality requirements
7.2.3 Validation of stakeholder quality requirements
7.3 Software requirements
7.3.1 Software boundaries
7.3.2 Software quality requirements
7.3.3 Verification of software quality requirements
ANNEX A: Relationship to ISO/IEC 15288 (System lifecycle process)
ANNEX B: Relationship to ISO/IEC/IEEE 29148 (Requirements engineering process)
ANNEX C: Recommended process for quality requirements

References

[1] ISO/IEC 25000:2005, Software Engineering Software product Quality Requirements and Evaluation (SQuaRE). Guide to SQuaRE.

[2] ISO/IEC 25030:2005, Software Engineering Software Quality Requirements and Evaluation (SQuaRE). Quality Requirements.

[3] ISO/IEC 25010 (New), Software Engineering: Software product Quality Requirements and Evaluation (SQuaRE) Quality Model.

[4] ISO/IEC 25020 (New) Software Engineering: Software Product Quality Requirements and Evaluation (SQuaRE)

Measurement Reference Model and Guide.

[5] ISO/IEC 25012 Software Engineering: Software Product Quality Requirements and Evaluation (SQuaRE)—Data quality model.

[6] ISO/IEC 25024 (New) Software Engineering: Software product Quality Requirements and Evaluation (SQuaRE) Measurement of quality in use.

[7] ISO/IEC 25063 Systems and Software Engineering: Systems and Software Product Quality Requirements and Evaluation (SQuaRE) Common Industry Format (CIF) for usability: Context of use description.

[8] ISO/IEC 25022 (New) Software Engineering: Software Product Quality Requirements and Evaluation (SQuaRE) Measurement of Internal Quality.

[9] ISO/IEC 25040 Systems and Software Engineering: Systems and Software Product Quality Requirements and Evaluation (SQuaRE) Evaluation Process.

[10] ISO/IEC 25023 (New) Software Engineering: Software Product Quality Requirements and Evaluation (SQuaRE) Measurement of External Quality.