Software Qual J (2010) 18:37–55
DOI 10.1007/s11219-009-9080-0

A software product certification model

Petra Heck · Martijn Klabbers · Marko van Eekelen

Published online: 27 June 2009

Abstract Certification of software artifacts offers organizations more certainty and confidence about software. Certification of software helps software sales, acquisition, and can be used to certify legislative compliance or to achieve acceptable deliverables in outsourcing. In this article, we present a software product certification model. This model has evolved from a maturity model for product quality to a more general model with which the conformance of software product artifacts to certain properties can be assessed. Such a conformance assessment we call a 'software product certificate'. The practical application of the model is demonstrated in concrete software certificates for two software product areas that are on different ends of the software product spectrum (ranging from a requirements definition to an executable). For each certificate, a concrete case study has been performed. We evaluate the use of the model for these certificates. It will be shown that the model can be used satisfactorily for quite different kinds of certificates.

Keywords Software certification · Certification levels · Formal methods

P. Heck
Software Quality Systems (SQS), Geneva, Switzerland
e-mail: petra.heck@sqs-group.ch

M. Klabbers
LaQuSo, Technical University Eindhoven, Eindhoven, The Netherlands
e-mail: martijn@laquso.com

M. van Eekelen (✉)
LaQuSo, Radboud University Nijmegen, Nijmegen, The Netherlands
e-mail: marko@cs.ru.nl

M. van Eekelen
Open University, Heerlen, The Netherlands

1 Introduction

More and more applications depend on the reliability, availability, and integrity of software systems. Due to the increase of complexity at the hardware, software, and communication level, creating quality systems has become both a major scientific and engineering challenge. Only a limited number of examples of these quality certification systems have been published, for example Nastro (1997), Alvaro et al. (2007), and Wegner (1999).

Next to proper methods for creation of quality systems, the verification of such systems is also important. Many times the failure of systems endangers human safety, so failure must be avoided. The developer of the system must verify the system before delivering it, e.g., through testing or manual review.

However, if systems are safety critical and are to be used in a broad environment, an independent third party (not the supplier or the acquirer of the system) should verify the quality of the system. A third party can produce an objective and complete judgment. A third party can even hand out certificates if the assessors use a standard way of producing the judgment.

The possible benefits of software product certification are numerous; it helps organizations to obtain certainty about or confidence in software artifacts. In software sales, a software product certificate offers an advantage over the competition, it gives more confidence for prospective buyers, like medical organizations, as stated by Forsström (1997). Certification can also help to verify and certify legislative compliance. Moreover, it can help outsourcing partners, the outsourcers as well as the subcontractor, to convince the other party that deliverables are acceptable. This could help prevent poor quality of the requirements (incomplete and changing requirements), the primary reason why so many projects continue to fail. See for example the Chaos reports of The Standish Group (1996–2006), and Boehm (2001).

In this study, we present a certification model for assessing the quality of software products. In contrast to other certification models, as described by Welzel and Hausen (1997) and strictly speaking also the one tried by Alvaro et al. (2007), we show how this model is applied and has resulted in two certificates. We do not consider hardware and network aspects of systems, because they require very different means of verification. Furthermore, we do not consider the assessment of the software development process. Assessment of the software development process is intensively studied elsewhere, like for example CMMI Product Team (2001), Kruchten (2004), and Bamford and Deibler (2004).

We believe that provable quality of software products can be based on the application of justified product analysis techniques and where possible based upon formal methods.

The remaining sections of the introduction explain what certification is in our model and give a summary of the main concepts of the model. Section 2 gives details on the model. Section 3 contains examples of predefined certificate types and our experience in using them. Section 4 refers to related work and Sect. 5 contains the conclusion.

1.1 Certification

For certification two types of input are required: (1) one or more software artifacts and (2) one or more properties of these artifacts that are to be certified. The software artifacts (1) are divided into 6 product areas that are detailed in the following chapter.

The properties (2) can be of one of the following categories:

- *Consistency*: do the different (parts of) software artifacts conform to each other?
- *Functional*: does input to the system produce the expected output?

- *Behavioral*: does the system meet general safety and progress properties like absence of deadlocks or are constraints on the specific states of the system met?
- *Quality*: do the artifacts fulfill nonfunctional requirements in the areas of for example performance, security, and usability?
- *Compliance*: do the artifacts conform to standards, guidelines, or legislation?

Properties can be general (valid independent of the software artifact involved) or dedicated.

In the following sections, we will call the above properties **Conformance Properties**. Appropriate conformance analysis techniques should be chosen depending on the application domain and input artifacts. Sometimes software artifacts need to be transformed before the analysis can take place. For instance, a requirements document must be translated into a formal model before any property of it can be proven. Similarly, source code and its properties have to be transformed into a model and a set of predicates to be able to apply theorem proving.

Software certification consists of a basic assessment of input software artifacts (on completeness and uniformity) and the before mentioned conformance analysis with the conformance properties.

In this document, we present a certification model for software products. This model describes a structured approach to software product certification.

1.2 Model concepts

As software artifacts we do not consider only the final software product (the source code or working system), but also intermediate deliverables like user requirements and detailed design. Each major class of deliverables is a **Product Area** in the model (see Sect. 2.1). We have identified different artifacts, called **Elements**, within the Product Areas, such that properties can be investigated on a more detailed level.

There are three **Certification Criteria**, which hold for all Product Areas: areas must be completely (and formally) specified, uniform, and conformant. There are four **Achievement Levels** for each Certification Criteria (see Sect. 2.2).

The Certification Criteria can be translated into **Specific Criteria** per Product Area that indicate what formal, uniform, and conformant means for that Product Area. The Specific Criteria indicate what required elements and checks are needed to achieve a certain level of the Certification Criteria (see Sect. 2.4).

When the desired achievement level, which can be derived from the desired certification level for each of the three Certification Criteria, has been established, the overall **Certification Level** of the product can be determined (see Sect. 2.3). The more formal elements are present in the product, and the more formal checks have been performed without detection of faults or nonconformance, the higher the confidence in the product certificate is.

The concepts of the model are summarized in Fig. 1. Our concepts are loosely based on CMMI (2001).

2 The LaQuSo software product certification model

Laboratory for Quality Software (LaQuSo) has developed a model for software product certification, which is called the LaQuSo Software Product Certification Model (LSPCM). The following sections describe this model.

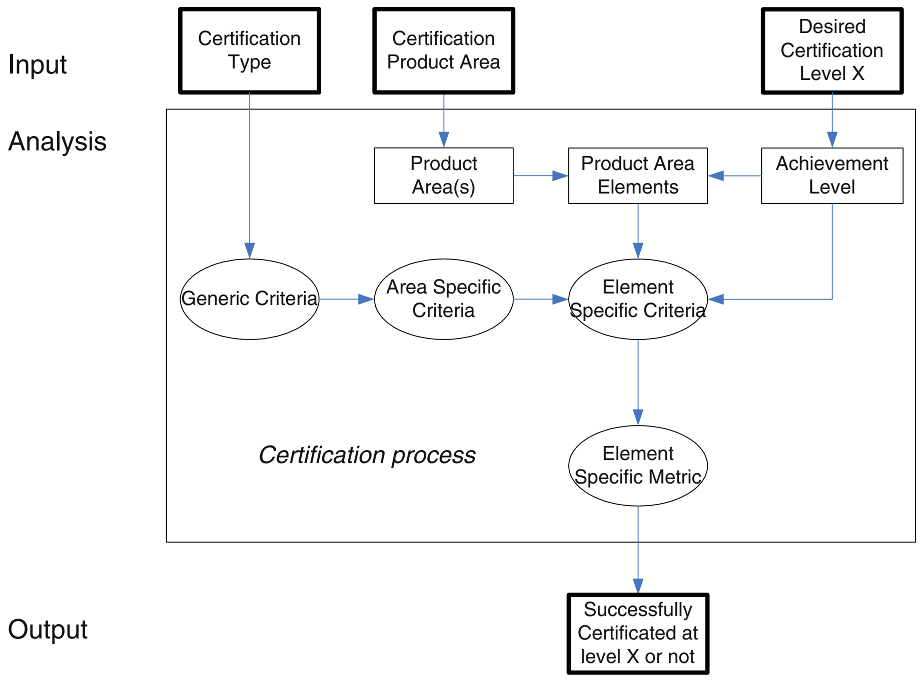

Fig. 1 Concepts of the certification model

2.1 Software product areas

For our division of the software product into Product Areas, we have taken the main deliverables of the development phases (requirements, high-level design, low-level design, implementation, and test). We have split the requirements into a context description and a user requirements part, to emphasize the importance of the analysis of the system environment.

The model consists of six Product Areas:

- The *context description* (CD) describes the environment and main processes of the system.
- The *user requirements* (UR) specify what functions the system has to fulfill.
- The *high-level design* (HD) (also called software requirements) is a translation of the user requirements into a more precise description in terms of system architects.
- The *detailed design* (DD) consists of several models of the system that describe how the system will be built.
- The *implementation* (IMP) contains the system and its documentation, built according to the design.
- The *tests* (TST) describe the tests of the different software components and the whole system.

Each area can be further divided into subparts, which we call *elements*. These elements can be separate artifacts, a chapter within a document, or different parts of a larger model. For instance, the user manual will be a separate artifact delivered with the system, the nonfunctional requirements will be a separate section of the user requirements document,

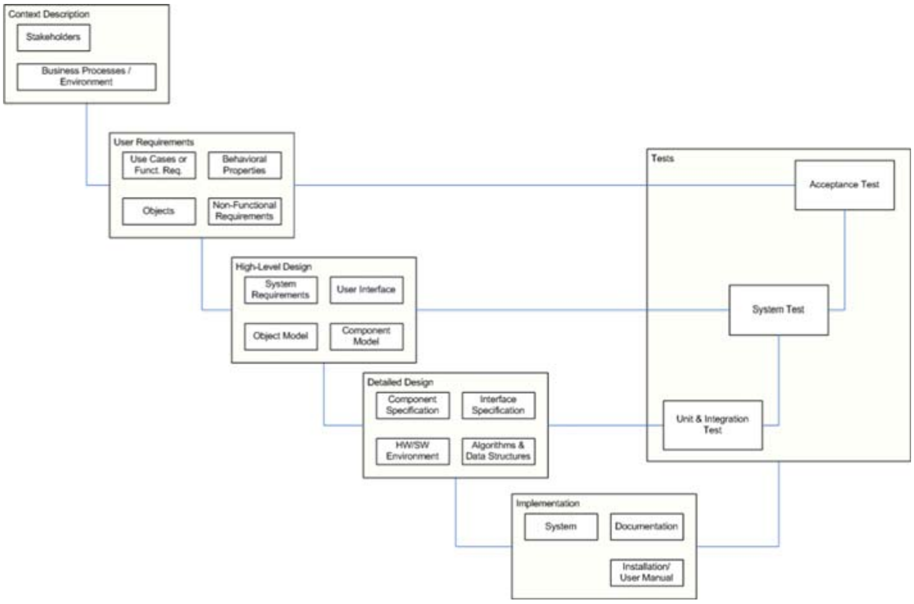

Fig. 2 Software product areas and their elements

and the stakeholders can be described as part of the business process description (for example in the same diagram).

Figure 2 shows the areas, their elements, and their interrelations. We have put the areas in the traditional V-layout. A line between two Product Areas means that elements in one area depend on a previous area in the V. For example, High-Level Design is derived from the User Requirements, the System Test tests all functionalities in the High-Level Design, and the Acceptance Test can refer to tests reported in the System Test to prevent duplication of test cases.

Note that a certification is not necessarily based on a complete Product Area. It is however, always required to classify the software artifacts that are input to the certification in the appropriate Product Area and elements. It must be determined how the artifacts correspond to the model presented in this document.

2.2 Certification criteria

Certification Criteria (CC) are criteria that apply to each Product Area. There are three Certification Criteria for all Product Areas:

[CC1] **Completeness**. All required elements in the Product Area should be present and there should be as many (derived) formal elements as possible.
[CC2] **Uniformity**. The style of the elements in the Product Area should be standardized.
[CC3] **Conformance**. All elements should conform to the property that is the subject of the certification.

For each of these Certification Criteria different *Achievement Levels* can be established, which we have summarized in Table 1.

Table 1 Certification criteria achievement levels

CC1	**Completeness**
0	Some required elements are missing
1	All required elements are present
2	Semiformal elements have been added
3	Formal elements have been added
CC2	**Uniformity**
0	No standardization
1	Within the product
2	Style complies to a company standard
3	Style complies to an industry standard
CC3	**Conformance**
0	Faults are detected
1	Manual review/testing has not detected any faults
2	Automated testing has not detected any faults
3	Formal verification has not detected any faults

The completeness of a Product Area (CC1) can be basic (all required elements are present, level 1) or extra elements may have been added. These elements can be semiformal (level 2) or formal (level 3), which refers to the fact that they are specified in a formal language. The more formal an element is, the easier it can be subjected to formal verification (less transformation is needed). For examples of semiformal and formal elements see SC1.2 and SC1.3 in Sect. 2.4.1.

The uniformity of a Product Area (CC2) can be only within the Product Area itself (level 1), with respect to a company standard (level 2), or with respect to an industry standard (level 3). By industry standard, we mean a general accepted description technique that is not specific for the company like the use of UML diagrams for design documents. If known standards are used, translations to formal methods are likely to be available, which makes formal verification easier.

The conformance of the Product Area (CC3) to a property can be established with different means that gradually increase confidence: manually (level 1), with tool support (level 2), or by formal verification (level 3).

From the levels in Table 1 and the Certification Criteria, we derive the scoring rules; one for each goal that simply indicates that the level should be as high as possible:

[CC1.1] *The prescribed elements of a certain formalization level (required, semiformal, formal) must be present*. Score 0 if any required element is missing; score 1 if any semiformal element is missing; score 2 if any formal element is missing; score 3 if all elements are present.

[CC2.1] *As much standardization as possible*. Score 0 if elements of the same type have different style (for example if all use case descriptions have a different structure); score 1 if elements of the same type have the same style; score 2 if all elements also comply with the company standard; score 3 if all elements also comply with industry standards.

[CC3.1] *Zero faults with the most thorough check possible on conformance*. Score 0 if any fault has been detected; score 1 if manual review of elements detects no faults; score 2 if tool-supported and manual review of elements detects no faults; score 3 if review of elements and formal checks detect no faults.

The Specific Criteria indicate for each Product Area what the required elements, applicable standards, and possible checks are. We will provide Specific Criteria in the Requirements Product Area in Sect. 2.4.1.

2.3 Certification levels

From the levels that have been achieved for the three Certification Criteria, an overall *Certification Level* can be calculated. This overall Certification Level represents the maturity of the software products and its artifacts.

The model indicates a certification level per Product Area. The certification level of the entire product can be determined by taking the minimum over the areas, but a Product Area-based certification is more informative. We can for example decide to only certify the Implementation Product Area if our interest is in the certification of the final product without taking into account the development deliverables or testing deliverables. We can even certify part of a Product Area, for example certifying only the communication protocol of a finished system.

The certification levels are based on an intuitive notion of when product certificates are more authorative. The highest level is achieved when a product is complete and uniform, and correctness and consistency have been verified with the most rigorous method.

The model has five certification levels, but only four are relevant for the certification process outcome. The first is only used for creating a baseline so that the certification analysis can start. For each certification level, we have indicated the level that is needed for each of the Certification Criteria (see also Table 1):

1. Initial

$CC1 \geq 1$ and $CC2 \geq 1$ and $CC3 = 0$

Each of the required elements is present in the product, and the elements are uniform. This is the level that indicates that the required elements for certification are there, and analysis can start.

2. Manually verified

$CC1 \geq 1$ and $CC2 \geq 1$ and $CC3 = 1$

All elements, relationships, and properties have been manually verified.

3. Automated verified

$CC1 \geq 2$ and $CC2 \geq 2$ and $CC3 = 2$

All elements, relationships, and properties have been verified with tool support.

4. Model verified

$CC1 = 3$ and $CC2 = 3$ and $CC3 = 3$

All elements, relationships, and properties have been verified with mathematically-based methods wherever possible, or the most rigorous method otherwise.

5. Formally verified

$CC1 = 3$ and $CC2 = 3$ and $CC3 = 3$ and 'Model == Input'

Model verified where it is proven that the results of the mathematically-based methods are true for the actual input (and not only for an abstract model of it).

The level represents the maturity of the software product. The lowest achievement level determines the maturity of the software product and thus the Certification Level.

2.4 Specific criteria

Specific Criteria (SC) are criteria that apply to one Product Area only. Each Product Area has a different set of Specific Criteria (although they convey some similarity as they are based on the Certification Criteria).

In the following sections, we give a (necessarily incomplete) list for the Requirements Product Area. The Specific Criteria are a direct translation of the three Certification Criteria to the Product Area. The required elements, standards and checks are different for each Product Area.

The Specific Criteria indicate the elements or checks that are required for a certain Certification Criteria Achievement Level. For instance, to achieve level 2 for CC3 (conformance), all checks in SC3.1 (manual) and SC3.2 (tool-supported) need to be performed and should not reveal any faults. The set of Specific Criteria has been collected from the literature, including ESA (1991), Sommerville and Sawyer (1997), Alexander and Stevens (2002), Hull et al. (2002), Firesmith (2003), and (Firesmith 2005), Young (2004), Wiegers (2003), and (2006), Mazza et al. (1996), our own experience and the following standards:

- ISO
  - 9001:2000
  - 9126:2001
- ESA software engineering standards. Issue 2, 1991
- IEEE Computer Society
  - Std. 610.12-1990.
  - Std. 829-1998.
  - Std. 830-1998.
  - Std 1016-1998.
  - Std 1063-2001.
  - Std 1233-1998.
- IEEE/EIA 12207.0-1996: Industry Implementation of International Standard ISO/IEC 12207:1995

Note that it can also be the case that the input artifacts contain only part of a Product Area. If the correspondence between the input artifacts and the Product Area elements is clear, the applicable checks can easily be determined.

2.4.1 User requirements

The following Specific Criteria apply for the User Requirements Product Area.

[SC1] Complete: The requirements is as detailed and as formal as possible. The first sub-criterion indicates what the required elements are. The last two sub-criteria indicate what elements can be formalized.

[SC1.1] Required Elements: The following elements are absolutely required for a user requirements specification.

- *Functional requirements*: *Functional requirements describe the functionality of the system from the perspective of the user. This can be done in plain text or in the form of use cases (see below).*

- *Nonfunctional requirements*: These are also called quality requirements. It is a set of different types of quality measures. See the ISO/IEC 9126 standard (2001) for quality characteristics.
- *Glossary*: Many types of entities play a role in the environment's processes but only those that have to be represented in the system are collected. Not the individual entities, but only the types or classes to which they belong are listed (so not "client Johnson", but only "client"). The object description can be quite informal in the form of a glossary (terms and definitions), or more advanced in the form of a data dictionary or object model (see below).

[SC1.2] Semiformal Elements: The following elements make the user requirements specification convertible into a formal specification: *data dictionary* or *object model*, *use cases* (with scenarios), *flowcharts of processes*, and *behavioral properties*.

[SC1.3] Formal Elements: The following elements formally specify the user requirements: *relational diagram* of data/object model, *process models* of *use case scenarios*, and *behavioral properties specification*.

[SC2] Uniform: The style of the requirements description complies with standards in requirements engineering.

[SC2.1] Uniformity—Within the Product Area there are no elements that deviate from the rest.
[SC2.2] Compliance with Company Standards—Within the Product Area there are no elements that deviate from the applicable company standards.
[SC2.3] Compliance with Industry Standards—Within the Product Area there are no techniques used that deviate from industry best practice.

[SC3] Conformance: Each element in the requirements is described in a correct and consistent way. The relations between the elements in the requirements description and with the context description are correct and consistent.

[SC3.1] Manual checks: The following checks can be executed manually.

- *No two requirements or use cases contradict each other*: it is not the case that one requirement describes property P and another requirement describes property Not P. It is not the case that one use case describes an order of steps and another use case describes a different order of steps, etc.
- *No requirement is ambiguous*: it is clear what the requirement means. No term in the requirement has an alternate meaning that can be misunderstood by any of the stakeholders. It is clear where the emphasis in the requirement is.
- *Functional requirements specify what, not how (no technical solutions)*: the user requirements do not constrain the technical solution. Any design and development constraints are part of the nonfunctional requirements.
- *Etc.*

[SC3.2] Automated checks: The following checks can be executed with tools, for example with a requirements management tool.

- *Requirements have a unique ID*: the tool assigns and checks that the ID of each requirement is unique.
- *Requirement elements' relations are checked automatically*: the tool shows the relationship between requirements, scenarios, actors, and objects.

[SC3.3] Formal Checks: The following checks can be executed with formal methods.

- *The use case scenario models are correct workflows*: a correct workflow has one or more start points and one or more end points. It does not contain any deadlocks or starvation and no dead tasks (tasks that can never be executed). When the end point is reached, no tasks are left over.
- *The use case scenario models are mutually consistent*: the aggregation of all use case models is a correct workflow (see previous item).
- *The data model diagram is in normal form*: a normal form rigorously defines the relationships between entities. The first normal form (1NF) basically states that an attribute can only store one value. The second and third (2NF and 3NF) normal forms deal with the relationship of non-key attributes to the primary key. The fourth and fifth normal forms (4NF and 5NF) deal specifically with the relationship of attributes comprising a multiattribute primary key. Sixth normal form (6NF) only applies to temporal databases.
- *Etc.*

The complete list of specific criteria can be found in Chap. 5 of the technical paper on LSPCM by Heck and Van Eekelen (2008).

2.5 Tailoring

The above list of specific criteria represents our view on product certification. We use this model in our own certification efforts. We are aware that this list is never complete and that others may have a different view or use different terminology. Therefore, we allow others to take the basics of the certification model and adapt it to their specific needs.

The tailoring of the Software Product Certification Model to company- or project-specific situations can be done in a number of ways:

1. *Add Product Areas or Elements*. It is not allowed to remove any of these.
2. *Change names of Product Areas or Elements* to company or project jargon.
3. *Add checks* (SC3). It is not allowed to remove any of these. If they are not applicable in the company or project situation, they should be marked as "N/A".
4. *Detail elements and checks* (SC1 till 3). Make the descriptions of elements and checks more detailed with, for example specific company or project information, standards, or tools to use.

In this way, the main concepts of the model remain standing, but each company and project can customize the contents of the concepts. By not removing any of the existing elements and checks, there is still comparison possible with software products from other companies or projects.

3 Concrete certificate types

For each combination of product area, property, and certification level, a concrete certificate has to be made. This requires deciding which concrete criteria are appropriate and formulating the corresponding checks in the terminology and context. Certain certificate types can be defined based on the artifact areas, the type of conformance, the certification criteria, and the achievement levels. A certificate type indicates a predefined "check", which can be performed on a software artifact. For each certificate type, the items are defined in Table 2. It may seem that it is impossible to produce concrete certificates for

Table 2 Certificate type items and their definition

Product area	<The type of input artifact: one of the areas, see Sect. 2.1>
Properties	<The type of conformance analysis or the property that has to be checked, see Sect. 1.1>
Level	<The level for the "Conformance" certification criterion, see Table 1>
Description	<A short description of the goal of the certificate type>
Input	<A precise specification of the input needed for this certificate type>
Checks	<For the relevant achievement levels of the different criteria (see Sect. 2.2), a list of checks is included that are part of the certificate type; detailed information on the check can be found in the LaQuSo Software Product Certification Model>

very different product areas within the same model. In the next section, we show how this can be done for two chosen certificates in the product areas of requirements and of implementation. We evaluate the resulting certificates in Sect. 3.3.

3.1 A consistency certificate for user requirements

One of our first certification projects comprised the requirements verification of a medium-size industrial project. The system to be built was a central point where new identification numbers are generated, distributed, and registered. We were asked to judge the quality of the functional design, which consisted of functional requirements, 15 use case descriptions with UML activity diagrams, a process model of the business processes, a functional architecture (logical module structure), an object model, a glossary, and a supplementary specification (all nonfunctional requirements such as legal, security, performance, etc.).

The following types of inconsistencies were found:

- A number of spelling and structural errors were found. [SC3.1]
- Some postconditions of use cases were not consistent with the main scenario. [SC3.1]
- Activity diagrams did not use the correct (UML) symbols: for example included states as activities. [SC2.1]
- The object model did not use ERD symbols correctly and did not contain a detailed description for the attributes. [SC1.3 and SC2.1]
- The glossary contained only abbreviations. [SC1.1]
- The activity diagrams did not always match the use case text (especially not for the alternative flows). [SC3.1]
- One of the actors was not used in a consistent manner (a mix between human and nonhuman). [SC3.1]
- One use case mentions two options in the summary and illustrates only one in the scenarios. [SC3.1]
- Use cases described system features that were not mentioned in the other documents. [SC3.1]
- There was an overview document that did not contain all use cases and their relations. [SC3.1]
- Some components to support the use cases were missing in the functional architecture. [SC3.1]
- Use cases were missing related to the life-cycle coverage of objects (for example there was an "Open Session", but no "Close Session" use case). [SC3.1]
- Use cases for administration functions such as user management were missing. [SC3.1]

Table 3 User requirement certificate on internal consistency

Product area	User requirements
Properties	Consistency
Level	Manually verified
Description	Check on the internal consistency of the requirements
Input	Natural language requirements specification

Table 4 Specific checks for the certificate described in Table 5

Check	Description	Checked OK?	N/A if:
Required elements			
SC1.1 a	Functional requirements	Yes	–
SC1.1 b	Nonfunctional requirements	Yes	–
SC1.1 c	Glossary	Yes	–
Uniformity			
SC2.1 a	Uniform	Yes	–
SC2.2 a	Compliance to company standards	Yes	–
Manual checks			
SC3.1 a	No contradictions	Yes	–
SC3.1 b	No ambiguity	Yes	–
SC3.1 c	No technical solutions	Yes	–
SC3.1 d	Testable	Yes	–
SC3.1 e	Unique ID	Yes	–
SC3.1 f	Unique names	N/A	No use case structure
SC3.1 g	Atomic	Yes	
SC3.1 h	Noncyclic	Yes	
SC3.1 i	Elaborate use case	N/A	No use case structure
SC3.1 j	Diagrams match text	N/A	No use case structure
SC3.1 k	Terminology in glossary	Yes	
SC3.1 l	Detail environment description	N/A	No env. descr. Available
SC3.1 m	No useless actors and use cases	N/A	No use case structure
SC3.1 n	No useless or unspecified objects	Yes	–
SC3.1 o	Life-cycle coverage objects	Yes	–
SC3.1 p	Behavioral properties match	Yes	–
SC3.1 q	Nonfunctional requirements match	Yes	–

All major inconsistencies were solved before the design was handed over to the developers of the system. This minimized the input needed from the designers during the development phase and the risk for confusion and misinterpretation.

After correction of the major inconsistencies, the functional design was ready to receive the certificate as depicted in Table 3.

The following checks need to be answered with 'Yes' or 'Not applicable'. It is explicitly specified in Table 4 when an item may be marked as 'N/A'. Items with '-' in the fourth column must always be answered with 'Yes' in the third column to obtain a certificate.

A certificate will be handed out if all checks in Table 5 are answered with 'N/A' (if allowed according to the table) or 'Yes'. More details and other examples are given in Heck and Parviainen (Heck and Parviainen 2008).

3.2 A behavioral certificate for an implementation

After a successful case study by Van Eekelen et al. (2006) and (2007) of an analysis of an industrial implementation of the session layer of a load-balancing software system, it was decided to start a certification project for the system.

The system's software comprises 7.5 thousand lines of C code. It is used for distribution of the print jobs among several document processors (workers). In the case study, a large part of this commercially used software system has been modeled closely and analyzed using process-algebraic techniques using the mCRL2 toolset. This toolset is described at its webpage by the mCRL2 toolset developers (2009) and illustrated in a chapter in Alexander and Gardner (2008).

In addition to the standard properties such as avoiding deadlock and starvation, the properties that were checked were:

- Critical log messages must not occur,
- Load balancer may not distribute load in a filled queue if any of the queues is empty,
- If a load is distributed to a queue, both the queue and the load must register that they are assigned to each other,
- The number of items in the queue must be limited.

Since the model was close to the code, all problems that were found in the model could be traced back to the actual code resulting in concrete suggestions for improvement of the code. All in all, the analysis significantly improved the quality of this real-life system. The certification of the improved model was performed with the certificate in Table 5.

The following checks need to be answered with 'Yes' or 'Not applicable'. It is explicitly specified in Table 6 when an item may be marked as 'N/A'. Items with '-' in the one-but-last column must always be answered with 'Yes' to obtain a certificate.

A certificate will be handed out if all checks in the above table are answered with 'N/A' (if allowed according to the table) or 'Yes'.

In addition to the case study, two extra properties were checked. These properties were not considered in the case study since in the case study the only properties that were checked were those that were asked for by the company. For the certificate also, other behavioral properties were required. Furthermore, the certificate required that the model matched the properties sufficiently. The case study only checked one combination (3 clients, 1 server). For the certificate all client–server combinations of four processes were fully checked. The final verification took almost 17 days processing time and more than 100 MB of memory. All checks were answered 'Yes'. So, the certificate was handed out.

Table 5 Implementation certificate for behavioral checks

Product area	Implementation
Properties	Behavioral
Level	Model verified
Description	Check the behavioral properties on the formal models; Check conformance between the source code and the formal models;
Input	Formal and informal models of the component behavior, source code, safety and progress properties

Table 6 Checks for the certificate as defined in Table 5

Check	Description	Checked OK?	N/A if:
	Detailed design certificate granted	N/A	Direct from source code
Required elements			
SC1.1 a	Software system	Yes	–
SC1.2 a	Technical specification	Yes	–
SC1.3 a	Process models of the system	Yes	–
Manual checks			
SC3.1 a till l	Consistency of elements	N/A	Element not delivered
SC3.1 m	Relevant and feasible properties	Yes	–
SC3.1 n	Formal and informal model conform	Yes	–
Formal checks			
SC 3.3 a, c, d, f	Consistency of elements	N/A	Element not delivered
SC3.3 b	Process models are correct	Yes	–
SC 3.3 e	Code correctly generated	N/A	No generated code
SC3.3 g	Models match behavioral properties	Yes	–

3.3 Comparison

The two example product areas (requirements and implementation) are very different.

The resulting certificates are very different although they follow a common structure. They use their own specific terminology both for the structure and the content of the checks. As a result, it is easy for someone knowledgeable in a specific product area to use the corresponding certificate.

Still, the certificates for both of these areas are built with one and the same certification model.

The basis for each certificate consists of the answers to three questions:

1. What is the input that we get (the heading in the certificate above)?
2. What elements and properties are present in the input and are they uniform (‘completeness’ and ‘uniformity’ in the certificates above)?
3. What are the relationships between the elements and properties that we have and how can we check them (‘Checks’ in the certificates above), either manually or with tools or formal methods?

These three questions are the same for each certificate type, but the answers are different. An example follows. Behavioral properties are present in both the requirements phase and the implementation phase. An example of a behavioral property in the requirements case study is “Sessions that are opened are eventually closed”. An example of a behavioral property in the behavioral case study is “Each thread that tries to acquire a lock will eventually get it”. For the behavioral certificate we do not only ask for the source code, but also ask for (or construct) formal models and behavioral properties. For the requirements certificate above, the input is only the requirements, but behavioral properties are normally included somewhere in the (functional) requirements. But for both certificate types, we check what we have (requirements or formal models) against the behavioral properties that are present or constructed; see check 3.1p in the requirements certificate and 3.3 g in the behavioral certificate.

4 Related work

The first version of our model was inspired by CMMI. In Heck (2006a) we translate the concepts for process maturity into product maturity and show applicability for the requirements product area. In Heck (2006b), this first version is extended to all product areas. The concept of maturity has been replaced with dependability (no faults detected), and some terminology has been adjusted to be more precise. Both maturity and dependability were translated into 'correctness' and 'consistency' properties. We have called the model a product certification model because it can be used to assess the quality (dependability) of software products.

The current version of the model by Heck and Van Eekelen (2008) has combined the two properties 'correctness' and 'consistency' of Heck (2006b), about which much confusion on the exact meaning existed, into one single property called 'conformance'. In that way, any kind of property can be assessed within the framework of the model (not only consistency and detection of faults). The broadness of the assessment makes it a true certification model: a framework to define concrete certificate types for specific properties to assess. We have also changed the names of the other certification criteria to cover their contents better.

We did not find any other models that describe product quality in the sense of analyzing the correctness of a product. However, there are some other related findings in the area of product quality.

The software product maturity model by Nastro (1997) has three core elements: product capability, product stability, and product maintainability. Two sub-elements, product repeatability and product compatibility, are not universal to every software product. Nastro provides example measures for each of the elements like tests failed, changes per week, number of patches.

The Component Certification Framework of Alvaro et al. (2007) comprises a component quality model, certification techniques framework, a certification process, and a metrics framework. This covers a wide spectrum of verification techniques and checks to evaluate software components.

Both the Nastro model and the Component Certification Framework differ from our model in the first and foremost place because it only measures properties of the end product and not for example the requirements or the design. The Nastro model also seems more appropriate for the tracking of development progress (i.e., comparison of builds within one project) than for the objective measurement of the product quality. As Nastro states the importance or weight of the elements and even the elements themselves may vary from project to project.

Similar to our model, the model of Welzel and Hausen (1997) evaluates all important software artifacts in five steps: requirements, specification, design, conduct, and report of evaluation. But in contrast to our model the Welzel and Hausen evaluation needs all artifacts of one software product. Also their evaluation level is related to the required reliability of the end product, not to the level of quality of the artifacts themselves.

The Requirements-driven Workbench of Lee et al. (2007) on software security certification shows a practical certification and accreditation process for end products; again, a focus particularly on the end product and only one of the characteristics in ISO/IEC standard 9126.

The ISO/IEC standard 9126: "Software engineering—Product Quality" (2001) describes a two-part model for software product quality: (a) internal and external quality, and b) quality in use. The first part of the model specifies six characteristics (see Fig. 3) for internal and external quality, which are further subdivided into subcharacteristics. These

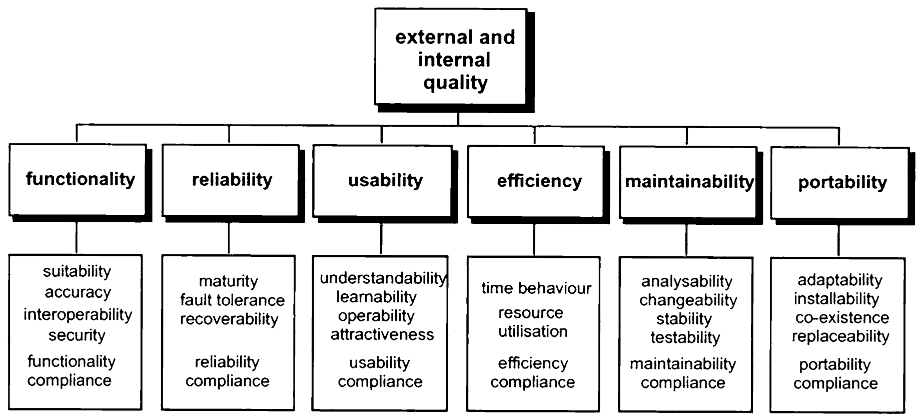

Fig. 3 ISO 9126 internal and external quality characteristics

subcharacteristics are manifested externally when the software is used as a part of a computer system and are a result of internal software attributes. The second part of the model specifies quality in use characteristics. Quality in use is the combined effect for the user of the six software product quality characteristics. The standard also provides metrics for each of the quality characteristics to measure the attributes. An explanation of how this quality model can be applied in software product evaluation is contained in ISO/IEC 14598-1 (1999).

An evaluation according to ISO/IEC 9126 is mostly based on metrics whereas our model uses a more rigid scale by providing yes/no checks. This yes/no scale leaves more room for expert opinions, but also caters for less precise comparison between two products. As our model focuses on correctness and consistency, ISO/IEC 9126 does not address consistency between elements as a separate concern. Correctness is in ISO/IEC 9126 mostly determined through indirect measures (for example measure the number of defects found during a production period). We, therefore, believe that our model is more suitable to determine product quality (correctness and consistency), whereas the ISO/IEC model is more suitable for specifying and measuring the desired product quality (all six characteristics). In the future, we could extend our model with other characteristics from ISO/IEC 9126.

The certification levels in our model are similar to the review levels in an article by Connie Clayton (1996). Clayton identified the different levels in which documents can be reviewed to standardize the review process:

Level 1: Document completeness: individual document;
Level 2: Compatibility with standards;
Level 3: First-level consistency check: internal;
Level 4: Second-level consistency check: requirements check, external, minimal CASE tool usage;
Level 5: Major review: review code logic, algorithms, full use of CASE tools.

Before a higher level of review can be attempted, the steps of all previous levels must be completed. The accompanying checklists are focused on the American Department of Defense related standards (DOD-STD-2167A and MIL-STD-498), so they are not always applicable. Furthermore, some questions are subjective (“is the document legible?”) or hard to check (“is there any irrelevant data?”). The lower level checklists contain many precise questions but the higher levels are less well defined.

Jakobsen et al. (1999) describe a five-level maturity model for software product evaluation, where they apply the concept of a maturity model to product quality evaluations. They assume that product quality increases when evaluation methods get more mature (from basic testing against requirements to continuously optimizing the evaluation process). Level 2 (testing against basic requirements and achieving satisfactory results) is carried out by checking a product's conformance to the ISO 12119 standard. Parts of the maturity model have been incorporated in the ISO/IEC 14598 standard. As said, this maturity model focuses on the evaluation process and thus fundamentally differs from ours. We could however also use ISO 12119 as an additional standard to collect Specific Properties from.

Software certification as performed by for example the FDA (2002) does not prove conformance. If a product receives certification, it simply means that it has met all the requirements needed for certification. It does not mean that the product possesses certain properties. Therefore, the manufacturer cannot use certification to avoid assuming its legal or moral obligations. We proposed a certification model that does focus on conformance.

In summary, we can say that we did not encounter any models that address product quality in the same sense that we do: related to conformance. There are, however, many approaches to software certification, that mostly rely on formal verification, expert reviews, or software metrics to determine the product quality, like described in Nastro (1997), Welzel and Hausen (1997), Lee et al. (2007), Alvaro et al. (2007), and Niinimäki and Forsström (1998).

We believe that our approach adds value with its comprehensiveness (from requirements to tests), its focus on conformance and by establishing a standard to perform software certification that also includes expert reviews and formal verification if necessary. It uniformly establishes what to check and how to check it. These checks are not new, but there is no place yet where they all have been put together into one model.

5 Conclusion and future work

In this article, we have described and applied a Software Product Certification Model. We applied the model to two product areas that are substantially different.

The model cannot only be used to certify products after their creation, but it also indicates which elements and relations should be present in the product when it is being created. Thus, the model can be used by both software developers and auditors. The Specific Properties are easily converted into a usable checklist, for convenient scoring.

We claim that for a thorough software product certification, formal verification is necessary, but requires a higher effort since products commonly will be derived from other software artifacts. It should first start with more simple checks: are all elements present, are their relations consistent, are standards complied to, etc. Our model is comprehensive and flexible enough to allow certification of software products in all life-cycle stages, with the applicable rigor for the criticality of the software, up to formal verification for the most critical products.

We continue to extend our model and apply it in industry case studies to demonstrate the added value of it, in the hope that our LaQuSo Software Product Certification Model (LSPCM) becomes recognized as a product standard.

References

Alexander, M., & Gardner, W. (2008). *Process algebra for parallel and distributed processing, 2008. Vol. 2*. London: Chapman & Hall/CRC Computational Science.

Alexander, I. F., & Stevens, R. (2002). *Writing better requirements*. London: Pearson Education Ltd.

Alvaro, A., Santana de Almeida, E., & Lemos Meira, S. (2007). Towards a software component certification framework, quality software, 2007. *QSIC'07, Seventh International Conference*, October 11–12, 2007 (pp. 298–303).

Bamford, R., & Deibler, W. J. (2004). *ISO 9001:2000 for software and systems providers. An engineering approach*. CRC Press: US.

Boehm, B. (2001). Software defects reduction top 10 list. *IEEE Computer, 34*(1), 135–137.

Clayton, C. (1996). *Defining review levels for software documentation*. CrossTalk, Vol. 9, Nr. 1.

CMMI Product Team. (2001). Capability maturity model® integration (CMMI^{SM}). Version 1.1. CMMI^{SM} for systems engineering and software engineering (CMMI-SE/SW, V1.1). Continuous Representation. CMU/SEI-2002-TR-001, ESC-TR-2002–2001, December 2001.

ESA. (1991). Board for Software Standardisation and Control (BSSC). ESA software engineering standards. Issue 2, 1991.

FDA. (2002). General principles of software validation; Final guidance for industry and FDA Staff. January 11.

Firesmith, D. G. (2003). Specifying good requirements http://www.jot.fm/issues/issue_2003_07/column7. *Journal of Object Technology*, 2(4), 77–87.

Firesmith, D. G. (2005). Quality requirements checklist http://www.jot.fm/issues/issue_2005_11/column4. *Journal of Object Technology*, 4(9), 31–38.

Forsström, J. (1997). *Why certification of medical software would be useful?* Medical Informatics Research Centre, University of Turku, Turku, December 1997; 47(3) pp. 143–152.

Heck, P. M. (2006). A Maturity Model for Software Product Certification. In *Proceedings International Workshop on Software Certification, Certsoft'06*, Hamilton ON, Canada, August 26–27, 2006.

Heck, P. M. (2006b). *A Software product certification model for dependable systems. CS-Report 06–20*. Eindhoven: Technische Universiteit Eindhoven.

Heck, P. M., & Parviainen, P. (2008). *Experiences on analysis of requirements quality, proceedings of the third international conference on software engineering advances ICSEA 2008 Sliema Malta 26–31 Oct 2008* (pp. 367–372). New York: IEEE computer society.

Heck, P. M., & van Eekelen, M. (2008). *The LaQuSo software product certification model, CS-Report 08–03*. Eindhoven: Technical University Eindhoven.

Hull, E., Jackson, K., & Dick, J. (2002). *Requirements engineering*. Germany: Springer.

Jakobsen, A., O'Duffy, M., & Punter, T. (1999). Towards a maturity model for software product evaluations. In *Proceedings of 10th european conference on software cost estimation (ESCOM'99).*

Kruchten, P. (2004). *The rational unified process: An introduction* (3rd ed.). Boston: Addison-Wesley.

Lee, S.-W., Gandhi, R. A., & Wagle, S. (2007). *Towards a requirements-driven workbench for supporting software certification and accreditation, software engineering for secure systems, 2007. SESS'07*: ICSE Workshops 2007. Third International Workshop, pp. 8–8, 20–26 May 2007.

Mazza, C., Fairclough, J., Melton, B., De Pablo, D., Scheffer, A., Stevens, R., et al. (1996). *Software engineering guides*. New Jersey: Prentice Hall.

mCRL2 development team. (2009). *mCRL2 toolset webpage*: http://mcrl2.org/. Consulted April 13th, 2009.

Nastro, J. (1997). *A software product maturity model*. CrossTalk, Vol. 10, Nr. 8.

Niinimäki, J., & Forsström, J. (1998). Approaches for certification of electronic prescription software. *International Journal of Medical Informatics, 47*(3), 175–182.

Sommerville, I., & Sawyer, P. (1997). *Requirements engineering: A good practices guide*. New Jersey: Wiley.

The Standish Group International. (1996, 1998, 2000, 2002, 2004, and 2006). *Inc The CHAOS Report, published on* www.standishgroup.com. Webpage consulted on April 28, 2009.

Van Eekelen, M., Hoedt, Ten, S., Schreurs, R., & Usenko, Y. S. (2006). Testen van proces-communicatie en synchronisatie van LoadBalancer software via model-checking. *Proceedings van de 12e Nederlandse Testdag*. November 17, 2006 (pp. 23–25). ASML, Veldhoven.

Van Eekelen, M., Hoedt, Ten, S., Schreurs, R., & Usenko, Y. S. (2007). Analysis of a Session-Layer Protocol in mCRL2. Verification of a Real-Life Industrial Implementation. *12th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2007)*, 1–2 July, Berlin, 2007. Lecture Notes Computer Science. Springer.

Wegner, E. (1999). *Quality of software packages: the forthcoming international standard*, National Research Center for Computer Science GMD, D-52754 Sankt Augustin, Germany, Vol. 20, No. 4–5 (208 p.), pp. 349–354.

Welzel, D., & Hausen, H.-L. (1997). Practical concurrent software evaluation for certification. *Journal Systems and Software, 38*, 71–83.

Wiegers, K. E. (2003). *Software requirements* (2nd ed.). Redmond, Washington: Microsoft Press.

Wiegers, K. E. (2006). *More about software requirements: Thorny issues and practical advice*. Redmond, Washington: Microsoft Press.

Young, R. R. (2004). *The requirements engineering handbook*. Norwood, MA: Artech House.

Author Biographies

Petra Heck received a M.Sc. in Computer Science from the Technical University Eindhoven, The Netherlands in 2002. At the Technical University Eindhoven, she worked as a consultant for the Laboratory for Quality Software (LaQuSo), which is a joint activity of the Technical University Eindhoven and the Radboud University Nijmegen. Currently, she is senior consultant at Software Quality Systems (SQS) in Geneva, Switzerland. Her research interests include certification, requirements specification, and business process descriptions.

Martijn Klabbers received a M.Sc in Computer Science from the Technical University Delft, The Netherlands in 1995. At the Technical University of Eindhoven, he is a senior consultant at LaQuSo (Laboratory for Software Quality). In the past, Martijn researched decision support models at the TUE. Before LaQuSo, he was successively software developer, project manager, and product manager at NIPO Software. His research interests include certification, decision support systems, and user requirements.

Marko van Eekelen received a Ph.D. in Computer Science from the Radboud University Nijmegen, The Netherlands in 1988 with promotor prof. Henk Barendregt. He is currently professor, holding the Software Technology chair at the Dutch Open University and also associate professor at the Institute for Computing and Information Sciences at the Radboud University Nijmegen, The Netherlands. He is the Nijmegen director of the Laboratory for Quality Software (LaQuSo), which is a joint activity of the Technical University Eindhoven and the Radboud University Nijmegen. His research interests include formal methods for software analysis, certification, validation and verification in general, and their applications on software security in particular.