Outsourcing and LaQuSo Certification

Marko van Eckelen
LaQuSo, Laboratory for Quality Software, a joint activity of Technical University Eindhoven and Radboud University Nijmegen
+31 24 365 3410, +31 24 365 2296, marko@laquso.com, www.cs.ru.nl-marko

ABSTRACT
In this paper the LaQuSo software product certification model will be discussed. 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 model is validated by creating concrete software certificates for two software product areas that are on different ends of the software product spectrum. For each certificate a concrete case study has been performed. The use of the model for these certificates has been evaluated. It has been shown that the model can be used satisfactorily for quite different kinds of certificates.

The LaQuSo software product certification model can be used for assessing the quality of software artifacts. We do not consider hardware and network aspects of systems, because they require very different means of verification.

The paper will focus on the use of the LaQuSo software product certification model for various artifacts that are used in the context of outsourcing.

Keywords: Software Certification, Outsourcing

1. INTRODUCTION

More and more applications depend on the reliability, availability, and integrity of ICT systems. Due to the increase of complexity in the hardware, software, and communication level, creating quality systems has become a major scientific and engineering challenge.

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 at all cost. The developer of the system must verify the system before delivering it, e.g. through testing or manual review.

However, if when systems are safety-critical and are to be used in a broad environment or when parts of a system are produced by an external development team (outsourcing), 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.

In the case of outsourcing, independent quality evaluation is important both at the beginning of the outsourcing (specifications/requirements) and at the end of the outsourcing (final software product). The need is felt to have a certification procedure that can be applied uniformly to each of the inputs/outputs of each phase. For this purpose we generalize the concept of software product from executable source code to all intermediate results of the software development process. So, a specification, a set of requirements and a technical design are also called software products.

In this paper we present a certification model for assessing the quality of such software products. 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 extensively studied elsewhere.

We believe that provable quality of software products can be based on the application of justified product analysis techniques, 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 describes how the model can be defined as a certificate type. Section 4 contains examples of pre-defined certificate types that can be used in the context of outsourcing. Section 5 refers to related work and section 6 contains the conclusion.

1 This paper builds heavily upon earlier work with Petra Heck [8].
2 Dr. M.C.J.D. van Eckelen (Marko) is an associate professor at the Diagonal Security Department in the Institute for Computing and Information Science in the Nijmegen Faculty of Science. Furthermore, he is the director of the Nijmegen part of the Laboratory for Quality Software (LaQuSo).
When the achievement level for each of the three Certification Criteria has been established, the overall Certification Level of the product can be determined (see Section 2). The more formal elements are present in the product and the more formal checks have been performed without detection of faults or non-conformance, the higher the confidence in the product certificate is.

2. THE LAQUISO SOFTWARE PRODUCT CERTIFICATION MODEL (LSPCM)

We describe the LaQuSo Software Product Certification Model in the following sections.

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.

![Software Product Areas Diagram](image)

Figure 2: Software Product Areas with their Elements

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 test (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 derived with the system, the non-functional requirements will be a separate section of the user requirements document, and the stakeholders can be described as part of the business process description (e.g., 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. E.g. 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.

CERTIFICATION CRITERIA

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

- **CC1 Formality.** All required elements in the Product Area should be present and as much formalized 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 subject of the certification.

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

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 semi-formal (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 subject to formal verification (less transformation is needed). For examples of semi-formal and formal elements see SCI.2 and SCI.3 in the Specific Criteria Section below.

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.

<table>
<thead>
<tr>
<th>Table 1: Certification Criteria Achievement Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CC1 Formality</strong></td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td><strong>CC2 Uniformity</strong></td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td><strong>CC3 Conformance</strong></td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

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:

---
As many formalized elements as possible. 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.

As much standardization as possible. Score 0 if elements of the same type have different style (e.g. 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.

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. As an example Specific Criteria for the Requirements Product Area are provided in the User Requirements paragraph of the Specific Criteria section below.

CERTIFICATION LEVELS

From the levels that have been achieved for the three Certification Criteria an overall Certification Level can be calculated.

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 e.g. 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, e.g. only the communication protocol of a finished system.

The certification levels are based on an intuitive notion of when product certificates are more authoritative. 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 four certification levels. 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=2 and CC2=2 and CC3=0
   Each of the required elements is present in the product and the elements are uniform.

2. Manually verified
   CC1=2 and CC2=2 and CC3=1
   All elements, relationships and properties have been manually verified.

3. Automated verified
   CC1=2 and CC2=2 and CC3=2
   All elements, relationships and properties have been verified with tool-support.

4. Model verified
   CC1=2 and CC2=2 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=2 and CC2=2 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).

Of course we also indicate that products with levels higher than 1 for CC1 and CC2 are more mature. However, for certification purposes these higher levels are only 'nice to have' (it will take less time to certify these types of products) and not mandatory. That is why they do not appear explicitly in the above scale.

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 literature, our own experience and the following standards:

- ISO
  o 9001:2000
  o 9126:2001
- ESA software engineering standards. Issue 2, 1991
- IEEE Computer Society
  o Std. 610.12-1990.
  o Std. 829-1998
  o Std. 830-1998
  o Std. 1016-1998
  o Std. 1233-1998.

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.

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.

a. 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).

b. Non-functional 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.

c. Glossary

Many types of entities play a role in the environment 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] Semi-formal elements

The following elements make the user requirements specification convertible into a formal specification.

a. Data dictionary or object model

A data dictionary is a set of metadata that contains definitions and representations of data elements. It includes semantics for data elements. The semantic components focus on creating precise meaning of data elements. Data dictionaries are more precise than glossaries because they frequently have one or more representations of how data is structured. Data dictionaries can be completed with data or object models that also include complex relationships between data elements or objects.

b. Use-cases (with scenarios)

A use-case is a named "piece of functionality" as seen from the perspective of an actor. For each use-case several use-case scenarios are given (sequences of actions, events or tasks), the permitted or desired ones as well as the forbidden ones.

c. Flowcharts of processes

A flowchart (also spelled flow-chart and flow chart) is a schematic representation of a process. Generally the start point, end points, inputs, outputs, possible paths and the decisions that lead to these possible paths are included.

d. Behavioral properties
General behavioral properties are e.g. properties that express that certain conditions may never occur or that certain conditions should always hold. Usually these properties have a temporal aspect and therefore it is possible to express them in temporal logic, although a translation in natural language is essential for most stakeholders.

[SC1.3] Formal Elements

The following elements formally specify the user requirements.

a. Relational diagram of data/object model
A relational diagram is a diagram used to describe conceptual data models by providing graphical notations for entities and their relationships, and the constraints that binds them. The basic graphical elements are boxes, representing entities, and arrows, representing relationships.

b. Process models of use-case scenarios
The process models describe the steps in the use-case scenarios in a formal language like Petri Nets or process algebra.

c. Behavioral properties specification
The behavioral properties are expressed in a formal language. If they e.g. have a temporal aspect, temporal logic can be used.

[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.

a. Elements and documents of the same type have the same style
E.g. all use cases have the same format. E.g. each requirement has the same attributes that are described.

[SC2.2] Compliance with Company Standards

Within the Product Area there are no elements that deviate from the applicable company standards.

a. All elements and documents comply with company standards
There can be templates for requirements or use cases within the company. There can be standard diagramming techniques as technical languages, etc.

[SC2.3] Compliance with Industry Standards

Within the Product Area there are no techniques used that deviate from the industry best practices.

a. ER diagram for object/data model
ER diagrams are a special kind of relational diagrams (see above). Entities are drawn as rectangles, relationships in diamonds. Attributes are drawn as ovals connected to their owning entity sets by a line. Lines are drawn between entity sets and the relationship sets they are involved in. The arity of the relationship is indicated on the line.

b. UML diagrams for use-cases
The UML use-case diagram indicates which use-cases and actors belong to the system and how they all are connected. The UML activity or state diagram shows the steps or states in the use-case scenarios. The UML class diagram can show the relationships between objects.

[SC3] Conformance

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


The following checks can be executed manually:

a. 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 the case that one use-case describes an order of steps and another use-case describes a different order of steps. Etc. to the contrary.

b. 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.

c. 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 non-functional requirements.

d. Each requirement is testable
The requirement can be objectively shown to hold. The requirement is expressed in precise and quantifiable terms.

e. Each requirement is uniquely identified
The requirement has a unique identifier (numbered). Preferably the requirements are ordered and numbered. Functional and non-functional requirements can only have the same number if they are preceded by a letter code.

f. Each use-case has a unique name
No two use-cases have the same or similar names.

g. Each requirement is atomic
A requirement is a single sentence that expresses one aspect of the system. Avoid long sentences and the use of too many proverbs. Different properties of the same aspect are sub-requirements. Requirements are split whenever possible.

h. The definitions in the glossary are non-cyclic
There is no definition d in the glossary that refers to other definitions, that refer to other definitions, until the definition d is referred to.

i. A use-case is elaborate
A use-case describes at least pre-conditions, post-conditions, normal flow, and alternate flows (including exceptions).

j. Use-case diagrams correspond to use-case text
If diagrams are drawn in the use-case description to show the steps in the use-case, the description and order of the steps is the same in both the text and the diagrams.

k. Ambiguity is explained in the glossary
Each ambiguous or unclear term from the requirements is contained in the glossary.

l. The use-cases or functional requirements detail the environment description
The use-cases or functional requirements detail the environment description in the context description (no contradictions). Each step in a business process that involves the system has been included in the requirements. Each task that the system should fulfill for its environment has been included in the requirements. All actors of the context description have been included in the requirements.

m. No useless actors and use-cases
Each use-case is involved with at least one actor and each actor is involved with at least one use-case.

n. No useless objects and all objects specified
Each object is mentioned in the requirements and all objects mentioned in the requirements contained in the object model.

o. Life-cycle coverage of the objects
For each object the create-, read-, update- and delete operations are covered in the user requirements or not applicable.

p. The requirements do not contradict the behavioral properties
None of the behavioral properties is rendered impossible by the requirements.

q. The functional and non-functional requirements do not contradict
The use-case or functional requirements do not render the non-functional requirements impossible.

r. The certification property is relevant and feasible
The certification property answers a question about the requirements. The question should be relevant with respect to the software product and the answer should be obtainable with current techniques.

s. Formal and informal descriptions conform
If there are both an informal (e.g. natural language) and a formal model (e.g. process models) of the requirements they should not contradict each other.

[SC3.2] Automated Checks

The following checks can be executed with tools.

a. Requirements are stored in a requirements management tool
The requirement management tool automatically assigns a unique ID to each requirement.

b. Requirements are stored in a requirements management tool
Requirements and glossary/objects are stored in a requirements management tool which shows the relations between requirements, scenarios, actors, and objects.

[SC3.3] Formal Checks

The following checks can be executed with formal methods.

a. 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 dead-locks or live-locks and no dead tasks (tasks that can never be executed). When the end point is reached, no tasks are left-over. The Workin tool can analyze the workflow soundness of Petri Nets.
b. The use-case scenario models are mutually consistent
   The aggregation of all use-case models is a correct workflow (see previous item).

c. 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 multi-attribute primary key. Sixth normal form (6NF) only applies to temporal databases.

d. The use-case scenario models comply with the certification property
   The certification (e.g. behavioral) properties are expressed in e.g. temporal logic and proved to hold for the process models in e.g. Petri Net formalism.

e. The use-case scenario models comply with the non-functional requirements
   The non-functional requirements are expressed in e.g. timing constraints and proved to hold for the process models in e.g. Petri Net formalism.

f. The requirements description complies with the environment description
   The process models and the behavioral properties comply with the process models from the context description. The integration of all process models is possible and yields a correct workflow (see before).

TAILORING

The above list of specific criteria represents LaQuSo's 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 adopt 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 e.g. 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 OUTSOURCING

In principle 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 types, the type of conformance, the certification criteria and the achievement levels. A certificate type indicates a predefined "check" LaQuSo can perform on software artifact. For each certificate type the following items are defined:

- **Product Area**
- **Properties**
- **Level**
- **Description**
- **Input**
- **Checks**

It may seem that it is impossible to produce concrete certificates for very different product areas within the same model. In the next section we show how this can be done for two chosen certificates that are in many cases required and the output of an outsourcing process: the product areas of requirements and of implementation. We evaluate the resulting certificates in the last part of this Section.

**A Consistency Certificate for an Outsourcing Input: 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 non-functional requirements such as legal, security, performance etc.).

The following types of inconsistencies were found:

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

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 below.

**Product Area**
- **User Requirements**

**Properties**
- **Consistency**

**Level**
- **Manually verified**

**Description**
- **Check on the internal consistency of the requirements**

**Input**
- **Natural language requirements specification**

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

<table>
<thead>
<tr>
<th>Check</th>
<th>Description</th>
<th>N/A if:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Check Required Elements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC1.1 a</td>
<td>Functional requirements</td>
<td>-</td>
</tr>
<tr>
<td>SC1.1 b</td>
<td>Non-functional requirements</td>
<td>-</td>
</tr>
<tr>
<td>SC1.1 c</td>
<td>Glossary</td>
<td>-</td>
</tr>
<tr>
<td>SC2.1 a</td>
<td>Uniformity</td>
<td>-</td>
</tr>
<tr>
<td>SC2.2 a</td>
<td>Compliance to company standards</td>
<td>-</td>
</tr>
</tbody>
</table>

| Manual checks | | |
| SC3.1 a | No contradictions | - |
| SC3.1 b | No ambiguity | - |
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'.

A Behavioral Certificate for an Outsourcing Output: an Implementation

After a successful case study ([3], [4]) 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. 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 below.

<table>
<thead>
<tr>
<th>Product Area</th>
<th>Properties</th>
<th>Implementation</th>
<th>Behavioral</th>
<th>Model verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Input</td>
<td>Check behavioral properties on the formal models</td>
<td>Check conformance between the source code and the formal models</td>
<td>Formal and informal models of the component behavior, source code, safety and progress properties</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Check</th>
<th>Description</th>
<th>N/A if: Direct from source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required Elements</td>
<td>SCL1.1 a a Software system</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>SCL1.2 a Technical specification</td>
<td>-</td>
</tr>
</tbody>
</table>

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'.

Comparing to the case study, two extra properties were checked. These properties were not considered in the case study since in the case study only those properties were checked 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 4 processes were fully checked. All checks were answered 'Yes'. So, the certificate was handed out.

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 three steps:

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 (‘required elements’ 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 we present or constructed; see check 3.1p in the requirements certificate and 3.3g in the behavioral certificate.

The evaluation of the two certificates shows that it is possible to check the quality of both input and output of an outsourcing process in a uniform way.

4. RELATED WORK

The first version of our model was inspired by CMML. In [7] we translate the concepts for process maturity into product maturity, and proof applicability for the requirements product area. In [6] this first version is elaborated to all product areas. The concept of maturity has been replaced with reusability (no faults detected) and some terminology has been adjusted to be more precise. Both maturity and reusability 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 [8] has combined the two properties ‘correctness’ and ‘consistency’ of [7], 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 John Nastro [10] 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. He provides example measures for each of the elements like tests failed, changes per week, number of patches. The Nastro model differs from our model in the first and foremost place because it only measures properties of the end product and not e.g. the requirements or the design. It also seems more appropriate for the tracking of development progress (i.e. compare builds within one project) than for the objective measurement of the product quality. As Nastro states the importance of weight of the elements and even the elements themselves may vary from project to project.

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 of software quality: functionality, reliability, usability, efficiency, maintainability and portability. 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).

Figure 3: ISO 9126 Internal and External Quality Characteristics

An evaluation according to ISO/IEC 9126 is mostly based on metrics where our model uses a more rigid scale by providing yes/no checks. This yes/no scale leaves more room for expert opinions, but also ceters for less precise comparison between two products. As our model focuses on correctness and consistency, ISO/IEC 9126 does not address correctness between elements as a separate concern. Correctness is in ISO/IEC 9126 mostly determined through indirect measures (e.g. 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 usable to specify and measure 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 Conrie Clayton [1]. Clayton has 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 highly specialized on the American Department of Defense relaxed standards (DOE/STD-2167A, and MIL-STD-498), so not always usable in general. 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.

Jakubow et al. [9] 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 product’s conformance to the ISO 12119 standard. Parts of the maturity model have been incorporated in the ISO/IEC 14598 standard. As stated, 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 e.g. the FDA [5] 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 assigning 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 way as we do: related to conformance. There are however many approaches to software certification, that mostly rely on formal verification or expert reviews to determine the product quality.

We believe that our approach adds value with its comprehensiveness (from requirements to results), 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 were they all have been put together into one model.

5. CONCLUSION AND FUTURE WORK

In this paper we have described a validated Software Product Certification Model. We validated the model using two examples of product areas that are wide apart. The software products in these two product areas are usually the inputs and the outputs of an outsourcing process.

The model can not 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 comes at the end of the process. It should start first 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 industrially in case studies to demonstrate the added value of it, such that our Software Product Certification Model (SPCM) becomes recognized as a product standard.

REFERENCES