Software Industrialization and Architecture Certification

Christoph Rathfelder, Henning Groenda, Ralf Reussner

Christopf Rathfelder, Henning Groenda

Software Engineering
FZI Forschungszentrum Informatik
Haid-und-Neu-Straße 10-14
76131 Karlsruhe
{rathfelder, groenda}@fzi.de

Ralf Reusser

Chair for Software Design and Quality
Institut für Programmstrukturen und Datenorganisation
Universität Karlsruhe (TH)
reussner@ipd.uka.de

Abstract: The industrialization of software development induces several changes to the development process as software development becomes distributed over company borders. They cooperatively develop individual components that are later assembled to software systems. This division of responsibilities requires a stricter quality assurance and in fact, creates a setting where the certification of software products becomes increasingly interesting. Until now, there are a few software product certification approaches, as in non-component-based software development processes, the considerable effort of software certification was only rarely justified. Therefore, existing certification approaches do not consider and support the requirements posed by industrialization, namely the separation of component development (by various providers) and system development. This paper presents a software certification approach which takes these requirements into account and allows certifying individual components as well as system architectures.

1 Introduction

The industrialization of software engineering is in an early stage. Other more mature and well-established engineering disciplines, like car manufacturing or civil engineering are more advanced regarding industrialization of products' production. Here, industrialization is a management approach to lower the costs per unit. However, the main difference between software development and the industrial production is that software is an immaterial good, which can be copied cheaply after its development.

Therefore, the approaches used in the production processes of more mature engineering disciplines cannot directly be mapped to software engineering. However, approaches lowering the costs in software development, deployment and management can be summarized in the process of the "industrialization of software", as discussed below.

One important part of software industrialization is the outsourcing of development tasks to specialized organizations, which means that the software development is distributed over several companies. This division of responsibilities requires a stricter quality assurance. In order to prevent conflicts of interest between the involved companies, a standardized quality assessment is often necessary. Ideally, the results of such quality assessments need to be comprehensible for the commissioning party. In reality, this understanding might require too many resources and is often not feasible, as component developers are often not willing to open a component black box to preserve specific knowhow. Due to this, it is interesting to let an independent third party perform the quality assessments. This independent institution assesses and certifies statements about the software system and its quality. Certification gains interest, if (a) the quality of an artefact is important but (b) hard to assess for an outsider. The latter can be the case due to the lack of resources, knowledge or information.

In the context of software engineering, certification is nowadays mainly used to certify the knowledge of individuals or the execution of mandatory processes and process steps. The certification of software products or artefacts of the development process has still not found a wide distribution. It is only used in a rather limited range in special domains, e.g. safety critical systems like control systems of nuclear power stations.

The contribution of this paper is the presentation of a software certification process. The certification consists of an architecture and component certification and allows the integration of several evaluation techniques. Furthermore, the paper includes an overview of software certification and software industrialization.

The paper is structured as follows. Section 2 presents the foundations of software certification and software industrialization. Section 3 shows the requirements on certification in the context of software industrialisation and presents an approach to realize this certification. Section 4 concludes the paper and provides an outlook.

2 Foundations

2.1 Software and Certification

In the domain of software engineering, three different types of certification are distinguished: certification of products, processes, and personnel. These three disjoined types form the Software Quality Certification Triangle [Vo99], whereas each type affects the development and therefore the quality of software systems.

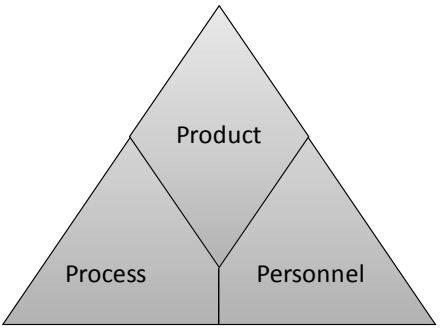

Figure 1: Software Quality Certification Triangle [Vo99]

The **certification of personnel** focuses on the knowledge and competence of individual engineers. Such a certificate can certify the presence of competences necessary to perform certain activities during the software development process. The Certified Tester and the Certified Professional for Requirements Engineering offered by the International Software Quality Institute (iSQI)[1] are two examples for these certifications. In addition to these more general certifications, there are a lot of technology and product dependent certificates available. The respective vendors are responsible for the definition of the curriculum. The certificate attests certified persons the required knowledge in a technology (e.g. Java) or a certain product (e.g. Microsoft Windows). Sun for example offers several different Java certificates within their Java Certification Program[2]. The knowledge helps a lot in handling the technologies or products. Nevertheless, a point of criticism of this certification type is that it is not certified that the certified knowledge is used during the development of a software system.

In contrast to the certification of personnel, the **process certification** is not linked to specific persons. It is rather valid for a department or a whole enterprise. To become certified, the enterprise has to prove that its processes are performed in a documented or specified way. The probably most well-known process certificate is the certification based on ISO9001 [In00] and its predecessor ISO9000, which defines the necessary activities of a general quality management process. Regarding software development, the Capability Maturity Model Integrated (CMMI) [Kn06], which is used to evaluate the software development process, is the most widely used process certification. However, there is no hard evidence that a better process automatically leads to a higher product quality. Even with a certified development process (e.g. CMMI Level 5) it is still possible that the resulting product has bad quality attributes [MW08].

The **product certification** focuses on a product and its related artefacts which were created during its development (e.g., test protocols, interface specifications, models). In the case of car manufacturing, the type approval is an example for a product evaluation and certification. Unfortunately, product certification has nearly no distribution in software engineering. It is used only in some special domains, for example safety critical systems within nuclear power station or airplanes.

[1] http://www.isqi.org
[2] http://www.sun.com/training/certification/java/index.xml

Although disjoined, all these types of certification affect the quality of software. Voas already mentioned in [Vo99] that a balanced combination of them may provide the best results and expressiveness. However, there is still no scientific knowledge on which is the right ratio. An example that combines product and process certifications is the Common Criteria (CC) [Cc07]. The CC is an internationally accepted standard that is used to certify the security of software systems. On the one hand, it defines mandatory activities that have to be performed during development and on the other hand it also includes an assessment of the final system.

Without certification, the software developer makes statements about its software product which the customer has to trust. Software certification aims at increasing the trustworthiness of statements about software. This is achieved by software assessments conducted by an independent third party. The resulting certification scenario and the participating roles are sketched in Figure 2 and described in the following. The certification authority assesses the product of the developer with a standardized and reproducible evaluation method in order to assure comparable evaluation results. A certificate is issued if the assessment leads to the conclusion that the statements are correct. Customers interested in the software can trust the certification authority instead of the developer himself as this is an independent authority which has a business model based on being trustworthy which makes them issuing wrong certificates unlikely.

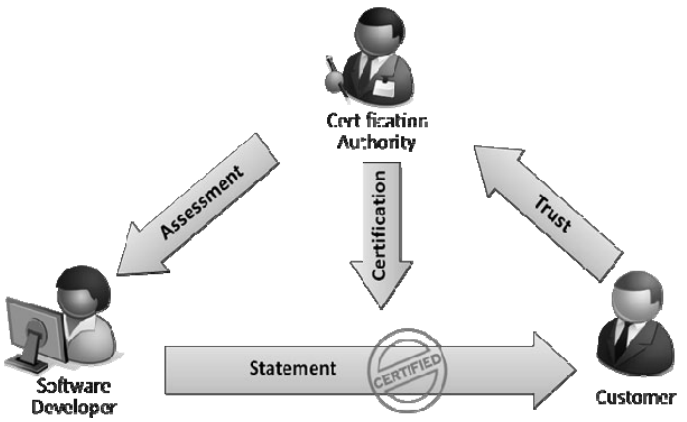

Figure 2: Software Certification Scenario

The acceptance and therewith the success of a certification initiative depends on several factors. One of most important factors is the independence and the trustworthiness of the certification authority. The trustworthiness, in turn, is affected by the performed certification. It has to be reproducible, which means, that a second certification of the same product yields the same results. This is of course the most important part of the trustworthiness.

2.2 Software Industrialization

Regarding the shift from manufacturing to industrial development and production, the software engineering discipline should learn from more mature engineering disciplines which already have undergone industrialization. Following alteration of these mature disciplines during their industrialization can be translated into software development:

- **Component-based development and standardization:**
  Car manufacturing is one example that has obviously a very high degree of standardization and reuse of the same components within several car models. For example, car engines are components which are reused within various car models of a brand, as this reduces the development cost for each individual car model. However, in order to allow flexible adoptions of the product, a standardization of the components respectively their interfaces is necessary. For example, electrical plugs must have the same measures and compatible cable connectors to allow the reuse and assembly of electronic car components.

- **Outsourcing and Offshoring:**
  Outsourcing and offshoring of development and production process steps are driven by specialization of work and are the second major ingredient of industrialization. Specialized companies are given specifications and entrusted with performing these processes steps or with delivering components. However, there is a slight difference between outsourcing and offshoring [Ta05]. In the case of outsourcing the main reason is focusing on the core competences of an enterprise, whereas offshoring is additionally driven by cost differences in a globalized market.

- **Model-based Quality Assessments:**
  In more mature engineering disciplines especially if they have undergone industrialization, an early quality assurance is an essential part of the development process. The quality of a product is thereby assessed on base of blueprints before the production of the product is started. Regarding bridge construction as an example, a structural engineer uses an architect's plan or model to calculate extra-functional properties e.g. the bearing capacity of a bridge *before* it is built. Car manufacturers likely use simulation techniques to conduct crash tests virtually on computers to check the quality of an engineered car, reduce costs, and speed up the development process by reducing the number of necessary prototypes.

With respect to software the approaches of component-based software development (CBSD) [SGM02] and also service oriented architectures (SOA) [KBS06] can be seen as first steps towards the industrialization of software development. Both of these approaches split complex software system into independent smaller parts with thorough requirement specifications - the components respectively services. The software system's architecture model describes the structure of the system and the interconnections between its elements [BCK99]. Due to the seclusiveness of the components and services it is possible to mandate other companies with their development. Quality assurance and in particular its proactive integration into software development processes is still a challenge in software engineering – in theory as well as practice. Especially the quality prediction and evaluation on base of models, respectively the software's architecture model, is a field in which software engineering is still immature compared to other engineering disciplines.

3 Certification and Software Industrialization

In this section we present how software certification should be accounted for in the software development process in order to support striving for software industrialization. Especially the distribution of the development process over several companies, caused by outsourcing and offshoring, requires a strict quality management. Certification of quality by an independent third party is an important part of this quality management.

As already mentioned above, component-based development and early model-based quality evaluations play an important role within software industrialization. The certification process should therefore allow an early quality assessment and certification on base of a system's architecture. This implies evaluations based only on the architecture model and the specification models of the assembled components. In so doing, flaws in the architecture can be detected faster and corrected without having to waste money for an inappropriate implementation first. The components themselves can be either bought from marketplaces or developed by contractors.

The evaluation of a software system's architecture without having a complete implementation needs model-based prediction and evaluation techniques as the system cannot be tested in this case. Each used component has to have a model of its behaviour, which can then be used for the architecture's assessment. The quality of the evaluation results is directly affected by the quality of the component models. Our approach therefore includes, in addition to the architecture certification, a component certification which certifies the correspondence of a component's model and its implementation. Each quality attribute is allowed to have a separate model, as long as the model of different components can be composed to allow inferences on the architectural level. If the component is not implemented yet the models can additionally be used to state the requirements for certain components in order to fulfil the requirements on the system level.

The distinction between architecture and component certification is an often neglected fact, although they differ in the certified statements. The first certifies that a modelled system will fulfil certain quality requirements; the later certifies that the quality model of a component is equivalent to a running component instance. We therefore propose to consider these as different parts of the certification process of software products. In the following, we describe these two parts in more detail.

3.1 Architecture Certification Process

Requirements posed on software systems differ dependent on the application scenarios of the software system. For example, a business information system typically requires only a certain percentage of all response times below a defined threshold. In contrast, a robot control system commonly requires hard deadlines, which means that all responses are below such a threshold. It must be mentioned, that even though most examples are based on performance of a systems, the certification process should allow the certification of other quality attributes (e.g., reliability, maintainability) as well. A certificate simply stating good or bad quality without taking the requirements into account is hence not desirable. For these reasons, the certification approach should be adaptable to allow the evaluation and certification with respect to different quality requirements. Nevertheless, the certification process should also support the usage of a standardized requirements catalogue which is defined for certain usage domains (e.g. business information systems) and thereby eases the comparison of different software systems within the same domain (e.g. a standardized response times for web applications).

The proposed approach to certify software architectures is illustrated in Figure 3. It consists of four activities which are described in the following in more detail.

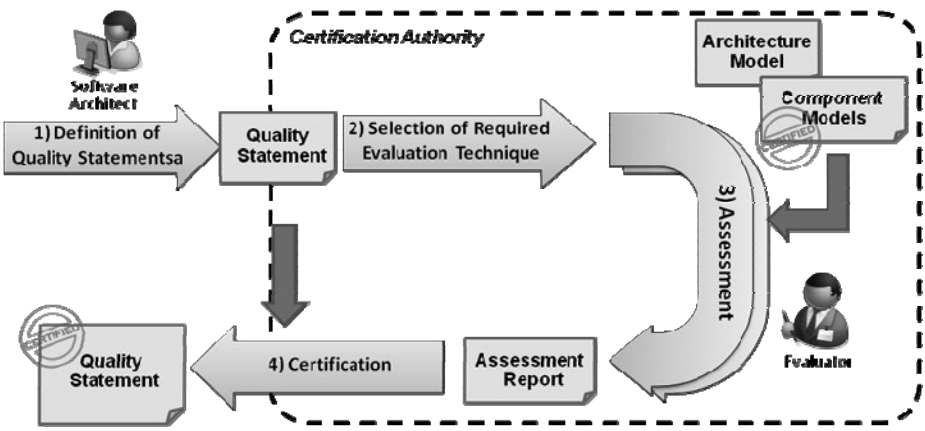

Figure 3: Architecture Certification Process

The certification of an architecture starts with the **Definition of Quality Statements** that should be certified. This is the only activity which can and should be influenced by the software architect. This step ends up with the document, which includes all *quality statements (QS)*. One example of a QS is: "For less than 6 concurrent users, the response time of the service A is below 3 seconds." As already mentioned above, it is also possible to include predefined QS in order to compare different systems of the same application domain. The use of a standardized language for the definition of these QS is necessary as this reduces semantic ambiguities. Only strict semantics of the certified QS avoids misinterpretations. Furthermore the standardization of the QS language eases the automatic processing, which is necessary to guarantee reproducible assessment results.

In the **Selection of Required Evaluation Technique** step, the QS are analyzed and the mandatory evaluation techniques are derived. This step is necessary because there is no evaluation technique available, that can be used for all quality attributes of a system based on its architecture. Even if only one quality attribute is regarded, there are different techniques which differ in the expressiveness of their results, their complexity, and the effort to perform the assessment. There are some quality attributes (e.g. performance) for which evaluation techniques are available that can be performed automatically. Two examples for tool-supported techniques to evaluate the performance of a system are SPE·ED [SW97] and the PCM-Bench [BKR08]. A high ratio of automation throughout the evaluation is expected to produce reproducible evaluation results. However, there are some quality criteria (e.g. maintainability) that still cannot be evaluated automatically. In these cases an evaluation expert is needed which analyzes the architecture based on his knowledge and experience. In order to achieve reproducible and correct results, the evaluation expert should be supported with additional utilities, for example checklists or guidelines. The selection of the adequate utilities for the evaluation expert is also part of this step. The effort for assessing QS can be prohibitive if the statements should be absolutely certainty. Falsification approaches systematically identifying possible counterexamples to invalidate the QS which can then be checked until a certainty threshold is reached. They are used to reduce the complexity although this negatively affects the certainty. The decision, if verification or extensive testing is required is also made in this step.

In the following **Assessment** step, the different evaluation methods are conducted and the results are logged. This step requires a description of the *architecture model* which is either transformed into an input model required by the tool-based and automated evaluation techniques or used by the evaluation expert as base of his analysis. This step additionally requires the already mentioned *component models*. As these models are generally provided by the component developer they have to be certified to guarantee that the model corresponds to the component's implementation. The therefore required component certification is explained in more detail in the next section. The result of this process step is an *Assessment Report* which includes the results of each evaluation.

In the last process step, namely **Certification**, the *assessment report* is checked. It is checked if the evaluation methods indentified in the second step have been performed. The respective assessment results are compared with the quality claims of the software architect, he has formulated as QS in the first step. If the assessment results substantiate these statements the QS are certified. A certificate thereby certifies only the compliance of the software with the quality statements. In order to have a comparable evidence of a system's quality, it is necessary to use QS which were standardized and predefined by the certification authority. The certificate is only valid for the assessed version of the architecture model and the component models and looses its validity if the architecture or components within are changed. Hence, a certificate must contain references to the assessed information.

3.2 Component Certification Process

The focus of the component certification process lies on certifying the correlation of a component's model and the component's implementation. These models are required during the evaluation of the architecture for assessing the QS and reason about the quality of the assembled system. Please note, that there should be at most one model for each quality attribute and in total there can be more than one model. The accuracy information can be seen as a kind of certification level – higher means better investigated but also with a higher assessment effort. This is the reason why the highest level is not always the best, as the reached accuracy has to be weigh against the resulting costs. The complexity of the assessment of the validity of a model for non-trivial components heavily depends on the quality attribute and can be hard or even impossible in practice. For this reason, a falsification approach is used. However, this means that the necessary effort to make a statement with a predefined certainty scales with the assessed degrees of freedom. The whole process for component certification and the participating roles are shown in Figure 4 and explained in the following.

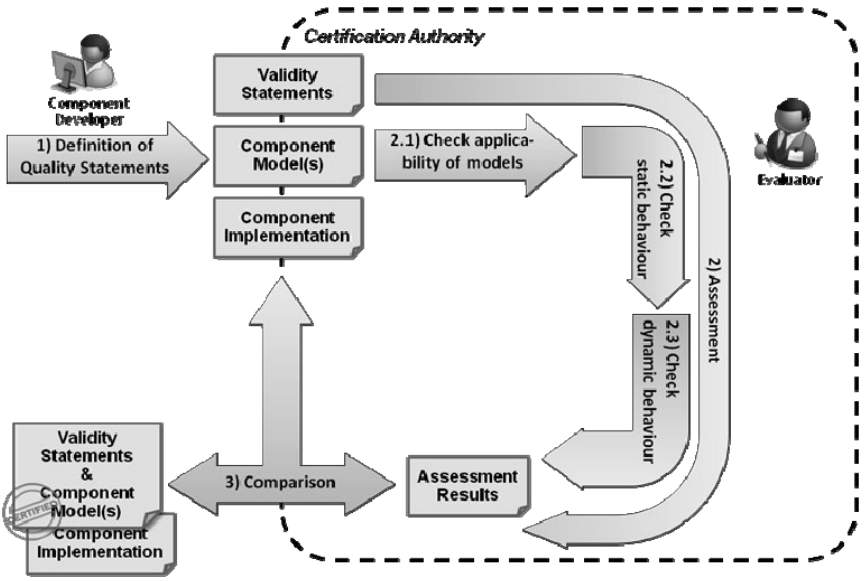

Figure 4: Component Certification Process

In **step 1**, the component developer defines *validity statements* for the component's behaviour *model(s)*. As mentioned above, these *validity statements* specify the range constraints and accuracy which influence the model's validity. For example, a mean response time below 2 seconds can only be provided for less than 10 concurrent users and in a defined environment. The decision which quality attributes should be certified and with which accuracy is the developer's choice, but he can base his decision on common domain-specific standards. The developer has to provide the necessary component model(s), its validity statement, and the implementation of the component to the certification authority.

The assessment in **step 2** can be split into three consecutive process steps. In the beginning in **step 2.1** the *applicability of the models* for the validity statements and the regarded quality attributes is checked. This is especially important with respect to architecture certification as models from different components must be composable to allow analyses on the architectural level. In **step 2.2**, models and implementation are checked in a static context. For example the provided and required interfaces of a component can be checked against their specification stated within the model. However, quality attributes like performance need to be checked in a dynamic context which is assessed in **step 2.3**. The assessment of dynamic behaviour requires a lot more effort for checking, as this is in most cases influenced by the usage profile, component configuration, required components, the used middleware, the operating system, and the hardware environment [BR06].

The costs and effort for provisioning the hardware and software environment for the component scale with the generality of the certificate. This makes test-beds desirable which allow varying these parameters. These test-beds can be realized for example with virtualization or simulation techniques. Software testing approaches [My04; EW01] can be applied for many quality attributes, e.g. performance, to gain confidence in the validity of the model. Most approaches are based on statistical testing, as exhaustive testing requires too much effort for non-trivial components. After step 2 all *assessment results* from the different process steps are available.

In certification **step 3** the decision is made if a certificate can be issued. Therefore the evaluation results are used to assess if the component model(s) are valid abstraction for the *implementation* with respect to the *validity statements*. An issued certificate has to be closely connected to the *component model(s)*, *validity statements*, and of course to the *implementation*. A packaging of model(s) and validity statements which only has a reference to the implementation is reasonable, as these are the interesting artefacts for architectural analyses. If models are proven to be invalid the software developer will not receive a certificate, but should receive feedback about the points of failure.

4 Conclusion and Outlook

This paper provided a short overview about software certification and software industrialization. It additionally pointed out the necessity of software certification for software industrialization. The certification approach sketched in this paper enables to certify the correctness of quality statements concerning quality attributes of a software systems. Based on this foundation, the need for a partition of software certification processes according to the certification on architecture and component level was presented.

The presented certification process allows the usage of user-defined just as well as standardised quality statements. This allows on the one hand using the certification as individual quality check. On the other hand the certification process provides a tool to compare different software systems. The approach certifies a software system based on its architecture and models of the included components. Thus, the certification can help in an early development stage to assess the quality of an assembled system although some components are still not implemented yet (but described in models).

In addition, the approach forms a framework which allows the integration of different software evaluation methods into one common quality certification process. The approach is thereby not limited to certain quality attributes and can be used to certify single quality attribute as well as a combination of different ones.

As future work we plan to develop a tool supported performance certificate. As a first step a language is developed which allows the definition of performance relevant quality statements. In parallel, we develop a test-framework for performance-models of components.

References

[BCK99] Bass, L.; Clements, P.; Kazman, R.:. Software architecture in practice. Addison-Wesley ; Bonn, 1999.

[BKR08] Becker, S.; Koziolek, H.; Reussner; R.: The Palladio Component Model for Model-Driven Performance Prediction. Journal of Systems and Software, To appear, 2008.

[BR06] Becker, S.; Reussner, R.: The Impact of Software Component Adaptation on Quality of Service Properties. Lóbjet, 12(1):105–125, 2006.

[Cc07] CCRA Members. The Common Criteria v3.1, 2007. http://www.commoncriteriaportal.org.

[EW01] El-Far, I.K.; Whittaker, J.A.: Model-Based Software Testing. In J. J. Marciniak, editor, Encyclopedia of Software Engineering. Wiley, 2001.

[In00] International Organization for Standardization. ISO 9001:2000, 2000.

[KBS06] Krafzig, D.; Banke, K.; Slama, D.: Enterprise SOA. Prentice Hall PTR, reprint. edition, 2006.

[Kn06] Kneuper, R.: CMMI. dpunkt, 2nd edition, 2006.

[MW08] Maibaum, T.; Wassyng, A.: A Product-Focused Approach to Software Certification. Computer, 41(2):91–93, Feb 2008.

[My04] Myers, G.J.: The Art of Software Testing. Wiley, Hoboken, NJ, 2nd edition, 2004.

[SGM02] Szyperski, C.; Gruntz, D.; Murer, S.: Component Software: Beyond Object-Oriented Programming. ACM Press and Addison-Wesley, NY, 2002.

[SW97] Smith, C.; Williams, L.: Performance engineering evaluation of object-oriented systems with SPE·ED. In Computer Performance Evaluation Modelling Techniques and Tools, pages 135–154, 1997.

[Ta05] Taubner, D.: Software-Industrialisierung. Informatik Spektrum, 28(4):292–296, 2005.

[Vo99] Voas, J.: Certification: reducing the hidden costs of poor quality. Software*, IEEE*, 16(4):22–25, 1999.