CERTSOFT06

First International Workshop on Software Certification

A satellite event of FM2006

McMaster University, August 26–27, 2006

Stefania Gnesi[1] Tom Maibaum[2] Alan Wassyng[2] (eds.)

[1] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Area della Ricerca CNR, Pisa, Italy

[2] Department of Computing and Software
McMaster University, Hamilton, Ontario, Canada

SQRL Report No. 37

Preface

This volume contains the proceedings of CERTSOFT06, the First International Workshop on Software Certification, organized at McMaster University, Hamilton, Canada, in August 2006. The workshop is co-located with the 14th International Symposium on Formal Methods.

Software is used to control medical devices, automobiles, aircraft, manufacturing plants, nuclear generating stations, space exploration systems, elevators, electric motors, automated trains, banking transactions, telecommunications devices and a growing number of devices in industry and in our homes. Software is also mission critical for many organisations, even if the software does not "control" what happens. Clearly, many of these systems have the potential to cause physical harm if they malfunction. Even if they do not cause physical harm, their malfunctions are capable of causing financial and political chaos. Currently there is no consistent regulation of software, and society is starting to demand that software used in critical systems must meet minimum safety, security and reliability standards. Manufacturers of these systems are in the unenviable position of not having any clear guidelines as to what may be regarded as acceptable standards in these situations. Software producers and their customers are becoming interested in methods for assuring quality that may result in software supplied with guarantees. Just recently, the U.S. government has expressed significant concern regarding the potential for large scale economic disruption caused by computer security flaws.

CERTSOFT06 has been organised to address the issues raised by certification of software, to provide a forum for discussion for interested parties and to help organise the community of researchers and software professionals interested in these issues.

Three distinguished speakers were selected to provide direction to the workshop:

Rance Cleaveland, Professor of Computer Science, University of Maryland, and Executive and Scientific Directory of the Fraunhofer USA Center for Experimental Software Engineering:

"Does Certification = Verification? Formal Methods And Software Certification"

Glenn Archinoff, President of Candesco Corporation:

"Real-Time Software Certification in the Nuclear Power Industry"

Dave Parnas, SFI Fellow, Professor of Software Engineering, Director of the Software Quality Research Laboratory of the University of Limerick:

"Certification of Software: What, How, and How Confidently"

We would like to thank the three invited speakers for the abstracts they contributed to these proceedings and for their participation.

We also have several interesting contributed papers, some on more general aspects of certification, as well as some on more technical issues related to the topic.

We hope that this workshop is just the first in a series, and will have a continuing association with the Formal Methods Symposium in years to come.

We would like to express our appreciation to all those who helped in the organization of this workshop: the Program Committee, listed below, who reviewed submitted papers, and the Organizing Committee, consisting of Alan Wassyng, Wolfram Kahl, Mark Lawford, and Jeff Zucker, who helped plan, advertise, compile the Proceedings and run the workshop. We would also like to thank Andrei Voronkov for the use of EasyChair, which was, indeed, easy to use. We thank the Faculty of Engineering, McMaster University, for financial support. Finally, we must thank Emil Sekerinski, who not only took on the difficult task of General Chair for FM06, but still had time to concern himself with details of CERTSOFT.

Stefania Gnesi & Tom Maibaum
Program Committee Co-Chairs

Alan Wassyng
Organizing Committee Chair

Program Committee

Stefania Gnesi (Co-Chair)	ISTI-CNR, Italy
Tom Maibaum (Co-Chair)	McMaster University, Canada
Rance Cleaveland	University of Maryland, USA
Alessandro Fantechi	University of Florence, Italy
Jan Friso Groote	Eindhoven University of Technology, The Netherlands
Connie Heitmeyer	Naval Research Laboratory, USA
Paola Inverardi	University of L'Aquila, Italy
S. Purushothaman Iyer	North Carolina State University, USA
Yoshiki Kinoshita	CVS-AIST, Japan
Dino Mandrioli	Politecnico di Milano, Italy
Jonathan Ostroff	York University, Canada
Natarajan Shankar	SRI International, USA
David von Oheimb	Siemens AG, Germany

Contents

Does Certification = Verification?
Formal Methods And Software Certification

(Abstract of Invited Talk)

Rance Cleaveland

Department of Computer Science
and Fraunhofer USA Center for Experimental Software Engineering
University of Maryland

The growing importance of software in safety-critical systems has led governmental agencies and standards bodies to devise regulations and guidelines on the development and deployment of such software. The underlying motivation for these forms of guidance may be summarized as follows: "Is the software sufficiently likely to behave correctly?" The resulting certification standards have tended, however, to focus on document-able proxies for absolute software correctness, such as development and V&V processes.

Formal methods is also devoted to the question of software correctness, although they are not mentioned in typical certification standards. This talk is devoted to exploring why this is the case and to suggesting roles that formal methods might usefully play in certification regimes.

Real-Time Software Certification in the Nuclear Power Industry

(Abstract of Invited Talk)

G.H. Archinoff, D.K. Lau, E.D. Houldin

Candesco Corporation, Toronto, Ontario, Canada

Computer control has been a significant part of the Canadian nuclear power industry since its inception. The Pickering A nuclear power plant just east of Toronto began operation in the early 1970's, and uses a dual redundant computer system to control major functions such as bulk reactor power and the spatial power distribution. More recent nuclear plants have taken advantage of advances in computer hardware and software technology, and use computers in both control and safety functions.

The Canadian nuclear industry has been a leader in developing and applying methods to ensure the correctness of software used in applications that require ultra-high confidence in computer systems. This presentation summarizes the development of these methods and how they are deployed in current designs. The presentation describes the process used to categorize the safety significance of software based on its required functions relative to the overall safety of the plant, discusses methods used to qualify embedded software in systems or components supplied by third parties, and describes methods used to certify software developed for safety-critical applications. The presentation also covers regulatory oversight and approval of computer-based, real-time control and safety systems in the Canadian nuclear power industry.

Certification of Software: What, How, and How Confidently

(Abstract of Invited Talk)

David Lorge Parnas

Software Quality Research Laboratory,
Department of Computer Science and Information Systems
Faculty of Informatics and Electronics
University of Limerick, Limerick, Ireland

and

Department of Computing and Software, Faculty of Engineering
McMaster University, Hamilton, Ontario, Canada

Where other products have a warranty, software carries a disclaimer. In spite of the widespread use of software in all aspects of our lives, there are still computer experts who are afraid to trust software for such simple tasks as counting votes. One of the signs of a maturing discipline will be software that comes with a document certifying that it is fit for its intended use.

This talk deals with some simple questions about certification.

- What exactly can we certify in the certificate?
- How can we be confident that the certificate is accurate?
- How can we measure our confidence?
- What can the designers do to increase our confidence?
- What can the designers do to decrease the expected damage if we are wrong?

Basic Concepts of Software Certification

Fabrizio Fabbrini, Mario Fusani, Giuseppe Lami
Systems and Software Evaluation Center
"Istituto di Scienza e Tecnologie dell'Informazione"
National Research Council
Pisa, Italy

Abstract. *This paper examines the subject of software certification together with various related issues through a question/answer paradigm. The general concepts of certification and certification schemes, developed over twenty-years of working with the Evaluation Center, are firstly introduced, and then certification problems to do with software technology (products, processes) are commented on. The paper aims to assure the reader that certification schemes can be applied to software, provided their current limitations and expected results are clearly understood.*

1. A tough concept

Certification is one of those concepts often talked about but which few appreciate all the rather grey areas in which it is used. The meaning that has been given to the term 'certification' is broadly varied and depends on culture, interest and experience. Positive and negative things have been said and written about certification [18], [22], and sometimes with good reason from both perspectives. Furthermore, when it comes to software processes and products, the topic can become really confusing and difficult. Apart from when it is associated with quality management systems (typically the case of ISO 9000 certification), the term, with all its derivatives, has for years even been banished from commercial advertisements and technical literature. The main reason for this is the big concern by suppliers that its use would be linked to such terrifying words like "guarantee" and "liability".

Now many things like certification, certificate, certification scheme for products and processes are being "discovered", even in the software domain, and a business has sprung up around them.

This paper aims, if not to clarify all doubts, at least to identify the main issues related to the use of certification, typically software certification, in the hope that readers will regard it more confidently. We will not discuss either certification of quality management systems or certification of professional skills, but only mention some of their relationships with other aspects.

2. Background

What is certification? This is obviously the first question on the list. Many people of various professional and non-professional backgrounds, use the term, all perhaps attaching a different meaning to it. Definitions, as we will see, in turn contain other expressions that can be interpreted in different ways. We will start by mentioning one definition, taken from ISO [10]:

'*a procedure by which a third party gives written assurance that a product, process or service conforms to specified requirements*'

There are several things that need to be clarified here or at least commented on. 'Third party' seems to be an actor in the 'procedure', and 'assurance', 'conforms', 'requirements', 'product', 'process' need some reflections. A discussion on these concepts follows below.

Such 'assurance' can be given as a result of an activity, 'conformity assessment', defined in the same Guide but perfected by the ISO/IEC DIS 17000 [9] as follows:

'*an activity that provides demonstration that specified requirements relating to a product, process, system, person or body are fulfilled'*.

Notice that nothing like a guarantee is wanted. In software technology, the word 'guarantee' has not been welcomed ever since this standard was defined (in actual fact, vendors have generated a good deal of literature on the use of disclaimers). In most technologies, including software, such a demonstration, intended as 'confidence' and not as 'proof', is something one can try to achieve and in many cases even be successful. This, as we mention later, can work as an added value to products and processes. Yet, being allowed to use 'proof' in a mathematical sense would be much better, but the boundaries within which a formally provable statement, with all the necessary math to get to the final statement, would generally be too restricted. In many cases, links to the everyday world (such as requirements texts and test scripts) would be necessary, and for these links there is no proof, only "confidence". For example, a formal proof could be used, and only in some cases, to certify that a product has such a property as "correctness". However if the property to be certified is "maintainability", we would most likely need to abandon going down the line of proofs and have less rigorous but nevertheless workable alternatives, such as informal, checklist-aided analyses and tests.

We will now proceed with our discussion using a question and ansser format. This wil draw from the state of the art,. from the widely accepted experience of international organizations, and from a lot of practice performed, successfully or otherwise, in the area. The discussion will also define and use some terms included in the above definitions.

3. Questions related to certification

What actors are associated with certification and how do they behave?

What process should be run by a Certification Body?

What types of things are certified?

What is the added value of certification?

Who benefits from certification?

What if software is an object of certification?

The questions above can be better understood by outlining possible scenarios where certification, even software certification, has become workable and recognized. Most of the answers apply to various technological areas, including software. Software requires

a little more attention. For commonly recognized requirements, we mostly refer to the ISO standard structure (although this is not strictly necessary for our purposes), as the ISO has developed a number of rules for certification and accreditation systems and schemes, and because it still counts on a significant amount of consensus worldwide.

3.1 What actors are associated with certification and how do they behave?

This is a rather complex aspect and can be quite boring when dealt with in detail, so we will just outline the essential points, bearing in mind that the entities and their relationships can vary depending on consensus and on other opportunities.

A *certification body* is an organism with internal rules, human/infrastructure resources and skills to perform certification procedures. To be able to compare the results of certification and then get a broad consensus, the internal rules themselves might need to conform to defined standards. In such cases, certification bodies can be 'accredited,' that is, declared capable of performing certification, upon periodical surveillance, by so-called *accreditation bodies*. This is compatible with the above definition of conformity assessment.

Strictly speaking, a certification body does not need to be accredited. However, accreditation is important to increase the value of the certification, and thus the value of the certified object. We will return to the concept of this "value" later. In any case it is understandable that having an object certified by an organism that conforms to continuously verified standard rules, gives greater confidence in the service associated with the object than in the case when the organism behaves unpredictably.

How do accreditation bodies themselves behave? Who is 'accrediting' the accreditation bodies? Well, in actual fact there are not many accreditation bodies (the average is one-two per country, compared to tens or hundreds of certification bodies, generally specialized per product category). This means that they can accredit each other by executing periodical conformity assessments typically 'peer reviews' upon their activities. Figure 1, which should be read starting from the square box in the center, depicts this scenario.

Requirements, as mentioned in the definitions given for certification, may be expressed in terms of standards (rounded rectangles at the bottom). Again, this is aimed at increasing confidence in certification, since standards are designed to be commonly accepted reference models, allowing certification bodies to express comparable and repeatable results. This in turn should facilitate the circulation of goods, and foster commercial co-operation with mutual acceptance of the certification results within the international trading framework.

What is usually certified (rectangles on the right, see also Section 3.2) is *processes*, *products* (materials may also be considered as products), *people* or *organizations*. To be more precise, in the same way as a measurement is a statement (for instance, a mapping to integer numbers) about an *attribute* of an object, certification often refers to properties or attributes of objects: So, we have certification of attributes (e.g. electrical properties) of a product, of attributes (e.g. capability) of a process, of professionals skills , and of organization quality systems .

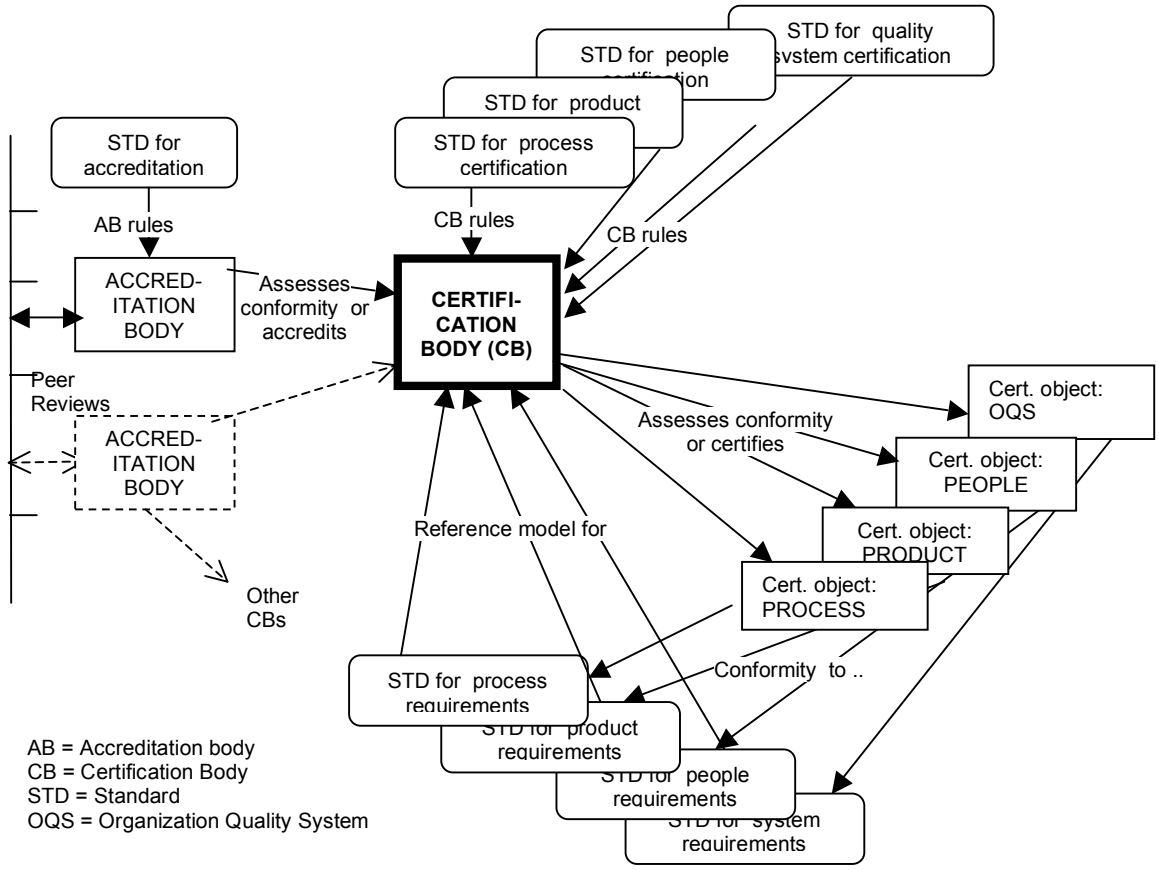

Figure 1 - Certification and accreditation scenario (simplified)

In order to assess conformity in a repeatable and documented way, a certification body must follow a defined process, and it is important that all the certification bodies follow the same rules for the same object types. Again, widely recognized standards for the assessment process (rectangles at the top) could give such confidence. Figure 1 represents a somewhat simplified scenario (yet more complex for some self-declared existing certification schemes). More entities, such as testing laboratories and scheme-managing organizations, and more standards, such as operative assessment models, should be added, together with their relationships to the existing elements. However what we have shown here is sufficient to give an idea of how a certification schema could be run quite satisfactorily.

We are not contradicting our definition of certification if we consider an even simpler scheme, like the one shown in Figure 2. Here the supplier of the object also acts as a Certification Body for the same object. In general, the users of the service associated with the object would have less confidence than in the case of third-party certification. This is because there is no evidence that the supplier adopts a defined, controlled and verified process to produce and certify the object. Yet various distinguished suppliers do this kind of certification, some of them producing a certification policy as well.

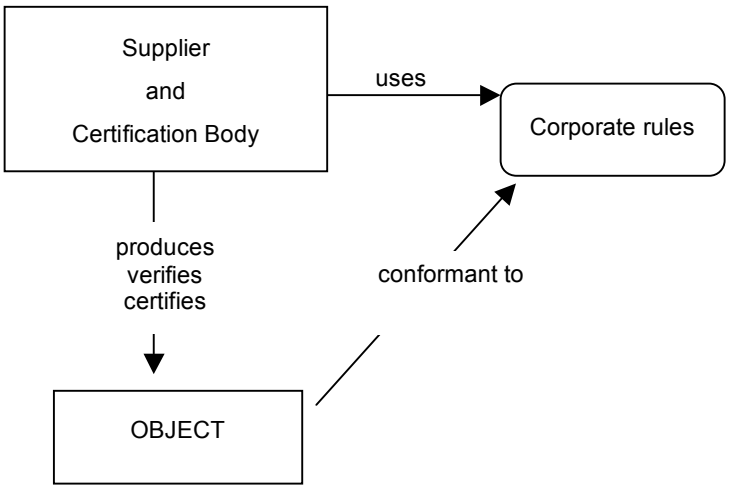

Figure 2 - Simpler Certification scheme

3.2 Which process should be run by a Certification Body?

Conformity assessment is a process which includes, but is not limited to, testing and analysis of the objects. For example, according to many product certification standards (such as those denoted in Figure 1 as rule providers for the certification bodies), certification includes: a) conformance assessment (verifying product conformance to product requirement standards); b) inspection and surveillance of the quality management system of the manufacturer; c) verification and surveillance of projects and production processes. For example, for non-software technologies, tests are performed first on prototypes, then on samples taken from production.

Tests are normally executed by the supplier / developer, or by independent testing laboratories. The certification body will use testing process evidence (plans, procedures, reports) in the certification process. Testing laboratories themselves (corporate or independent) can be accredited according to opportune accreditation rules or standards by special accreditation bodies (usually different from those that accredit certification bodies). In software technology, it is best to have the product directly assessed (reviewed and tested) by the developer, possibly by means of automatic conformity assessment procedures and tools.

The certification scenario may not always be based on voluntary rules (rules that match overall consensus). When public health, safety or the environment are at stake, conformity certification is usually made obligatory by Government regulations. Without official assessment and approval by an accredited certification body, goods are barred from sale, or suppliers are not accepted as vendors for public administrations.

The scenario can be more complex when the so-called *competent organisms* are involved: usually they act as certification bodies and testing laboratories, and are accredited by government entities (which, alas, do not necessarily follow accreditation standards, nor undertake peer reviews). We will not take the latter into account, as what we have said so far is enough to understand the other aspects in our question/answer list.

3.3 What can be certified?

As mentioned, strictly speaking what are certified, i.e. the object of certification, are not exactly products, processes, or people, but some of their properties, through verification of conformance to one or more defined requirement standard. For our purposes here we refer to the objects or to their properties, indifferently.

Not all the standards would be well suited for certification. This is because ISO/IEC Guide 7 [11] criteria are established both for standards to be used as rules and guidance for certification bodies and some criteria about requirement standards for the target objects. The same guide also mentions criteria for another set of rules that may be conceived and established as a standard, that is, methods for assessing the conformance, depending on the application domain, which may include analyses, tests, extended surveillance over quality management systems, projects, production processes. So various types of standards may be involved in a certification scheme. A typical situation is shown in Figure 3, where some elements are the same as those shown in Figure 1.

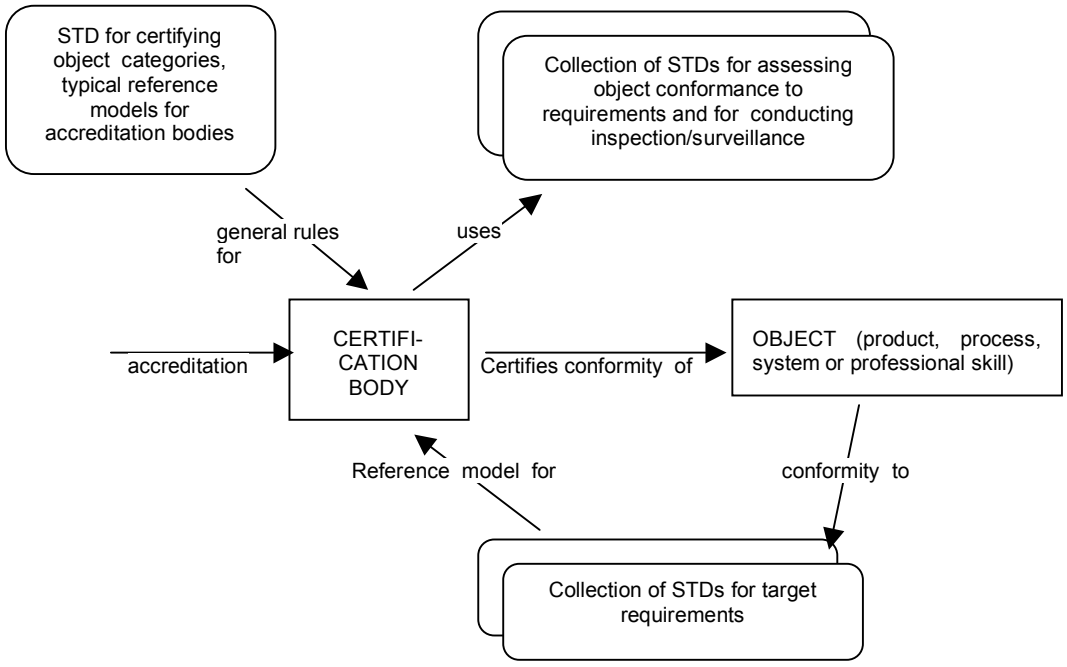

Figure 3 – Role of standards in a certification scheme

A scheme usually includes agreements among suppliers, customers, standard providers, as well as means to identify and manage certified targets, such as certificates (typical outputs of the certification process), marks and usage policies, and also includes liability issues (see for example ISO/IEC Guide 44 [12], or ISO/IEC 17011, "Standard for accreditation of conformity assessment" [17])

People certification is not the focus of this paper, yet it deserves a few words. There is much advertising relating to the certification of professional competence (e.g. PC experts, Windows operating system experts), but the related schemes are mostly unclear (as are many things to do with certification). What may be interesting here is the fact

that skilled people may be essential in order to perform conformity assessments (again to increase consensus about the results of certification). This is linked to our scheme in the requirements expressed by the formal rules for certification bodies (rounded rectangles on the left and top right in Figure 3). The standards in such adverts specify that and how the conformity assessors must be instructed through certified training courses and certified themselves by appropriate accredited certification bodies, according to specific requirement standards.

3.4 What is the added value of certification and accreditation?

Let us consider a product that a customer wants to buy. If it is certified within a certification scheme and if known and reliable certification and accreditation bodies operate in the scheme, then this product may be considered as having "added value" with respect to a similar uncertified one. Whether the certified product is worth the inevitable higher price or not depends on the value the customer gives to the requirement standards the product is supposed to be conformant to. There is then the question of trusting the certificate.

One way to decide whether a certificate can be trusted is to check if a *certification policy* is available with the certified object. Such a policy should briefly and clearly provide information about the scheme and the standards. In the case of a certified product, it may come together with a visible mark. A certification policy also contains rules or restrictions on the use of the certificate.

Again, the added value is usually based on consensus and trust.

It is also possible that a scheme is a mandatory one (it's the law!) for the object to conform to a standard. In this case it may not be meaningful to speak of added value, but it is very important to check for the presence of the policy mentioned above. There can be legally marked goods based only on testing and/or on the supplier's declaration of conformity, even to safety standards. Evaluating associated risks entails checking for a certification scheme that the mark is associated with.

Added value does not usually imply any proof or guarantee, but just 'enhanced confidence'. Yet, in the expectation of the supplier of the service associated with the certified object, greater confidence in the service is probably why a customer prefers a certified good or service to a similar, uncertified one. Unfortunately, there is no objective criterion to determine if such confidence is rightly associated with the higher selling price.

3.5 Who benefits from certification?

Certification may be suitable both for responsible manufacturers and service providers, and having their products assessed and certified as conformant to a commonly accepted international standard allows them to distinguish themselves from less reputable suppliers. Customers would generally be free (in principle) to choose their goods and services. As explained above, they can benefit when they experience good, affordable added value.

As we strive for quality in many goods and services, certification should in theory lead to better quality. This may be true when requirement standards include quality, and the certification scheme is operated accordingly.

3.6 What if software is an object of certification?

Although the so-called software crisis is over according to many academic and advanced technical centers, we nevertheless have some reservation about this. In fact, statistics [21] show that project percentage overrun has improved (decreased) but still exists. Some things have changed (for example, [3]), but not that much, since the famous essay by Fredrick Brooks [2]. New application fields (e.g., vehicle control) have been invaded by software technology and in such fields single cases may count, not just average data. If demand for more confidence that software be suitable, during exercise, to end user's needs would turn in demand for more certification in software products/services, then software certification is going to be taken seriously.

But how would that be possible? To conduct a complete survey, we would have to examine the existence of reference standards for the four object categories mentioned, as well as standards for assessing conformity for each category, plus standards providing rules for the bodies, and discuss their suitability in a certification scheme. Here we limit our discussion to the following types:

1. **requirements standards** for software **products** [lower right rectangles in Figure 1]
2. **requirements standards** for software **processes** [lower right rectangles in Figure 1]
3. standards enabling a certification body to **assess conformance** (by using techniques for direct / indirect testing, analyses, inspections, quality management system and process surveillance) of products or processes to type 1 and type 2 requirements [rounded rectangles in right top of Figure 3]
4. standards usable as **internal rules by certification bodies** for software (product, process) targets [rounded rectangles in left top of Figure 3]

To these, we should add a) standards for accreditation of software certification bodies; b) standards for accreditation of software testing/evaluation laboratories; c) standards for using test execution techniques and other evaluation processes. Here we will just focus on the four types listed above.

Even if all these had workable certification schemes, suppliers, customers, and users would still only gain in terms of confidence.

3.6.1 Requirements standards in a product certification scheme

Standards of type 1 may vary widely depending on what characteristics of a software product or process we are looking for.

3.6.1.1 Functional standards

Functional standards can be expressed as customer expressions of what a system depending on software is expected to do in defined working conditions. Usually such expressions are denoted as System and Software Requirements Specifications. In this case we should not speak of standards, because requirements change for each software project and implementationInstead we should talk in terms of Verification and Validation of software (against the Software Requirements and other project requirements).

With regard to functional standards, the conformity assessment by a typical third-party certification body would technically be possible, but: i) much doubt remains about the

'degree' of conformity; ii) it would be rather costly (see Section 3.6.3); iii) requirements stated by a customer are typically affected by incompleteness, ambiguity and continuous changes (this last characteristic is not a defect, but makes certification impractical); iv) certification of compilers, operating systems, libraries should also be performed but in most cases this cannot be done. Consequently good certification, although not impossible, is very costly and complicated and in any case would not add much value to the product.

Particular application domain functional standards may be 'true' standards (that is, be a common reference) and represent a more accessible way to product certification. In this case, requirements can be defined thoroughly and are comparatively stable. Test suites would be defined for the product, with strict pass/fail criteria. However, so far this has been done only for compilers, communication protocols and graphic kernels. There is not much work for certification bodies because of: i) incomplete functional/structural coverage for test suites for compilers; ii) ISO/OSI lower levels replaced by TCP-IP protocols; iii) too many options in protocol definition; iv) evolution of GUIs towards higher flexibility. Corporations and associations, such as accountants, once in a while try to launch functional standards, but as far as we know there are no stable certification schemes.

Thus there is little hope for product certification usage with functional requirements and standards, in spite of the rather high demand.

3.6.1.2 Quality standards

Much work has been done regarding software product quality definition and standardization. Families of standard have been proposed as reference models for software product quality (such as ISO/IEC 9126 [7]). They could be used to express quality requirements for conformity assessments if reference or target 'values' for quality characteristics could be given for the objects of certification. Although a standardized way of measuring such characteristics has been proposed [7], [8], it hasn't been proved valid in a certification scheme. There are still no really useful solutions, and progress has been quite slow. One related gap is that the relationship between software development process and product qualities is not yet well understood so, from a research point of view, the area is still attractive [1][23],

For particular product characteristics, such as security and safety (i.e. a *system* whose behavior depends on software rather than the software itself), standards are being proposed which cover both product and process aspects.

Regarding security, ITSEC first and then Common Criteria (CC) [http://www.commoncriteria.org] have been interesting references, and several related *corporate certification schemes*, related to specific application domains, are being proposed and advertised. Until recently they worked as self-accredited bodies but, after growing demand from the market and from public areas, special accreditation bodies have been created in many countries. The business flourishing in the field would seem to indicate that it will be the most active as far as certification is concerned.

Regarding safety, application-domain-related authorities (e.g. for medical equipment, nuclear plants, transportation areas) have also been acting as standardizing and certification bodies. Related standards are usually a collection of elements from types 1-

4 above. There is demand for safety certification, yet it is not as pressing as it is for security.

Coding standards have also been proposed (see for example MISRA [http://www.misra.org.uk/]), which would, at least formally, be well suited to a certification scheme, but the little added value provided hardly justifies the effort.

3.6.2 Requirements standards in a process certification scheme

We will just mention three examples, which can be easily found in the related literature.

The Capability Maturity Model (CMM) developed and used by SEI (see for example [19]) is a remarkable example of a well-established, successful certification scheme with self-accreditation (Figure 2). Although the term 'certification' is not welcome in the providers' area, the related activities essentially fit the definition given in Section 2 (a preferred expression is a declaration by SEI that includes a statement such as: "…achieving Maturity Level *n* by satisfying Key Process Areas *x, y*, …"). The object is a group of processes enacted by an organizational unit for developing software and systems dependent on software, and the conformity to the reference requirements standard is measured by a well-known capability maturity scale. However, the reference requirements standard itself is not expressed as a public standard, but as a set of software engineering practices, related to key processes, and maintained by the 'certification body' itself (SEI). Accreditation is achieved by worldwide popular consensus. To sustain confidence about certificates, SEI also publishes studies and statistical reports about correlation between measured maturity levels and achievements obtained by the organizational units that have been assessed [20].

ISO/IEC 15504 (Also known as SPICE) published most of its International Standards (IS) in 2003, 2004 [15] and 2006 [16]. It invokes a reference model for the so-called 'software processes dimension', which is a standard itself (ISO/IEC 12207 [5]). This might play the role of requirements reference standard in a process certification scheme. Unlike CMM, objects of ISO/IEC 15504 are single processes, whose description requirements are defined in the process reference model and, like CMM, each process is assessed against a defined capability level model. ISO/IEC 15504 has a public IS (and a public Guide) for conformity assessment [14], so it would fit better in a certification scheme (see Figure 3), but it explicitly forbids its own usage for certification. There doesn't seem to be any technical reason for this, because surveillance, for instance, could be replaced by multiple or continuous assessments. Other initiatives have appeared, built on SPICE, probably to overcome the obstacle of formally banishing its use in a certification framework. One of them, Automotive SPICE, is still being developed (http://www.automotivespice.com/).

IEEE standards are also very popular and have been adopted worldwide. There is a complete family that includes all aspects of a software lifecycle (see complete series at http://standards.ieee.org/catalog/olis/se.html). They might be used in a certification scheme. Reference requirements are basically expressed in terms of documents and document requirements. Problems with confidence in certification results may derive from the fact that standardized documents might not exactly reflect the execution of processes.

Other more specific standard families may be found that cover various application areas (such as web services or e-learning domains)..

3.6.3 Standards for software conformity assessment

There are many ways in which a body executes conformity assessment activity.

As far as software products are concerned, there is a collection of reference standards to examine various product instances and aspects: standard test suites for functional testing, ISO/IEC 14598 [6] for quality characteristics and various analyzing tools for code requirements.

Regarding customer's defined requirements, as we said certification is difficult, but conformity could be assessed by examining documented software engineering practices employed by the developer, including testing. Independent testing is generally quite costly and not easy. Standards are difficult to choose but, for instance, the IEEE set for software processes (see below) can be adopted. This is the area where there would probably be much demand for certification (see alsothe first part of Section 3.6.1.1).

As far as software processes are concerned, the CMM and ISO/IEC 15504 (see Section 3.6.2), have their own reference standards (one private, the other public [14]) for assessing conformity to requirements for software processes.

It is also worth noting that there is no clear separation between the four types of standards:

1) requirement reference for products;
2) conformity assessment for products;
3) requirement references for processes;
4) conformity assessment for processes

, thus software certification objects, activities and objectives are still frequently confused.

3.6.4 Rules for certification bodies

Related standards were expressly designed for certification purposes. They are more independent of the application domains, and include the ISO Guides and some of the EN standards (for example, for products, see [4] and its evolution into the ISO/IEC 17000 family [9]).

4. Conclusions

The concepts presented in this paper are only a small part of the various issues involved in certification and software certification. We have not mentioned, for example, certification/accreditation body liability, legal aspects, assessor training and responsibilities, insurance, and mutual recognition of certificates issued by different bodies. Another aspect to investigate in more detail regards the techniques used by Certification Bodies (typically in an indirect way, through a testing organism), in particular the relationship between software testing/verification/analysis and certification, in the various scenarios outlined above, including adopting automated certification procedures and tools. How adopting such techniques would change the balance value/cost of certification may be the subject of future research. We have tried to highlight some issues, problems and common beliefs, and lay the basis for a classification. We have also commented on the various types of standards, and pointed out their suitability for software certification.

We have discussed a set of questions rather than giving complete answers which is often just not possible. Other questions might be:

Why can't certification be based on a "snapshot" rather than requiring periodical surveillance?

Why are good verification techniques not sufficient?

Is certification without "proof" meaningful?

Which actors and entities are liable with regard to the conformity of the object to its requirements standards, and to what extent?

We leave such questions to future research. There is also much more literature than we have had space to cover in Section 4, especially as far as myths and mistakes about certification are concerned.

4. References

[1] Bollinger, Terry; Voas, Jeffrey; Boasson, Marten. "Persistent Software Attributes." *IEEE Software*, IEEE Computer Society, November/December 2004

[2] Brooks, F. P., The Mythical Man-Month: Essays on Software Engineering . Addison-Wesley, Reading, MA, 1975.

[3] Glass, R., Software Runaways: Lessons Learned from Massive Software Project Failures, Prentice Hall, 1998

[4] ISO/CASCO 228:1994, General Requirements for Bodies Operating Product Certification Syatems

[5] ISO/IEC 12207: 1995/FDAM, Information technology – Software life cycle processes

[6] ISO/1EC 14598-1: 1999, Information technology - Software product evaluation - Part 1: General overview

[7] ISO/IEC 9126-1: 2001. Software engineering- Product quality- Part 1: Quality model

[8] ISO/IEC CD 25000.2:2003, Software and Systems Engineering - Software product quality requirements and evaluation ': SQuaRE - Guide to SQuaRE

[9] ISO/IEC DIS 17000: 2004, ISO/IEC 17000:2004, Conformity assessment - Vocabulary and general principles

[10] ISO/IEC Guide 2:1996, Standardization and related activities – General vocabulary

[11] ISO/IEC Guide 7:1994, Guidelines for drafting of standards suitable for use for conformity assessment

[12] ISO/IEC Guide 44:1985, General rules for ISO or IEC international third-party certification – schemes for products

[13] ISO/IEC Guide 62:1996,

[14] ISO/IEC 15504-2:2003 - Information technology -- Process assessment -- Part 2: Performing an assessment

[15] ISO/IEC 15504-1:2004 - Information technology -- Process assessment -- Part 1: Concepts and vocabulary

[16] ISO/IEC 15504-5:2004 - Information technology -- Process assessment -- Part 5: An Assessment Model and Indicator Guidance

[17] ISO/IEC 17011, “Standard for accreditation of conformity assessment”, 2005

[18] Kolawa, A., Certification Will Do More Harm than Good, IEEE Computer, June 2002, pp. 34-35

[19] Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. B., "Capability Maturity Model, Version 1.1", IEEE Software, Vol. 10, No. 4, July 1993, pp. 18-27

[20] SEI, Software Engineering Institute - Process Maturity Profile - CMMI® v1.1 SCAMPISM v1.1 Appraisal Results, 2003 Year End Update, March 2004

[21] Standish Group Chaos Report Press Release, 2003

[22] Tripp, L. L., Benefits of Certification, IEEE Computer, June 2002, pp. 32-33

[23] Voas, Jeffrey. “Software’s Secret Sauce: The “-ilities”” IEEE Software, IEEE Computer Society, November/December 2004.

A Maturity Model for Software Product Certification

Petra M. Heck

LaQuSo, Laboratory for Quality Software,
an activity of Technische Universiteit Eindhoven and Radboud Universiteit Nijmegen
Den Dolech 2, P.O. Box 513
5600 MB Eindhoven, The Netherlands
+31 (0)40 247 2526
p.m.heck@laquso.com

ABSTRACT

In this paper we describe a framework for assessment of software maturity, called the Software Product Maturity Model. This Maturity Model expresses that the confidence in the absence of faults in a software product increases when formal methods are used for verification of the software product.

Maturity is, as defined in [11], the capability of the software product to avoid failure as a result of faults in the software. From the definition it follows that a software product without faults is the best guarantee for maturity. The absence of faults we call *correctness*. In our maturity model we cater for an objective evaluation of the correctness of the product. Establishing that faults are absent is also the main topic in certification of (critical) systems, which means our model can be of use in this field.

We do not only consider the maturity of the final software product (the source code or working system), but also of intermediate deliverables like requirements and detailed design.

In the spirit of the well-known CMMI models we define Product Areas (main product deliverables), Goals per area and Properties per goal. The scores on these properties indicate the Maturity Level of the product or product area.

Our Maturity Model is unique in its focus on the product in all its phases and on correctness. It establishes a standard to perform software certification that also includes e.g. expert reviews and formal verification if necessary. It uniformly establishes what to check and how to check it. The checks are not new, but there is no place yet were they all have been put together into one model.

Keywords

Software Product Maturity, Correctness, Consistency, Software Product Certification, Maturity Levels

1. INTRODUCTION

Software product certification evaluates a software product for its quality. Often, when a contractor needs information on the product quality, she will use an audit on the development processes. Well-known models in the process assessment area are CMMI [3] and ISO 15504 (SPICE) [13]. These models prescribe a set of activities that have to be present in the software development process to reach a certain level of maturity. The underlying assumption is that sufficiently rigorous development processes will also produce products of desired quality.

The assumption is, generally speaking, unjustified. Following an exact recipe can still lead to a meal that tastes awful if, for instance, low-quality ingredients are added. In the same way a product's quality can turn out to be low, although the prescribed procedures have been followed. If too many faults were introduced in the coding phase, even the amount of peer review and testing prescribed by process models will not catch all of them.

Hence, there is need for a standard that is based on the software product itself and not on the processes. Software certification verifies the reliability, safety, or security of software products. Basically certification checks if the software product does not contain any faults. The absence of faults can be established with different means with increasing confidence. To express this increase in confidence in the absence of faults we have developed a software product maturity model (SPMM). The SPMM can be used in a number of ways:

- To guide the developers of software in good engineering practice;
- To measure the evolvement of a single product over time;
- To compare two products or versions of products;
- To certify a product for a certain level of maturity.

We start by introducing the concept of maturity and the CMMI model in Sections 2 and 3, respectively. In Section 4 we summarize one of the case studies the maturity model has originated from. In Section 5 we investigate typical elements of a software product. In Section 6 and 7 we describe the general framework and in Section 8 we

provide several examples of product maturity measures and their level. We conclude with a discussion of related and future work.

2. MATURITY

ISO/IEC 9126 [11] is the standard in the area of product quality (see also Section 9). It describes a quality model with several quality characteristics and accompanying metrics. One of these quality characteristics is maturity. According to ISO/IEC 9126 the definition of maturity is: "The capability of the software product to avoid failure as a result of faults in the software", where a fault is "an incorrect step, process or data definition".

From the definition we deduce that a software product without faults is the best guarantee for maturity. The absence of faults we call *correctness*.

In our maturity model we cater for an objective evaluation of the correctness of the product. Thereby we do not assume any detailed domain knowledge for the evaluator.

Hence, the model does not aim to determine if the system specification is correct with respect to the real world the system needs to operate in. Instead of this our maturity model specifies what needs to be done to determine if the system that is designed and/or built corresponds to a given specification and does not contain any internal faults (like imprecision or deadlocks).

We do not only consider the maturity of the final software product (the source code or working system), but also of intermediate deliverables like requirements and detailed design. We have identified different elements (artifacts, see Section 5 and Figure 2) within these deliverables, so that maturity can be investigated on a more detailed level.

We consider correctness of the software product to be consisting of two types of properties: *internal correctness* properties (within elements) and *external consistency* properties (between elements). So we make a difference between faults that are internal to an element and faults that indicate that two different elements do not match with each other. Internal correctness comprises a number of properties such as termination of an algorithm or unambiguity of requirement statements. Similarly, external consistency may demand that functional and non-functional requirements do not contradict or that the class structure in the code is the same as in the class diagram in the design document.

We do not (or to a lesser extent) consider other software product quality attributes like usability, maintainability and performance. However, the model does include checks for these quality properties of a software product if they have been specified in the requirements. The model will e.g. not consider code with 30% comments more mature than code with 10% comments, unless in both cases it was specified in e.g. the company standard or in a project document that code should contain at least 20% comments.

3. CMMI CONCEPTS

Among the most well-known models in software (process) maturity are the Capability Maturity Model Integration (CMMI) models. We discuss them here in more detail, because we used their concepts as a basis for the product maturity model.

Capability Maturity Model Integration (CMMI) models provide guidance for improving an organization's processes and the ability to manage the development, acquisition, and maintenance of products or services. The models place proven approaches into a structure that helps an organization to appraise its organizational maturity or process area capability, establish priorities for improvement, and implement these improvements. Bodies of knowledge for systems engineering and software engineering have been reflected in the CMMI-SE/SW. [3]

The CMMI-SE/SW model has two different representations:

- The staged representation where an organization can be in one of five maturity levels depending on the process areas it has implemented
- The continuous representation where maturity levels have been replaced by six capability levels as a measure assigned individually to each process area. A conversion into maturity levels is possible.

The main components of the CMMI-SE/SW model are process areas (PAs), goals, practices and work products. A process area is a cluster of related practices like Requirements Development, Verification or Product Integration. Each PA contains up to three Specific Goals (SG) that describe what must be implemented to satisfy the PA. Each of those specific goals comes with several Specific Practices, which are activities considered important in achieving the goal. The specific goals and practices are the operational trajectories of the standard and lead to typical work products that are also mentioned in the standard. There are also Generic Goals that hold for all PAs that describe the institutionalization that the organization must achieve for a certain capability level. Each of these Generic Goals also comes with several Generic Practices.

The capability levels from the continuous representation can be transformed into the five maturity levels of the staged representation.

For the product maturity model we will use similar notions. Because we do not consider the process, the concept of practices has been transformed into properties of the product. A summary of the concepts is also given in Figure 4 at the end of this paper:

- **Product Areas** which are the highest level of division of the product into parts. Each product area contains different sub-parts, called elements (see Section 5);
- **Generic Goals** for all product areas and **Generic Properties** that are important to achieve the goals (see Section 6);
- **Specific Goals** per product area and **Specific Properties** that are important to achieve the goals (see Section 8);
- A four-dimensional **Maturity Scale**, for which the scorings can be transformed into five **Maturity Levels** (see Section 7).

Before we continue with a more detailed explanation of each of these notions, we will first summarize a case study we have performed, to show our basic idea on internal correctness and external consistency checks and the kind of faults we were able to retrieve.

4. CASE STUDY

The Software Product Maturity Model was developed during and inspired by case studies that we have done. Case studies are projects for companies in the area of product certification, in which we can test and extend our verification techniques. The case studies range from manual reviews of requirements to formal verifications of source code.

In this section we will describe one of these case studies, where a company that was subcontracted to create the *functional design* of a government central system, asked us to *verify the quality* of the design before delivery.

We started the case study by picturing the different elements in the functional specification of this subcontractor. The architects appeared to be following the Rational Unified Process [14] for their software development, which indicates working with stakeholder needs, system features derived from them and use cases that implement the features. There are also non-functional (supplementary) requirements.

Figure 1 describes the elements in the functional specification and their relations. The stakeholder had delivered user requirements and a process model of their business processes. These user requirements could be divided into needs (high-level user wishes), features (system functionality) and non-functional requirements. The architects added a use case description, a functional architecture (logical module structure), an object model and a supplementary specification (all non-functional requirements like legal, security, performance etc.). The 'Reference' element in Figure 1 refers to e.g. standards or laws in the requirements domain; everything from outside the project that can be used to check the specification against.

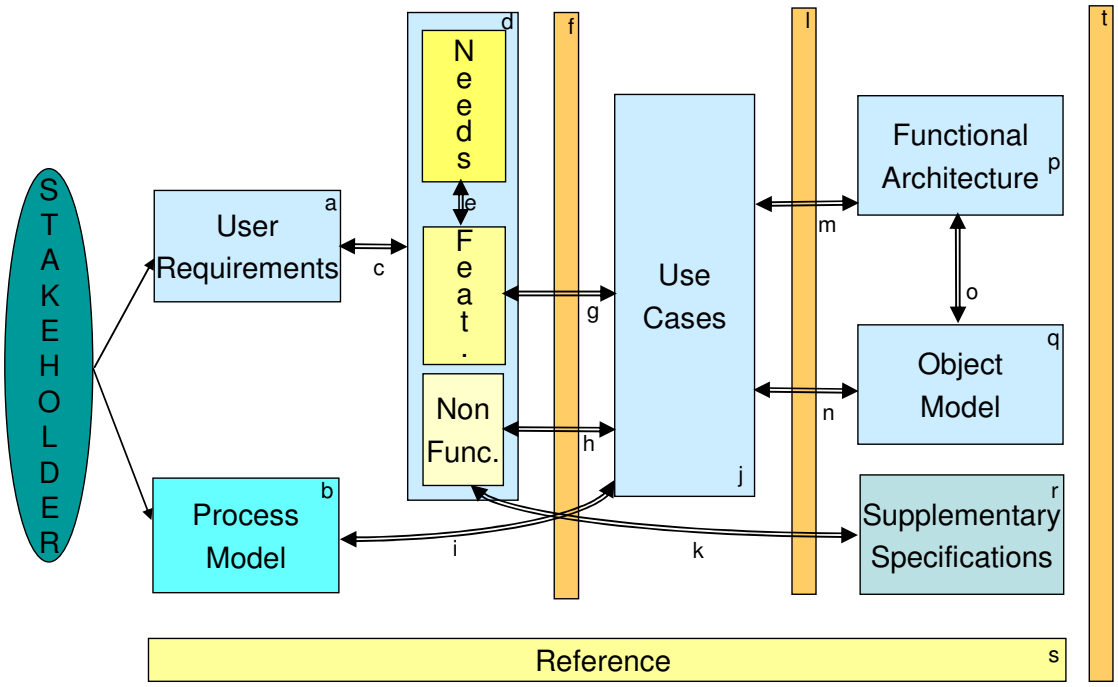

Figure 1: Case Study: Functional Design Review

The characters a…t in the picture refer to checks that can be made on the elements and their relations. Examples of checks are:

[a] Similar and contradicting statements should be identified.

[j] The use case diagram and the activity diagrams of the use case flow should match the textual description and follow the UML modeling standard. The main flow should guarantee the postcondition, assuming the precondition.

[e] The needs (high-level user wishes) should be linked to the features (system functionality). Each need should be covered by at least one feature and there should be no "orphan" features.

[m] It has to be decided which functional component supports which use case. All functional components should be needed and all use cases should be supported sufficiently (all functionality described in the use cases should be executable by one of the components).

'f', 'l' and 't' are major checkpoints where the elements to the left of it were checked for spelling errors and wrong referrals (e.g. to a chapter that does not exist). If the quality of the element would not have been good enough to continue the process (of matching with other documents) it would have been returned for improvement.

All of the individual elements and their relations were checked for completeness and consistency. The result was a number of violations in one of the following three categories:

1. Minor inconsistency: the inconsistency is identified and it is clear which of the options is the right one or how to correct it (e.g. a spelling error). These kinds of inconsistencies hinder the reader, but he/she is still able to understand the intent of the designer.
2. Dangerous inconsistency: it is not immediately clear which of the options is the right one or the inconsistency could go unnoticed by the reader and therefore lead to a wrong interpretation of the design (e.g. a diagram that does not match the text, while used as basis for the implementation).
3. Incompleteness: the design is incomplete in some way because not all relations have been implemented. The reader could create her own solution for the missing part but most often the designer or users of the system need to be asked which solution they prefer (e.g. which scenarios they foresee for a missing use case).

The diagnosis consisted of a summary of all found violations in each of the three categories. To achieve a "consistent and complete" functional design the second and third category should be empty or a feasible reason should be given to allow incompleteness or inconsistency for some items.

Examples of findings:

- Some post-conditions are not correct. (type 1)
- The actor description ("used in use case…") does not correspond to the use cases. The use cases are obviously leading for the right option. The functionality overview does not contain all of the use cases and actors described elsewhere. (type 1)
- The activity diagrams do not match the use case text (especially not for the alternative flows) (type 2)
- <A certain actor> is not used in a consistent manner (mix between human and non-human) (type 2)
- The supplementary requirements are not complete (type 3)
- Missing features are identified from the use cases (type 3)
- Some components and links between components to support the use cases are missing in the Functional Architecture (type 3)

All inconsistencies of the second and third type 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 chance for confusion and misinterpretation.

The quality of the design (and thus the effort needed for the developers to understand and interpret it) was further improved by also resolving the inconsistencies of the first type.

The method that was developed during this case study is the basis for our maturity model. The method consists of identifying elements and their relations and using this for checks on correctness. The method was further developed in subsequent case studies and extended for the other phases of the software development life cycle.

In the remainder of this paper we will present the framework of the method in more detail. The complete model facilitates a structured approach to system maturity evaluation.

5. 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 analysis and a user requirements part, mostly to emphasize the importance of the analysis of the system environment. We have split the test deliverables into their well-known sub-parts as well.

The framework consists of eight product areas:

- The *context analysis (CA)* 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 system 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.
- Finally the *implementation (IMP)* contains the system and its documentation, built according to the design.
- The *unit & integration test (UIT)* describes tests of the different software components. Components can either be tested stand-alone (unit) or together (integration). Tests are based on the detailed design specification.
- The *system test (ST)* describes tests of the whole system on functionalities. The tests are based on the functional specification.
- The *acceptance test (AT)* describes tests of the whole system on suitability for the users. The tests are based on the user requirements and performed in the actual production environment.

The risk assessment is not a separate product area, but can be part of any of the tests to indicate which part of the system needs the most thorough verification (high-risk). The risk assessment is updated after every development step.

Each area can be further divided into subparts, which we call elements in this paper. 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 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 Risk Assessment is updated after every deliverable.

Due to space restrictions not all sub-elements will be discussed in this paper. Some of them will be explained with the examples in Section 8.

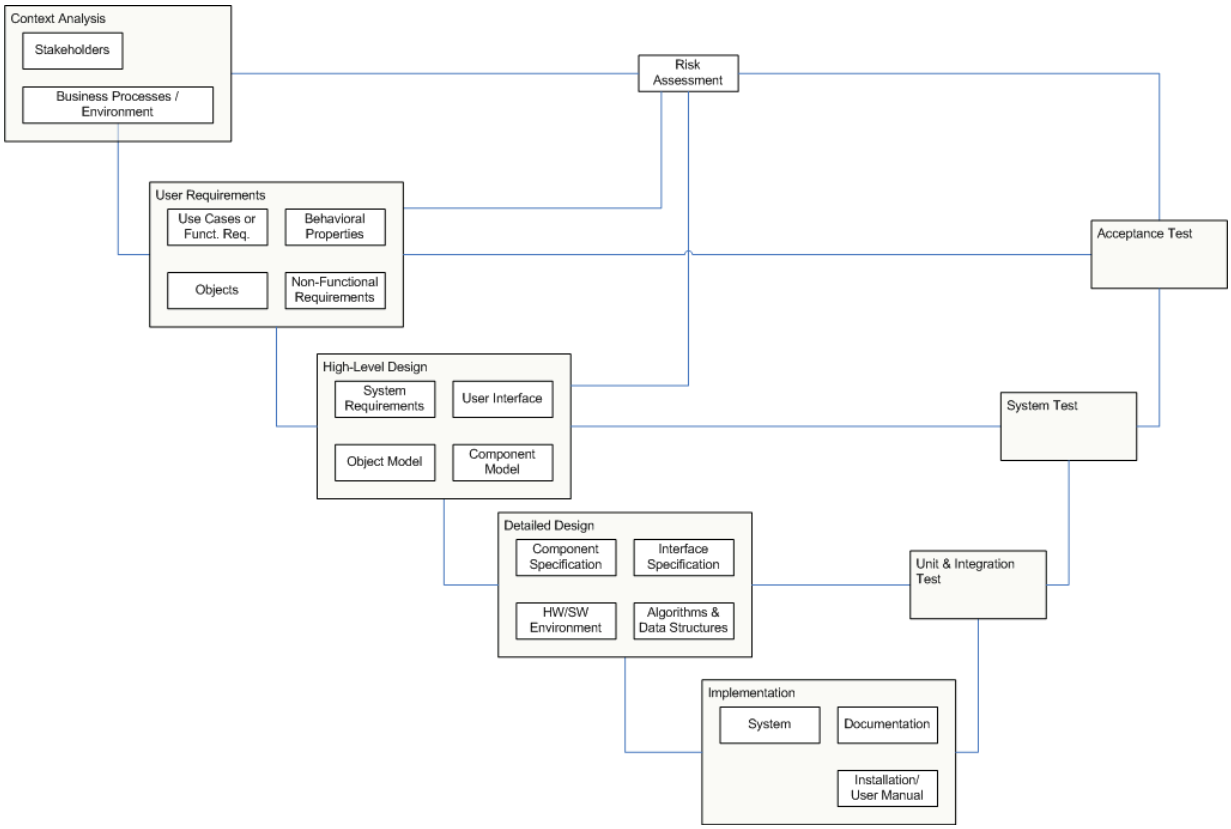

Figure 2: Software Product Areas with their Elements

6. GENERIC GOALS

Generic goals (GG) are goals that have to be achieved for each product area. They are translated into generic properties that hold for each product area.

There are four generic goals for all product areas:

[GG1] **Complete**. All required elements in the product area should be present and as much formalized as possible.

[GG2] **Uniform**. The elements in the product area should all be standardized.

[GG3] **Correct**. Each element should contain no internal consistency or correctness faults.

[GG4] **Consistent**. All elements should be consistent with each other and with the other product areas.

For each of these generic goals different *achievement levels* can be established, which we have summarized in Table 1.

The completeness of a product area (GG1) 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 SP1.2 and SP1.3 in Section 8.

The uniformity of a product area (GG2) 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. 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 correctness and consistency of the product area (GG3 and GG4) can both be established with different means that gradually increase confidence: manually (level 1), with tool support (level 2), or by formal verification (level 3). For each product area the Specific Properties (see Section 8) will indicate with respect to what correctness and consistency will be verified.

From the levels in Table 1 and the generic goals we derive the generic properties: one for each goal that simply indicates that the level should be as high as possible:

[GP1.1] *As many formalized elements as possible*. Score 0 if any required element is missing; score 1 if any semi-formal element is missing; score 2 if any formal element is missing; score 3 if all elements are present.

[GP2.1] *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.

[GP3.1] *Zero faults with the most thorough check possible on internal correctness*. 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 detects no faults and formal checks have been done.

[GP4.1] *Zero faults with the most thorough check possible on external consistency*. 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 detects no faults and formal checks have been done.

The specific goals indicate for each product area what the required elements, applicable standards and possible checks are. We will give an example of the specific goals for the requirements area in Section 8.

Table 1: Generic Goal Achievement Levels

GG1	**Complete**
0	Some required elements are missing
1	All required elements are present
2	Semi-formal elements have been added
3	Formal elements have been added
GG2	**Uniform**
0	No standardization
1	Within the product
2	Compliance to a company standard
3	Compliance to an industry standard
GG3	**Correct (within elements)**
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
GG4	**Consistent (between elements)**
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

7. MATURITY LEVELS

From the four levels that have been achieved for the Generic Properties a maturity level can be calculated.

The model indicates a maturity level per product area. The maturity 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 maturity of the final product without taking into account the development deliverables or testing deliverables.

The maturity levels are based on an intuitive notion of when products are more mature. A basic level is achieved when at least all required elements are present. The next level is when the required elements are present and uniform. We place external consistency above internal correctness because for the former all relations between elements need to be correct, which is more complicated than just considering the elements on their own. Finally, the highest level of maturity 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 maturity levels. For each maturity level we have indicated the level that is needed for each of the Generic Properties (see also Table 1):

1. **Initial**
   GP1.1≥1 and GP2.1=0 and GP3.1=0 and GP4.1=0
   Each of the required elements is present in the product.
2. **Standardized**
   GP1.1≥1 and GP2.1≥1 and GP3.1=0 and GP4.1=0
   All elements are uniform.
3. **Correct**
   GP1.1≥1 and GP2.1≥1 and GP3.1≥1 and GP4.1=0
   All elements are internally correct and consistent.
4. **Consistent**
   GP1.1≥1 and GP2.1≥1 and GP3.1≥1 and GP4.1≥1
   All elements are consistent with each other and with other product areas.
5. **Verified**
   GP1.1≥1 and GP2.1≥1 and GP3.1=3 and GP4.1=3
   All elements and relationships have been verified with mathematically-based methods wherever possible, or the most rigorous method otherwise.

Note that this 5-step scale could be further subdivided. For instance we could say that “Industry standardized” (GP2.1=3) is more mature than “Company standardized” (GP2.1=2), or we could distinguish different maturity levels of correctness, depending on the verification method used. To reduce complexity we have restricted the scale to 5 levels.

8. SPECIFIC GOALS: REQUIREMENTS

Specific goals (SG) are goals that hold for one product area only. Each product area has a different set of specific goals (although they convey some similarity as they are based on the generic goals). The specific goals are all implemented by one or more specific properties (SP) for the product area.

In this section we suggest a necessarily incomplete list for the User Requirements Product Area. The specific goals are a direct translation of the four generic goals to the user requirements area. The specific goals for other product areas will be similar, but of course the specific properties that accompany them (the required elements, standards and checks) will be different for each product area.

[SG1] *Complete*. Describe the requirements as detailed and as formal as possible.
[SG2] *Uniform*. Comply with standards in requirements engineering.
[SG3] *Correct*. Describe each element in the requirements in a correct way.
[SG4] *Consistent*. Maintain correct and consistent relations between the elements in the requirements description and with the context analysis.

We will first briefly describe the different elements we consider to be part of a complete user requirements specification:

- *Functional requirements* or *Use-cases*. 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. A use-case is a named "piece of functionality" as seen from the perspective of an actor. For each use-case several use-case scenario's are given (sequences of actions, events or tasks), the permitted or desired ones as well as the forbidden ones.
- *Objects*. Many types of entities play a role in the environment processes and some of them have to be represented in the system. Only these are collected. In fact we do not list here the individual entities, but only the types or classes to which they belong (so we do not collect items like "client Johnson", but only "client"). The object description can be quite informal in the form of a glossary, or more advanced in the form of a data dictionary or object model.
- *Behavioral Properties*. General behavioral properties that should hold, such as 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.

- *Non-functional requirements*. These are also called quality requirements. It is a set of different types of quality measures.

From this set of elements and the specific we derive a number of specific properties. These specific properties indicate the elements or checks that are required for a certain generic goal level. For instance, to achieve level 2 for GP3 (correctness), all checks in SP3.1 (manual) and SP3.2 (tool-supported) need to be performed and should not reveal any faults. This set of specific properties has been collected from literature, standards [1][4][6]-[10][15] and our own experience.

[SP1.1] Required Elements
- Functional requirements;
- Non-functional requirements;
- Glossary.

[SP1.2] Semi-formal Elements
- Data dictionary or object model;
- Use cases (with scenarios);
- Behavioral properties.

[SP1.3] Formal Elements
- Relational diagram of data/object model;
- Process model of use case scenarios;
- Behavioral properties specification, e.g. temporal logics.

[SP2.1] Compliance with Industry Standards
- ERD diagram for object/data model;
- UML diagrams for use cases.

[SP3.1] Internal Correctness
- No two requirements or use cases contradict each other.
- No requirement is ambiguous.
- Functional requirements specify what, not how (no technical solutions).
- Each requirement is testable.
- Each requirement is uniquely identified.
- Each requirement is atomic.
- The definitions in the glossary are non-cyclic.
- Use case diagrams correspond to use case text.

[SP3.2] Automated Correctness Checks
- Requirements are stored in a requirements management tool which uniquely identifies them.

[SP3.3] Formally Verified Correctness
- Verify use case scenario models for e.g. correct workflows (no deadlocks or dead tasks) and mutual consistency.
- Check data model diagram for normal form.

[SP4.1] External Consistency
- Each ambiguous or unclear term from the requirements is contained in the glossary.
- The use cases or functional requirements are a detailing of the environment description in the context analysis (no contradictions). Each step in a business process that involves the system has been included. Each task that the system should fulfill for its environment has been included. All actors of the context analysis have been included in the requirements.
- Each object is mentioned in the requirements and all objects mentioned in the requirements are contained in the object model.
- The requirements do not contradict the behavioral properties.
- The use case or functional requirements do not render the non-functional requirements impossible.

[SP4.2] Automated Consistency Checks
- Requirements and glossary/objects are stored in a requirement management tool which shows the relations between requirements, scenarios, actors, and objects.

[SP4.3] Formally Verified Consistency
- Verify use case scenario models for e.g. compliance with behavioral properties and non-functional requirements.
- Verify that the requirements description complies with the environment description from the context analysis.

Note once again that the list above is not complete, but gives an idea of how a maturity model as described in this framework would look.

Just as with CMMI, the complete model also contains comments and advice that e.g. indicate which tools to use or which formal methods are suitable to verify the properties.

These comments also explain what an atomic requirement is, what the characteristics of a testable requirement are, etc. The comments and advice are included to make the maturity model more usable and self-contained.

As an example we apply the maturity model to the case study from section 4 (after all faults found have been corrected). For GG1, we only achieve level 1, because no behavioral properties have been specified. For GG2 we achieve level 3 because all elements were uniform and ERD and UML diagrams were included. For GG3 we achieve level 3 because the use case activity diagrams were formally verified as well as the ERD diagram. For GG4 we only achieve level 1 because no requirements tool was used and the absence of behavioral properties prevented formal verification.

From the achievement levels we can conclude that the requirements part of the functional design has maturity level 4. However, it is very close to maturity level 5,

because for this behavioral properties need to be specified and verified against the use case model.

9. RELATED WORK

We have mentioned that our maturity model was inspired by CMMI. CMMI is however for process maturity, not for product maturity. We did not find any other models that describe product maturity in the sense of analyzing the correctness of a product. However, there are some other related findings in the area of product quality and maturity.

The software product maturity model by John Nastro [16] 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 maturity. As Nastro states the importance or weight of the elements and even the elements themselves may vary from project to project.

The ISO/IEC standard 9126: "Software engineering — Product Quality" [11] 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 Figure 3) for internal and external quality, which are further subdivided into subcharacteristics. These 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 [12].

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 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 (e.g. measure the number of defects found during a production period). We therefore believe that our model is more suitable to determine product maturity (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 maturity model with other characteristics from ISO/IEC 9126.

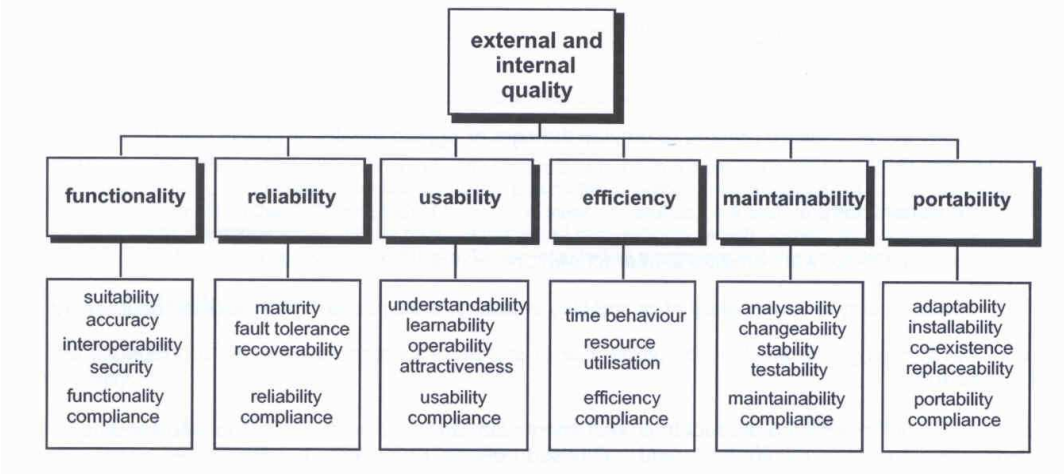

Figure 3: ISO 9126 Internal and External Quality Characteristics

The maturity levels in our model are similar to the review levels in an article by Connie Clayton [2]. 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 related standards (DOD-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.

Jakobsen et al. [14] 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 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 e.g. the FDA [5] does not prove correctness. 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 is fault free. Therefore, the manufacturer cannot use certification to avoid assuming its legal or moral obligations. We proposed a certification model that does focus on correctness.

In summary we can say that we did not encounter any models that address product maturity in the same sense that we do: related to correctness and consistency. 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 test cases), its focus on correctness 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.

10. CONCLUSION AND FUTURE WORK

In this paper we have described a framework for a Software Product Maturity Model. We have given some examples for the Requirements Product Area, but this list is not complete and in the final model will be accompanied by many more comments, advices and explanations. This is what we are currently elaborating on, also for the other Product Areas. Most of our input comes from case studies in industry, with real-size products and documentation.

The maturity 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 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, so that our Software Product Maturity Model (SPMM) becomes recognized as a product maturity standard.

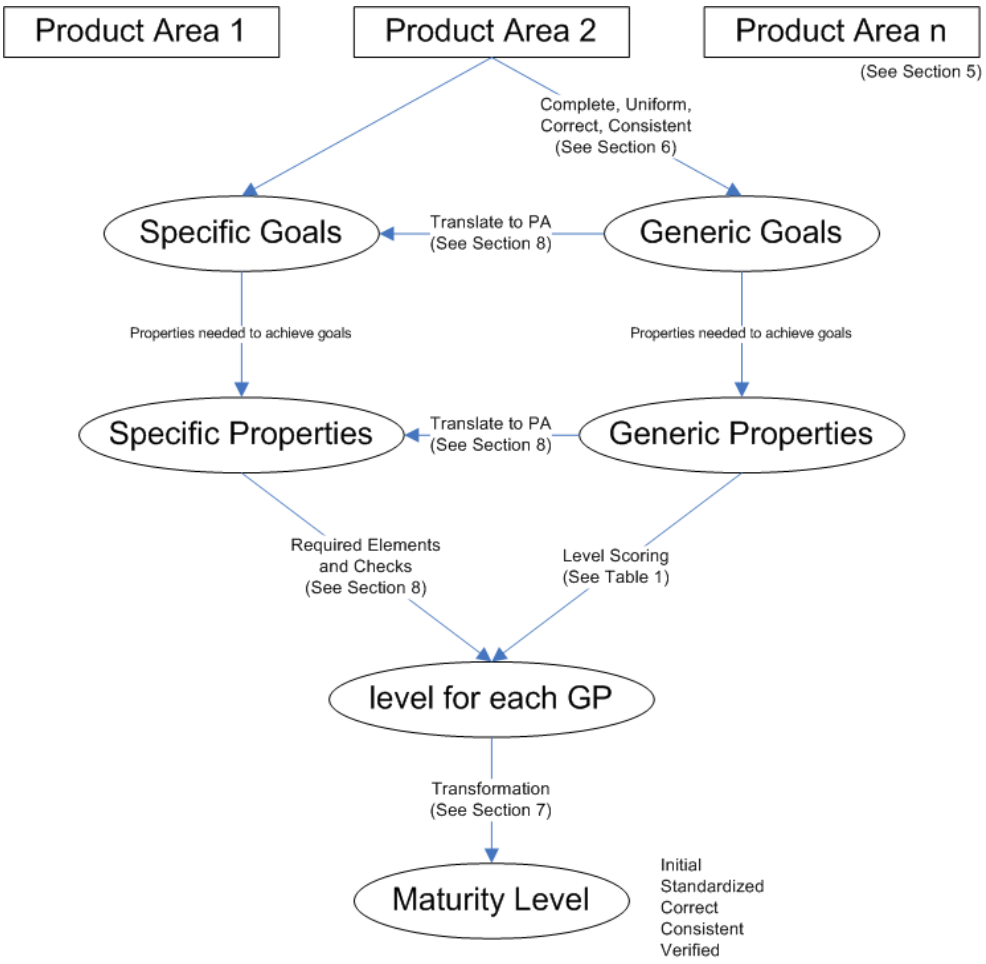

Figure 4: Summary of the Maturity Model Framework

11. REFERENCES

[1] R. Bamford and W.J. Deibler. ISO 9001:2000 for Software and Systems Providers. An Engineering Approach. CRC Press, 2004.

[2] C. Clayton. Defining Review Levels for Software Documentation. CrossTalk, Vol. 9, Nr. 1, 1996.

[3] CMMI Product Team. 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-2002-001, December 2001.

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

[5] FDA. General Principles of Software Validation; Final Guidance for Industry and FDA Staff. January 11, 2002.

[6] E. Hull, K. Jackson and J. Dick. Requirements Engineering. Springer-Verlag, 2002.

[7] IEEE Computer Society. IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications. 1998.

[8] IEEE Computer Society. IEEE Std 1233-1998. IEEE Guide for Developing System Requirements Specifications. 1998.

[9] IEEE and EIA. IEEE/EIA 12207.0-1996. Industry Implementation of International Standard ISO/IEC 12207: 1995 (ISO/IEC 12207) Standard for Information Technology - Software life cycle processes. March 1998.

[10] IEEE and EIA. IEEE/EIA 12207.1-1997. Industry Implementation of International Standard ISO/IEC 12207: 1995 (ISO/IEC 12207) Standard for Information Technology - Software life cycle processes - Life cycle data. April 1998.

[11] ISO/IEC. Software Engineering - Product Quality. International Standard ISO/IEC 9126, 2001.

[12] ISO/IEC. Information Technology - Software Product Evaluation. International Standard ISO/IEC 14598, 1999.

[13] ISO/IEC. Information technology - Process assessment. ISO/IEC 15504, 2004.

[14] A. Jakobsen, M. O'Duffy and T. Punter. Towards a Maturity Model for Software Product Evaluations, in: Proceedings of 10th European Conference on Software Cost Estimation (ESCOM'99), 1999.P. Kruchten. The rational unified process: an introduction. 3rd ed., Addison-Wesley, 2004.

[15] C. Mazza, J. Fairclough, B. Melton, D. De Pablo, A. Scheffer, R. Stevens, M. Jones, G. Alvisi. Software Engineering Guides. Prentice Hall, 1996.

[16] J. Nastro. A Software Product Maturity Model. CrossTalk, Vol. 10, Nr. 8, 1997.

Examining Security Certification and Access Control Conflicts Using Deontic Logic

M. Smith, M. Kelkar, R. Gamble

University of Tulsa
Department of Mathematical and Computer Sciences
gamble@utulsa.edu

Abstract. Component-based software has become a mainstream practice as organizations attempt to streamline application development tasks. These applications invariably contain third-party Commercial-off-the-Shelf (COTS) systems with black box functionality. When integrated applications require security certification, COTS components—even if individually certified—may introduce vulnerabilities into the system if their security mechanisms are poorly combined. One cause of improper integration can be found in the access control mappings across COTS component domains. Missing, conflicting, and ambiguous mappings can lead to non-compliance with security certification criteria. In this paper, we discuss certification criteria applicable to COTS integration and their interpretations to access control across domains. Highlighting common conflicts using deontic logic, we indicate how resolution strategies to those conflicts can comply with certification criteria.

Keywords: COTS-based Systems, Security Certification, Access Control

1 Introduction

Commercial-off-the-shelf (COTS) components are independent pieces of software that can be combined into larger systems, the main goal being that they are reusable. Inserting COTS software into larger integrated applications is a common, inexpensive development approach. Conflicting security mechanisms can diminish the potential for the complete security certification of the application by a certification and accreditation authority.

Security certification assesses a software system to determine the extent to which the system design and implementation complies with pre-specified security requirements [1, 2]. This process increases the level of confidence in the system's security by ensuring correct implementation, intended operation, and expected output. Certification discloses security vulnerabilities with their associated risk.

Currently, certification is a manual process with no widely-accepted automated solution. Security certification for integrated systems composed of COTS components is necessary to ensure that the procured components which match the desired functionality is

already certified in isolation as an atomic entity [3]. Apart from this, the certification should include security property compatibility checking between interacting components and the global properties [4]. Failure to attain this may result in incomplete certification and lead to unidentified vulnerabilities.

Predominant software certification sources are government and military documents, such as the Common Criteria [5], DITSCAP [2]and the NIST Information System certification document [1], which outline the need to assess critical security features for correct and complete policy implementation of an integrated system. The documents are primarily directives for the integrator and certifier to review the interfaces between components and determine their security compliance. Thus, the criteria are used as a basis for compliance checking, ascertaining contributing properties, and identifying conflicts that cause non-compliance.

Given that each COTS software component has its own domain with a consistent and correct security policy within that domain, we consider the problems arising from an interdomain policy that is expressed through role assignment inheritance or global policy statements for all roles, i.e., a single root representing that is inherited by all roles in all components. Applicable certification criteria for integrated COTS-based systems can be interpreted to access control policies as follows.

A. The interdomain access control policy should be assessed to ensure correct and complete implementation (Verification phase, task 2.2 - Level 2) [2].
B. Each policy requirement must be defined unambiguously by a complete set of attributes (e.g., initiator of an action, source of the action, the action, the object of the action, constraints) [6].
C. If policies conflict among component domains, a deny-overrides strategy should be used to resolve the conflict [7]. Simply stated, if no other conflict resolution strategy can be formulated, access is denied.
D. Policies should be traced to the originating component to enforce non-repudiation and accountability for any conflicts that may arise [2, 5, 6].

Access control is a major cross-component security related issue for integrated COTS software systems. Conflict detection and conflict resolution are often dealt with separately using different processes, languages, mechanisms, etc. [8-10]. Most approaches define security policies using logic-based languages [9, 11-13], variations on RBAC systems [8, 14, 15], or other specifications such as XACML [16]. Conflicts among access control policies have been intensively studied [10]. One variation of RBAC, the XGTRBAC, specifically focuses on interdomain conflicts [17]. The administrative model shows conflicts primarily related to role mappings across domains as well as temporal constraints on the assignment of roles. Though the conflict resolution technique is automated, conflict detection is a manual process and is done by visually scanning directed sub-graphs.

Certification of integrated systems provides a different perspective on access-control conflict detection and resolution than that used in prior research:

- Integrated systems have an overriding policy with which COTS components must comply.
- Pair-wise comparisons are not sufficient.
- Any conflict resolution must be globally consistent.
- Traceability of role inheritance and localization of conflict resolution reduces the re-certification effort due to evolution.
- Missing interdomain mapping formations can cause anomalies in propagated policies that can lead to vulnerabilities.

In this paper, we use standard deontic logic for property expressions that indicate valid access permissions. We investigate interdomain access control policy statements for security vulnerabilities related to the certification priorities of consistency and correctness. We use deontic logic to highlight common conflicts, their discovery, and resolution to comply with certification criteria.

2 Role-Based Access Control

Role-Based Access Control (RBAC) is commonly used to define access parameters within COTS components and combinations thereof. This is done by creating rule-sets of permissions, assigning the rules to roles, and then assigning roles to users [18]. The robust, low-maintenance, and efficient nature of RBAC systems allow for simple modeling of many constraints including hierarchical and separation of duty (SoD) [19]. A role hierarchy is a partial order relationship established among roles. SoD constraints define mutual exclusion relations between two entities. Users assigned to a role have access rights defined by the permissions assigned to the same role.

In RBAC, access is defined in terms of user-role mappings that specify a set of security constraints restriction on role access for a set of users. When these mappings are between distinct software components or domains, they are called interdomain mappings. Approaches that examine interdomain conflicts at the design phase use XML or UML to detect interdomain conflicts [10]. For example, recurrence of a conflict template in a UML system representation denotes conflict. Thus, the UML architecture diagram is manually scanned. Similar research is found in the XGTRBAC Admin model, which shows examples of cross-domain problems [17]. A conflict is detected by visually scanning directed sub-graphs. The conflict resolution technique is automated using formal language implementation [17].

Security policies of an integrated COTS system can be defined at three levels:

1. Local policy of a component that specifies the constraints between its roles.
2. For each role which is included in the local policy of a component.
3. Globally for the system, defining interdomain mappings

Conflicts arise when any of the above policies have a contradiction.

RBAC systems have noted limitations that should be addressed in integrated system certification analysis [15, 17]. Inheritance may be ambiguous where it does not correspond to an organization's hierarchy. It is not necessary for all roles to allow user assignments as some roles in the hierarchy are defined solely for convenience and are not intended to have users assigned to them. We call these non-user roles. Context is not included in role assignment constraints, which can be eased if traceable origins of inherited access are maintained across domains of the integrated system.

3 Deontic Logic

Deontic logic, as proposed by vonWright [20], extends modal logic with concepts that define operations on the obligatory, the permitted, and the forbidden. Several variations of deontic logic exist, but we use Standard Deontic Logic (SDL) that historically has been used for reasoning about morality, law, and social norms [21]. Many of the basic illustrations of the logic are taken from vonWright [20].

An *action* is the antecedent for the deontic operators such that "it is permitted that *action*." If an action is performed (or not performed), nothing can be stated as to its obligatory, permitted, or forbidden character. In other words, actual performance is unrelated to any given permission.

Permission is defined as being allowed to complete some action. Logically, it is represented as PA, which is read as, "Permission to achieve A," or "It is permitted that A." Disjunctive and conjunctive rules of logic apply to permission, such that (1) and (2) necessarily hold. However, it is not necessarily true that the contrapositive of (2) holds.

$$PA \vee PB \equiv P(A \vee B) \tag{1}$$

$$P(A \wedge B) \rightarrow PA \wedge PB \tag{2}$$

Prohibition is the negation of Permission, designated logically as ¬PA ("not permission to achieve A," or "forbidden to achieve A"). For example, we are not allowed to steal hence we must not steal.

Obligation requires a Subject to do an Action and is stated as "it is obligatory that A," which is equivalent to ¬(P¬A). Obligation implies permission, as in equation (3). For example, in the non-smoking compartment, you are obligated to not smoke, therefore non-smoking is permitted, and smoking is forbidden.

$$OA \rightarrow PA \tag{3}$$

Indifference implies permission and is denoted in equation (4). This occurs when an action and its negation are both permitted. Permission does not imply indifference while it is equivalent to ¬O¬A, or the statement, "You are not obligated to not achieve A." For example, in a smoking compartment, we may smoke, but we may also not smoke. If

Action A is indifferent, then ¬A is also permitted, but if A is obligatory (and thus permitted), then ¬A is forbidden, as seen in equation (5).

$$PA \wedge P\neg A \rightarrow PA \tag{4}$$

$$OA \wedge PA \rightarrow \neg(P\neg A) \wedge PA \tag{5}$$

Dispensation is a non-obligation; a waiver from a required action [22]. This is denoted as equation (6). Notice that this is not equivalent to, nor does it imply, ¬PA. For this reason, prohibition does not imply non-obligation.

$$\neg OA \equiv P\neg A \tag{6}$$

VonWright defines six laws of commitment (equations (7) through (11)) and one rule that says you can never be obligated to do the forbidden (equation (12)). We extend these to include the converse of (9) to state that if you are not obligated to do A and the obligation of A implies the obligation of B, you are therefore not obligated to do B (equation (14)). These prove important when dealing with the inheritance of different kinds of constraints, as can be seen in the Chinese Wall example. Finally, the *Principle of Permission* is stated as, "you cannot have both permission and not permission at the same time" (equation (15)).

$$PA \wedge O(A \rightarrow B) \rightarrow PB \tag{7}$$

$$\neg PA \wedge O(A \rightarrow B) \rightarrow \neg PB \tag{8}$$

$$OA \wedge O(A \rightarrow B) \rightarrow OB. \tag{9}$$

$$O(A \rightarrow (B \vee C)) \wedge \neg PB \wedge \neg PC \rightarrow \neg PA \tag{10}$$

$$OA \wedge O((A \wedge B) \rightarrow C) \rightarrow O(B \rightarrow C) \tag{11}$$

$$O(\neg A \rightarrow A) \rightarrow OA \tag{12}$$

$$\neg O(A \vee B) \wedge \neg PA \wedge \neg PB \tag{13}$$

$$\neg OA \wedge O(A \rightarrow B) \rightarrow \neg OB \tag{14}$$

$$PA \vee \neg PA. \tag{15}$$

3.2 Access Control Usage

Cuppens, et al. [23] represented access control policies using deontic logic. He denotes a difference between having the right to know something and actually knowing it, which brings in the concept of having a memory. This affects all separation of duty constraints and requires including a temporal concept to the model. Organizational-based Access Control is another approach that specifies a logical model for policy management. Here, a permission is denoted as *Permission(org, r, v, a, c)* that is read as *"in organization org, within context c, role r is permitted to perform activity a on view v"* [15]. This approach is tailored to alleviate the concerns regarding organizational policy hierarchies and the context limitations of the RBAC model.

In another approach, Free Variable Tableaux is also used to analyze policy conflicts [9]. Deontic logic is used in the form of Obli+, Obli-, Auth+, and Auth-. They also introduce Chinese Wall and SoD constraints to express mutual exclusion of different policies. Chinese Wall defines two mutually exclusive target roles or objects and is commonly used in financial sectors where conflicts of interest must be avoided. For example, a Chinese Wall is necessary in an investment bank that provides corporate financial services to companies and concurrently provides financial research information to the general public. If the bank manages Company A's finances, and no Chinese Wall is in place, the bank could manipulate the financial reports to encourage people to buy extra shares in it.

4 Interdomain Access Control Certification Issues

For our use, an RBAC access control policy from a deontic logic perspective is divided into a set of permissions and a set of constraints for each role. The policies defined in a permission set should be complete and unambiguous [2]. Constraints are applied to the permission set that should not contradict each other. In this paper, we address constraints in the form of role-based SoDs, user-based SoDs, and Chinese Wall.

4.1 Applying Certification Criteria to Deontic Logic Specifications

Interdomain role inheritance can propagate incomplete policy information. Thus, the Principle of Permission must be altered to remove the possibility of ambiguity—if you only have permission to do A, how do you know if you have permission to not do A? Or in the absence of permission to not do A, are you then obligated to do A?

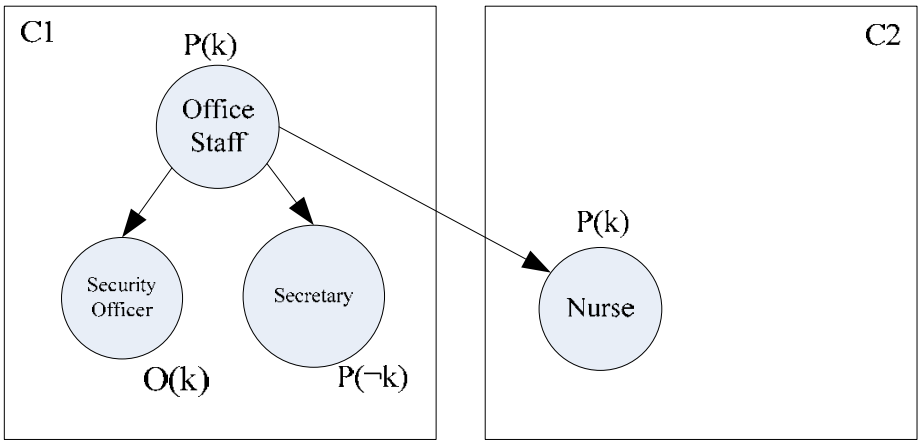

Fig. 1. Incomplete Interdomain Mapping Example

In Fig. 1, component C1 has three roles: office staff, security officer, and secretary (we do not use the user, object, action triplet in this example for simplicity). Office staff is a non-user role meaning no user is directly assigned to that role and gives permission to have keys P(k). This is inherited by user-role security officer who is obligated to have keys (O(k)) and user-role secretary who has permission not to have keys (P¬k). C2 is added to the system with user-role nurse. Nurse inherits all the policies from office staff, which is P(k) and is a ambiguous. It is evident that there is an interdomain inheritance mapping missing that would allow nurse to have complete and unambiguous permissions.

Looking at the conflict definitions and model, we adopt the triplet representation from [24]: (u,o,a), where the user, u, completes an action, a, on the object, o. Using the triplet, we clarify the above ambiguity with equation (16). O(u,o,a) is an unambiguous statement, so it follows that we must specify that $O(u,o,a) \rightarrow P(u,o,a)$ (17). The derived policy of P(u,o,a) and O(u,o,a), as seen in equation (18), explains the absence of a stand-alone O(u,o,a) in equation (16).

$$(P(u,o,a) \wedge O(u,o,a)) \vee (P(u,o,a) \wedge P\neg(u,o,a)) \vee (\neg P(u,o,a)) \quad (16)$$

$$O(u,o,a) \rightarrow P(u,o,a) \quad (17)$$

$$O(u,o,a) \rightarrow P(u,o,a) \wedge O(u,o,a) \quad (18)$$

Policies are attached to roles and can be divided into a set of permissions and a set of constraints (see equation (19)). Permissions are passed downward through the hierarchy.

$$Policy_{r1}: \{Permissions\} \cup \{Constraints\} \quad (19)$$

Now, in the case of related actions, we follow the law of commitment in equations (7) and (9), depending on the permission applied to the action. Then we add the implied permission to the policy's set of permissions and add the implication, $O(a_1 \rightarrow a_2)$, to its set

of constraints. For example, if we are required to write and writing implies reading, we must also read, meaning both actions (read and write) must be permitted or obligated.

4.1.1 Complete, Unambiguous, and Correct Policies

Certification criteria assert that the security requirement specifications should be complete (Section 1 - Cert Criteria A, B), and thus, unambiguous. A policy statement is unambiguous if and only if there is only one interpretation of the statement. Given (17) states that obligation assumes permission, equation (20) shows indifference that gives the user the choice to achieve A. Equation (21) denotes prohibition.

$$P(u,o,a) \wedge P\neg(u,o,a) \tag{20}$$

$$\neg P(u,o,a) \tag{21}$$

For certification, the security requirement specification must be correct (Section 1 - Cert Criteria A), and thus, not in conflict. For integrated system certification, the interdomain policy, as a combination of local policies, should be free of conflicting policy statements. Direct conflict of permissions is shown in (22).

$$P(u,o,a) \wedge \neg P(u,o,a) \tag{22}$$

4.1.2 Need of Deny-Overrides

In case of conflict between the combined policies of interacting components that must cooperate, the deny-overrides strategy should be used (Section 1 - Cert Criteria C). In a combined policy set, if any statement denies access, then deny-overrides allows that statement to prevail over the other contradicting statements. Thus, the access is denied in the resulting policy. Since deny-overrides is commonly specified as a revocation of permission, we define a function, called the Permission Revocation (PR) to apply the deny-overrides policy to conflicting policy statements. This function moves to false all permissions of the type that is being overridden by the global policy. Thus, the mappings of the function to the derived permissions are shown in (23) and (24). Note that obligation implies permission, so these mappings also apply for permission statements.

$$PR(\ O(u,o,a) \wedge \neg O(u,o,a)\ ) \rightarrow \neg O(u,o,a) \tag{23}$$

$$PR(\ O(u,o,a) \wedge \neg P(u,o,a)\ ) \rightarrow \neg P(u,o,a) \tag{24}$$

4.1.3 Using Origin Tags

Redundant policy statements that are propagated from different roles through role inheritance are allowable but must be managed properly. We maintain an origin tag denoted by a superscript on the permissions in order to trace a permission back to its original role. This aids conflict resolution by providing a path back to an earlier culprit. In other words, maintaining the role-association with the users that propagate the permission through role inheritance helps detect the conflicting, redundant permissions that might have come from roles of distinct origins. The ability to trace the information flow helps in maintaining accountability and takes care of non-repudiation, as addressed by certification (Section 1 - Cert Criteria D).

4.2 Direct Conflict Detection and Resolution

Direct conflicts occur when one role's policy includes contradicting statements making the resulting policy incorrect (Section 4.1.1), as seen in Fig. 2 where r_3 inherits both obligation and prohibition for the same object and action. This creates a logically contradicting statement of the form $A \wedge \neg A$.

One resolution strategy is to define a global policy-combining rule, similar to XACML where each <Policy> and <Policy Set> can respectively define a rule and policy combining algorithm [16] as in Fig. 3 below. However, this must also resolve the logical policy for r_3 in order for it to be a conjunctive tautology. We use the Permission Revocation function in Section 4.1.2 to apply the global policy to conflicting policy statements. This function makes false all permissions of the type that is being overridden by the global policy.

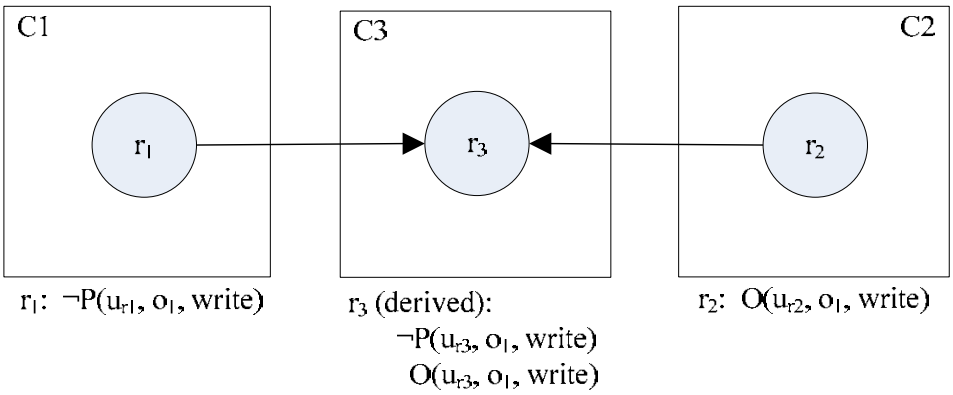

Fig. 2. Direct Conflict Example

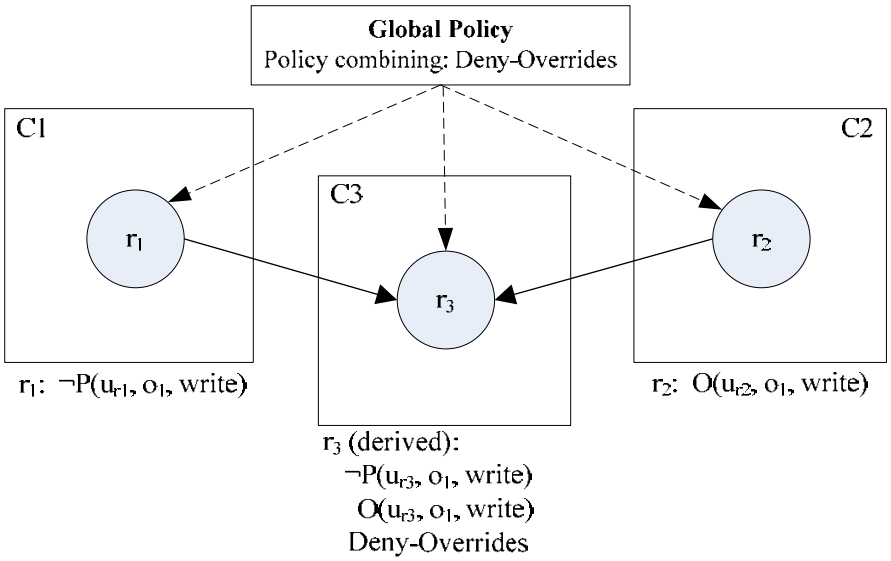

Fig. 3. Global Policy Resolution Strategy

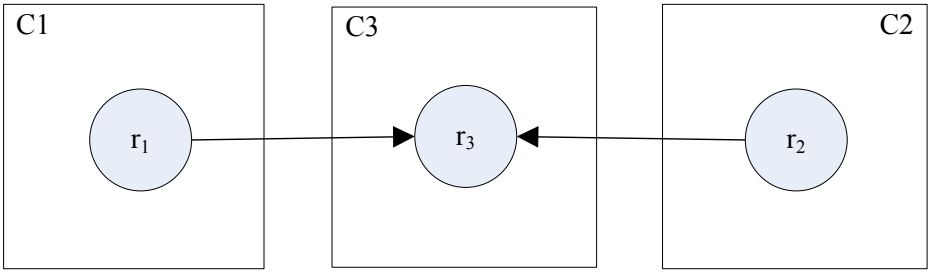

Fig. 4. Chinese Wall Example

4.4 Chinese Wall Conflict Detection and Resolution

In Fig. 4, access control policies for roles r_1, r_2, r_3 form the local policies of components C1, C2 and C3 respectively. This example illustrates how inherited permissions and constraints can induce additional constraints when the interdomain mapping causes conflict. The Chinese Wall (cw) concept is one form of mutual exclusion.

Role r_1 policy:

$$O(u_{r1},o_2,\text{read}) \tag{25}$$

$$O(u_{r1},o_3,\text{read}) \tag{26}$$

$$cw(u_{r1},\{o_3,o_2\},\text{read}) \rightarrow \neg(\, O(u_{r1},o_2,\text{read}) \wedge O(u_{r1},o_3,\text{read})) \tag{27}$$

As stated in (19), an access control policy specified for a role is divided into a set of mutually exclusive permissions and non-conflicting constraints. Given the Chinese Wall constraint, r_1 has two sets of permissions.

Role r_1 policy:

$$\text{Permissions} = \{\{ O(u_{r1},o_2,\text{read})\},\{ O(u_{r1},o_3,\text{read})\} \} \quad (28)$$

$$\text{Constraints} = \{\neg(O(u_{r1},o_2,\text{read}) \wedge O(u_{r1},o_3,\text{read})) \} \quad (29)$$

No constraints are given in Role r_2's policy. Its permissions are stated in (30).

Role r_2 policy:

$$\text{Permissions} = \{ O(u_{r2},o_3,\text{read})\} \quad (30)$$

Role r_3 does not have any local policies but inherits the permissions and constraints from r_1 and r_2, denoted by their origin tags. Given role inheritance properties, we form the cross product of the permission sets across distinct roles shown separately in (31) and (32). The union of the resulting ordered pairs results in the new permissions for r_3. The constraints also propagate as shown in (33). Thus, the resulting policy of r_3 is as follows.

Role r_3 policy:

$$\{ O^{r1}(u_{r3},o_2,\text{read}), O^{r2}(u_{r3},o_3,\text{read}) \} \quad (31)$$

$$\{ O^{r1}(u_{r3},o_3,\text{read}), O^{r2}(u_{r3},o_3,\text{read}) \} \quad (32)$$

$$cw^{r1}: \ \neg(O^{r1}(u_{r3},o_2,\text{read}) \wedge O^{r1}(u_{r3},o_3,\text{read})) \quad (33)$$

If viewed as purely inherited (no origin tags) it is clear that (31) conflicts with (33). This induces a new Chinese Wall for r_3 that explicitly focuses compliance given the permission origins.

$$cw^{r3}: \ \neg(O^{r1}(u_{r3},o_2,\text{read}) \wedge O^{r2}(u_{r3},o_3,\text{read})) \quad (34)$$

Equation (34) is added to the constraint set of role r_3 as a conflict resolution strategy. This causes (31) to be partitioned into:

$$\{ O^{r1}(u_{r3},o_2,\text{read}) \} \quad (35)$$

$$\{ O^{r2}(u_{r3},o_3,\text{read}) \} \quad (36)$$

However, it is clear that equation (32) is not in conflict with (36). Thus, the resulting permissions and constraints for r_3 are:

$$O^{r1}(u_{r3},o_2,\text{read}) \quad (37)$$

$$O^{r2}(u_{r3}, o_3, \text{read}) \quad (38)$$

$$O^{r1}(u_{r3},o_3,read) \wedge O^{r2}(u_{r3},o_3,read) \quad (39)$$

$$cw^{r1}: \neg(O^{r1}(u_{r3},o_2,read) \wedge O^{r1}(u_{r3},o_3,read)) \quad (40)$$

$$cw^{r3}: \neg(O^{r1}(u_{r3},o_2,read) \wedge O^{r2}(u_{r3},o_3,read)) \quad (41)$$

Thus, r_3's inherited policy is:

$$\text{Permissions} = \{\{ O^{r1}(u_{r3},o_2,read)\}, \{O^{r1}(u_{r3},o_3,read), O^{r2}(u_{r3}, o_3, read)\}\} \quad (42)$$

$$\text{Constraints} = \{ cw^{r1}(u_{r3},\{o_3,o_2\},read), cw^{r3}(u_{r3},\{o_3,o_2\},read) \} \quad (43)$$

4.5 User-Based SoD Conflict Detection and Resolution

In Fig. 5, there is bidirectional inheritance between roles r_1 and r_2, where both roles inherit permissions and constraints from each other, thus becoming equal. Access control policies for roles r_1 and r_2 which form the local policies of component C1 and C2, respectively, are specified below. Let m, n, p, q be integers to denote different users. For brevity we define the user based SoD between users as a negated conjunction of permissions.

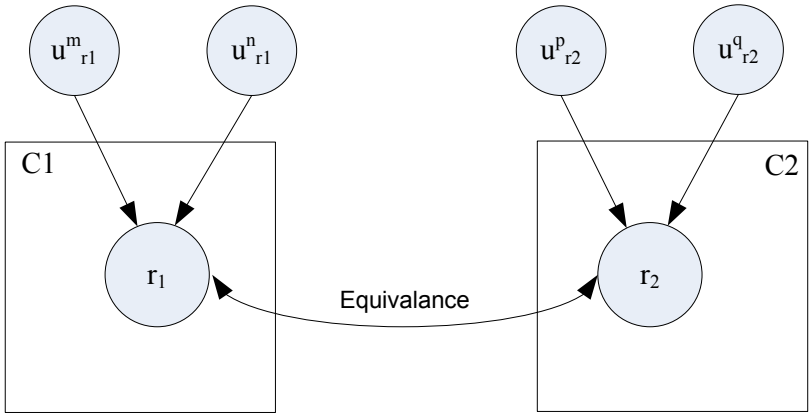

Fig. 5. Original user based SoD example

The user-based SoDs in roles r_1 and r_2 are found in (44) and (45), respectively.

$$usod^{r1}(\{u^m_{r1}, u^n_{r1}\}, o, a) : \forall o, a, m, n, u^m_{r1}, u^n_{r1}, m \neq n : \neg(O(u^m_{r1},o,a) \wedge O(u^n_{r1},o,a)) \quad (44)$$

$$\text{usod}^{r2}(\{u^{p}_{r2}, u^{q}_{r2}\}, o, a) : \forall o, a, p, q, u^{p}_{r2}, u^{q}_{r2}, p \neq q : \neg(O(u^{p}_{r2}, o, a) \wedge O(u^{q}_{r2}, o, a)) \quad (45)$$

These force the following permission sets on roles r_1 and r_2 (equations (46) and (47)):

$$\forall o, a, m, n, u^{m}_{r1}, u^{n}_{r1}, m \neq n \text{ , Permissions} = \{\{O(u^{m}_{r1}, o, a)\}, \{O(u^{n}_{r1}, o, a)\}\} \quad (46)$$

$$\forall o, a, p, q, u^{p}_{r2}, u^{q}_{r2}, p \neq q \text{ , Permissions} = \{\{O(u^{p}_{r2}, o, a)\}, \{O(u^{q}_{r2}, o, a)\}\} \quad (47)$$

The cross product of permission sets the following six policy statements:

$$\forall o, a, m, p, u^{m}_{r1}, u^{p}_{r2}, m \neq p \text{ , } \{\{O(u^{m}_{r1}, o, a)\}, \{O(u^{p}_{r2}, o, a)\}\} \quad (48)$$

$$\forall o, a, m, q, u^{m}_{r1}, u^{q}_{r2}, m \neq q \text{ , } \{\{O(u^{m}_{r1}, o, a)\}, \{O(u^{q}_{r2}, o, a)\}\} \quad (49)$$

$$\forall o, a, n, p, u^{n}_{r1}, u^{p}_{r2}, n \neq p \text{ , } \{\{O(u^{n}_{r1}, o, a)\}, \{O(u^{p}_{r2}, o, a)\}\} \quad (50)$$

$$\forall o, a, n, q, u^{n}_{r1}, u^{q}_{r2}, n \neq q \text{ , } \{\{O(u^{n}_{r1}, o, a)\}, \{O(u^{q}_{r2}, o, a)\}\} \quad (51)$$

$$\text{usod}^{r1} : \forall o, a, m, n, u^{m}_{r1}, u^{n}_{r1}, m \neq n \text{ , } \neg(O(u^{m}_{r1}, o, a) \wedge O(u^{n}_{r1}, o, a)) \quad (52)$$

$$\text{usod}^{r2} : \forall o, a, p, q, u^{p}_{r2}, u^{q}_{r2}, p \neq q \text{ , } \neg(O(u^{p}_{r2}, o, a) \wedge O(u^{q}_{r2}, o, a)) \quad (53)$$

If viewed as purely inherited (no origin tags), it is clear that (48), (49), (50) and (51) conflicts with (52) and (53). This induces a new global usod for both r_1 and r_2 that explicitly focuses compliance given the original permissions. Equation (54) shows the induced SoD between u^{m}_{r1} and u^{p}_{r2}. Similarly, user SoD's are created between all four users in the system, as seen in the Constraints for r_3 in equation (56).

$$\text{usod}^{r3} : \forall o, a, m, p, u^{m}_{r1}, u^{p}_{r2}, m \neq p \text{ , } \neg(O(u^{m}_{r1}, o, a) \wedge O(u^{p}_{r2}, o, a)) \quad (54)$$

The above usod (equation (54)) is added to the constraint set of both the roles r_1 and r_2 as a conflict resolution strategy. This causes (48), (49), (50) and (51) to be partitioned into mutually exclusive obligations. Thus, the permission and constraint sets for r_1 and r_2 are:

$$\begin{aligned}\text{Permissions} = \{ &\{\forall o, a, m, u^{m}_{r1} : O(u^{m}_{r1}, o, a)\}, \{\forall o, a, p, u^{p}_{r2} : O(u^{p}_{r2}, o, a)\}, \\ &\{\forall o, a, m, u^{n}_{r1} : O(u^{n}_{r1}, o, a)\}, \{\forall o, a, p, u^{q}_{r2} : O(u^{q}_{r2}, o, a)\}\}\end{aligned} \quad (55)$$

$$
\begin{aligned}
&\text{Constraints} = \{\\
&\{\ \forall o, a, m, n, u^{m}_{r1}, u^{n}_{r1}, m \neq n\ ,\ \text{usod}^{r1}\colon\ \neg(O(u^{m}_{r1}, o, a) \wedge O(u^{n}_{r1}, o, a))\ \},\\
&\{\ \forall o, a, p, q, u^{p}_{r2}, u^{q}_{r2}, p \neq q\ ,\ \text{usod}^{r2}\colon\ \neg(O(u^{p}_{r2}, o, a) \wedge O(u^{q}_{r2}, o, a))\ \},\\
&\{\ \forall o, a, m, p, u^{m}_{r1}, u^{p}_{r2}, m \neq p\ ,\ \text{usod}^{r3}\colon\ \neg(O(u^{m}_{r1}, o, a) \wedge O(u^{p}_{r2}, o, a))\ \},\\
&\{\ \forall o, a, m, q, u^{m}_{r1}, u^{q}_{r2}, m \neq q\ ,\ \text{usod}^{r3}\colon\ \neg(O(u^{m}_{r1}, o, a) \wedge O(u^{q}_{r2}, o, a))\ \},\\
&\{\ \forall o, a, n, p, u^{n}_{r1}, u^{p}_{r2}, n \neq p\ ,\ \text{usod}^{r3}\colon\ \neg(O(u^{n}_{r1}, o, a) \wedge O(u^{p}_{r2}, o, a))\ \},\\
&\{\ \forall o, a, n, q, u^{n}_{r1}, u^{q}_{r2}, n \neq q\ ,\ \text{usod}^{r3}\colon\ \neg(O(u^{n}_{r1}, o, a) \wedge O(u^{q}_{r2}, o, a))\ \}\ \}
\end{aligned}
\tag{56}
$$

5 Conclusion

In this paper, we have interpreted security certification criteria for access control issues across multiple COTS components in an integrated system. We have taken commonly found conflicts, applied them to interdomain mappings, evaluated the resulting policies, and analyzed them using deontic logic operations. Deontic logic allows permissions to be represented unambiguously and indicates when permissions are in conflict. We exemplified SoD and Chinese Wall constraints with inheritance issues in respect to certification criteria. By utilizing this approach, COTS component certification becomes less vague and subjective while moving closer to an automated and logically evaluated process.

Acknowledgements. This work is supported in part by a U.S. AFOSR award FA9550-05-1-0374. The U.S. government has certain rights to this publication.

References

1. Ross, R., *Guide for the Security Certification and Accreditation of Federal Information Systems*, in *NIST Special Publication 800-37*. 2004.
2. *Department of Defense Information Technology Security Certification and Accreditation Process (DITSCAP)*. 2000.
3. Brenner, E. and I. Derado. *Specifying a Certification Process for COTS Software Components Using UML*. in *4th International Symposium on Object-Oriented Real-Time Distributed Computing*. 2001. Magdeburg, Germany.
4. Khan, K.M. and J. Han. *Deriving Systems Level Security Properties of Component Based Composite Systems*. in *Australian Software Engineering Conference (ASWEC'05)*. 2005.
5. *Common Criteria for Information Technology Security Evaluation*. 2005.
6. Wallace, D.R., L.M. Ippolito, and B. Cuthill, *Reference Information for the Software Verification and Validation Process*, in *NIST Special Publication 500-234*. 1996.

7. Allen, J.H., *CERT Guide to Systems and Netwrok Security Practices*. 2 ed. 2001: Addison Wesley.
8. Benferhat, S. and R.E. Baida. *Conflicts Resolutions of Prioritized Security Policies*. in *9th ACM Symposium on Access Control Models and Technology (SACMAT '04)*. 2004. Yorktown Heights, New York, USA.
9. Kamoda, H., M. Yamaoka, and S. Matsuda. *Policy Conflict Analysis Using Free Variable Tableaux for Access Control in Web Services Environments*. in *14th International Word Wide Web Conference*. 2005. Chiba, Japan.
10. Ray, I., et al. *Using UML to Visualize Role-Based Access Control Constraints*. in *9th ACM Symposium on Access Control Models and Technology*. 2004. York town, New York, USA.
11. Benferhat, S., R.E. Baida, and F. Cuppens. *A Stratification-Based Approach for Handling Conflicts in Access Control*. in *8th ACM Symposium on Access Control Models and Technology (SACMAT '03)*. 2003. Villa Gallia, Como, Italy.
12. Bieber, P. and F. Cuppens. *A Definition of Secure Dependencies using the Logic of Security*. in *Computer Security Foundations Workshop IV, 1991. Proceedings*. 1991. Franconia, NH, USA.
13. Cuppens, F. and R. Demolombe. *A Modal Logical Framework for Security Policies*. in *10th International Symposium on Foundations of Intelligent Systems*. 1997. Charlotte, North Carolina, Usa.
14. Al-Kahtani, M.A. and R. Sandhu. *Induced Role Hierarchies with Attribute-Based RBAC*. in *8th ACM Symposium on Access Control Models and Technologies (SACMAT)*. 2003. Como, Italy.
15. El Kalam, A.A., et al. *Organization Based Access Control*. in *4th International Workship on Policies for Distributed Systems and Networks (POLICY 03)*. 2003. Toulouse, France.
16. Anderson, A. *Core and Hierarchical Role Based Access Control (RBAC) Profile of XACML v2.0*. in *Oasis*. 2005.
17. Bhatti, R., et al., *X-GTRBAC Admin: A Decentralized Administration Model for Enterprise-Wide Access Control.* ACM Transactions on Information and System Security, 2005. **8**(4): p. 388-423.
18. Jensen, K., *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*. Vol. 1. 1992: Springer-Verlag.
19. Shafiq, B., et al., *Secure Interoperation in a Multidomain Environment Employing RBAC Policies.* IEEE Transactions on Knowledge and Data Engineering, 2005. **17**(11): p. 1557-1577.
20. von Wright, G.H., *Deontic Logic.* Mind, New Series, 1951. **60**(237): p. 1-15.
21. McNamara, P., *Deontic Logic*. Spring Edition ed. The Stanford Encyclopedia of Philosophy, ed. E.N. Zalta. 2006.
22. Kagal, L. and T. Finin, *Modeling Conversation Policies Using Permissions and Obligations.* Journal of Autonomous Agents and Multi-Agent Systems, 2006.
23. Cuppens, F., et al., *Merging Regulations: Analysis of a Aractical Example.* 16th International Journal of Intelligent Systems, 2001.
24. Kamoda, H., et al. *Policy Conflict Analysis Using Tableaux for On Demand VPN Framework*. in *6th IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*. 2005.

A Report of the Current Situation on Software Certification in Japan

Daichi Mizuguchi* Satoshi Matsuoka †

Abstract

In Japan, the concern about software certification is increasing mainly in the following three industrial fields: the safety control equipment industry, the automotive industry, and the measuring instrument industry. In this paper, we report about the current situation of the three industries on software certification. We also report about our activities for software certification in CVS-AIST.

1 Introduction and background

Software is ubiquitous now, and it is an important component of the modern social infrastructure. Therefore, it is a necessary and pressing need to ensure the quality of software for the people of today to lead a safe and secure life.

As a means for that, several international standards that step into the quality and performance of software have been published in succession.

In response to such an international trend, the concern about those standards and conformity certification to them is increasing not only in the related Japanese companies but also in several Japanese governmental agencies.

In such a situation, CVS-AIST, which is short for Research Center for Verification and Semantics in National Institute of Advanced Industrial Science and Technology, has been conducting investigation and research activities on software certification for several years.

In this short paper, we report the situation in Japan around the software standards and their certification, and also introduce the activities for software certification in CVS-AIST.

2 Increasing concern to software certification in Japan

Our survey found that the concern to software certification is very high especially in the following three industrial fields in Japan: the safety control equipment industry, the automotive industry, and the measuring instrument industry.

In each of the three industries, an international standard which specifies the requirements about the quality and the performance of software built into a product has been, or is planned to be, issued. In addition, certification systems based on these standards are being established mainly in Europe.

This means that the products which do not acquire certification may be excluded from the Europe market. Not only companies but also the Ministry of Economy, Trade and Industry in Japan begins to have a sense of impending crisis about this.

In the following, we explain the present situation of these three industrial fields.

2.1 The safety control equipment industry

The international standard IEC 61508 – Functional Safety of electrical/electronic/programmable elec-

*Research Center for Verification and Semantics (CVS), National Institute of Advanced Industrial Science and Technology (AIST)

†National Metrology Institute of Japan (NMIJ) & Research Center for Verification and Semantics (CVS), National Institute of Advanced Industrial Science and Technology (AIST)

tronic safety-related systems was published in 1998 through 2000 [1]. This is a standard for computer-based safety systems for maintaining safety, for example in a plant and a factory [1].

IEC 61508 is a huge standard which consists of all seven parts. It covers all the life cycle of a safety system from risk analysis, through development and installation, till disposal of the system. The design and the development of software are incorporated in the life cycle, and the whole of part 3 of the standard is spent for the requirements of software only.

The standard recommends the traditional V-model as a development process of software. And for every phase of the process, such as the design, development, verification, and validation, it recommends to use techniques and measures according to the degree of the safety integrity level (SIL). The use of formal methods is recommended generally, and is highly recommended for the highest safety integrity level (SIL 4).

As the influence of the standard spreads, Japanese safety control equipment manufacturers, who are producing sensors, controllers, network equipments, and so on for safety-related systems, have been gradually forced to develop products along IEC 61508 in particular for the European market. Actually, several Japanese companies have already developed and released products which were approved as conforming to IEC 61508.

At the same time, however, there are voices to point out the problems with the conformity certification to this standard. We have often heard of the opinion that understanding the requirements is difficult, and the evaluation criteria for the requirements are not clear enough. Companies aiming to achieve the certification of their products are perplexed because they cannot understand which of the recommended measures and techniques they should/must use and to what extent. In fact, the assessment results can vary depending on the assessors.

In addition, there is no certification body of the functional safety based on IEC 61508 in Japan up to now. Japanese companies cannot help depending on a European certification company to achieve the certification. Therefore the expense and the difference of the languages can become big obstacles to the achievement of certification. So there is a demand for the establishment of a domestic certification organization with a certification criterion which considers Japanese way of development.

[1] In Japan, JIS C 1508, which is a translation standard of IEC 61508, was established by Japanese Industrial Standards Committee in 1999 through 2000.

2.2 The automotive industry

The international standard ISO/WD 26262 – Road vehicles – Functional safety is under deliberation. This standard is the adaptation of IEC 61508 to the automotive industry [2]. Now, it is still in the stage of working draft and is planned to be established in 2008.

The scope of this standard is the computer-controlled safety related systems in a road vehicle, such as the break, steering, powertrain, hybrid vehicle, air bag, and so on. The scope will expand more and more when the so-called by-wire technology and the cooperative control with the car navigation system are advanced.

The automotive industry in Japan points out several problems to this draft standard. Like the opinion on IEC 61508, there is an opinion that since the judgment criteria of certification are not clear, it is a standard for an assessor rather than for developers' sake. Also, since the already existing softwares such as case tools, compilers, and OS's are also the target of the safety proof, there is an opinion which worries about the influence over the development when they can not be proved to be safe enough.

Other opinions point out more fundamental problems. One of them is that, though this is the standard related to safety, the requirements included are not for safety, but for reliability of software, and argument on safety of software should be done more thoroughly. Another one is that, though the draft standard is claimed to the best practice for development, the grounds for that claim are not clear. This leads to the question that how we can be sure that a system that is certified as conforming to the standard is actually safer than the one that is not.

[2] See [2, 3, 4, 5, 6] for other sector standards based on IEC 61508.

The automotive industry in Japan has pride of having achieved excellent safety through the original way which goes along the Japanese culture. So now they seem to be trying to make the standard accept their way of development as much as possible before it is fixed and published. It is claimed that otherwise the standard could function as a new non-tariff barrier to trade.

2.3 The measuring instrument industry

There is a long history in the legal approval system of measuring instruments, for they must be fair for everyone at any time. WELMEC, the European Cooperation in Legal Metrology, has published several guides for software used in measuring instruments, such as weighing instruments, utility meters, and taxi meters, as they are getting controlled by more and more software [9, 10, 11].

These guides mainly concern the security aspects. For example, one requirement states that a computer program that handles measured data or values to be displayed or printed must be protected from corruption and must not be influenced by other programs. Also there are requirements for data transmission and software downloading.

In fact, some of the guides are for the Measuring Instruments Directive, MID for short, which is a directive for the European Union [8]. The MID has already been published and will soon come into force [3]. Under this trend, Japanese measuring instrument manufacturers who export their products to Europe are concerned how to satisfy the requirements in the MID and WELMEC guides.

Another recent movement is that one of the technical sub-committees of the OIML, the International Organization of Legal Metrology, is preparing a new international document, called "OIML D-SW", which specifies the requirements for software in measuring instruments.

When published, this international document will have influence not only on the measuring instrument manufacturers in Japan but also on the national notified body of the measuring instruments because, as a member nation of OIML, Japan has to harmonize its domestic laws with the document. NMIJ, the National Metrology Institute of Japan, will have to revise the related internal laws and standards, and also have to revise the approval system for measuring instruments mainly in order to include software as a direct target of the evaluation.

NMIJ has already been preparing for this task in collaboration with CVS-AIST for about five years [7].

3 Activities in CVS-AIST

3.1 About us

In each of the three industries, as described above, the situation is that Japan follows the standard led by the West later on, and the industries have been seeking for the way out.

In order to break this situation, we consider that a proper software certification system should be established with in the country and decided to carry out research on the software certification technique.

CVS-AIST mainly conducts research on formal verification of software with two directions of the research. One is the theoretical/academic research on formal verification methods, such as model checking and theorem proving. The research topics include model checking of a pointer system, categorical semantics of abstraction techniques, a unified verification environment with the theorem prover Agda at its core, and so on.

Another one is the case studies of formal verification methods. We actively conduct those case studies, which we call "fieldwork", with industrial partners in order to promote the spread of the formal methods in the industries. Our experience includes applying model checking technique to verify several embedded softwares and web-based systems.

Technical reports can be downloaded from the web site of CVS-AIST[4].

[3] The MID itself is now adopted as a WELMEC guide [12].

[4] http://unit.aist.go.jp/cvs

3.2 Software certification process

In the certification of software, only the completed program is not examined. The activities which a certification body performs in software certification contain the following:

1. Document review: Documents such as requirement specification, function specification, design specification, test reports, and so on, are reviewed.

2. Source code review: Some of the source codes are reviewed by human beings, possibly with tool support. Also, static analysis tools may be used.

3. Testing: Some of system tests are conducted. Unit tests of software modules may be conducted too.

4. Development process examination: Other than the products, the process of the software development including the management system of the process, documentation and quality control may be examined. Competency of the personnel involved may also be evaluated.

3.3 About our plan

Software verification technology is sure to play an important role in software certification. Moreover, we consider that it is necessary to use formal techniques actively in software certification in order to improve the accuracy and objectivity of the certification.

Setting this as one of the goals, CVS-AIST is now planning to start the "mock" certification project of software based on an existing standard like IEC 61508.

In this project, we set up a development project of a prototype system with industrial partners, and run both the development and a simulation of certification in parallel. Through this project, the developer is able to obtain the knowledge of how to develop along the standard.

From the view of certification, we hope to establish objective criteria for the certification. The judgment whether a product should be certified or not is done by an individual who has the responsibility in the end, based on the submitted evidence and explanation. Thus, it would be impossible to remove subjectivity completely in this judgment. However, in some of the software certification activities such as the source code investigation and testing, it might be possible to develop objective criteria of certification with formal techniques.

Furthermore, in this project, we could examine validity of the techniques and measures listed in the standards. This would be a good feedback to them.

Also the knowledge obtained through this project would be useful in developing a new standard for software and starting up a software certification body in Japan.

3.4 About our activities

For the moment, CVS-AIST is performing preparation activities for the mock certification project since last year.

The activities include survey of the standards, meeting people with the domain knowledge in the industry and experts on the standards, participation to seminars, investigation in the certification activities outside of Japan, and attendance to the domestic and international committees corresponding to standards.

We also held a one-day workshop titled "the functional safety standard and its certification" last February, to which we invited nine speakers from companies and universities. We had more than one hundred participants in the workshop.

4 Concluding remarks

As noted above, a proper scheme for software certification is eagerly needed in several industry sectors today. The needs will become greater and greater as the demand for certification will spread to other familiar domains, such as medical apparatus and home electronics. Our trial certification project is to try to answer these needs.

In general, it is indispensable for a certification system to obtain trust from the related industry and also those who use the certified products. For software

certification, we are aiming to make a contribution with formal approach in establishing and tightening the trust.

References

[1] IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, 1998-2000.

[2] IEC 61511: Functional safety – Safety instrumented systems for the process industry sector, 2003-2004.

[3] IEC 61513: Nuclear power plants – Instrumentation and control for systems important to safety – General requirements for systems, 2001.

[4] IEC 62061: Safety of machinery – Functional safety of safety-related electrical, electronic and programmable electronic control systems, 2005.

[5] IEC 62138: Nuclear power plants – Instrumentation and control important for safety – Software aspects for computer-based systems performing category B or C functions, 2004.

[6] IEC 62304: Medical device software – Software life cycle processes, 2006.

[7] Satoshi Matsuoka, *Integrity check of embedded software in weighing instruments via the Internet*, OIML Bulletin, Volume XLV, Number 2, pp. 14–18, April 2004.

[8] MID-software, Software Requirements and Validation Guide, version 1.00, 2004.

[9] WELMEC, Guide for Examining Software (Weighing Instruments), WELMEC 2.3, Issue 2, 2002.

[10] WELMEC, Guide for Examining Software (Non-automatic Weighing Instruments), WELMEC 2.3, Issue 3, 2005.

[11] WELMEC, Software Requirements on the Basis of the Measuring Instruments Directive, WELMEC 7.1, Issue 2, 2005.

[12] WELMEC, Software Guide (Measuring Instruments Directive 2004/22/EC), WELMEC 7.2, Issue 1, 2005.