

Challenges in Software Certification*

Tom Maibaum
May 2010

*

© Springer-Verlag

This technical report is a revision of the article "Challenges in Software Certification" (invited talk), Formal Methods and Software Engineering (ICFEM 2007), LNCS 4789, 4-18, Springer Verlag 2007, ISSN 0302-9743 (Print) 1611-3349 (Online); see <http://www.springerlink.com/content/a687h7n634k80u75/>



SQRL REPORT 59

MCMasterUNIVERSITY

Abstract. As software has invaded more and more areas of everyday life, software certification has emerged as a very important issue for governments, industry and consumers. Existing certification regimes are generally focused on the wrong entity, the development process that produces the artifact to be certified. At best, such an approach can produce only circumstantial evidence for the suitability of the software. For proper scientific evaluation of an artifact, we need to address directly the attributes of the product and their acceptability for certification. However, the product itself is clearly not enough, as we need other artifacts, like requirements specifications, designs, test documentation, correctness proofs, etc. We can organise these artifacts using a simple, idealised process, in terms of which a manufacturer's own process can be "faked". The attributes of this idealised process and its products can be modelled, following the principles of Measurement Theory, using the product/process modelling method first introduced by Kaposi.

1 Introduction

Software standards have been a concern amongst the software engineering community for the past few decades and they remain a major focus today as a way of introducing and standardising engineering methods into the software industry. Software certification, or at least certification of systems including software, has emerged as an important issue for software engineers, industry, government and society. One has only to point to the many stories of serious disasters where software has been identified as the main culprit and the discomfort that is being felt about this amongst members of these communities. Several organisations, including standards organizations and licensing authorities, have published guidance documents to describe how software should be developed to meet standards or certification criteria.

In this paper, we focus on the issues related to software certification and refer to standards only when relevant, though much could be said about the failures of software related standards to meet criteria characterising rigorous engineering standards. “These licensing organisations, through their guidance documents, aim to establish a common understanding between software producers and certifiers (evaluators). The US Food and Drug Administration (FDA) is one of these organisations. [The Common Criteria consortium, focusing on security properties of IT systems, is another.] The FDA has published several [voluminous] guidance documents concerning the validation of medical software [(as has The Common Criteria consortium on security properties). However, these] recommendations are not specified in an explicit and precise manner. In more detail, the FDA validation approach, as described in the FDA guidance document [6]:

- does not describe effectively the objects that are subject to assessment,
- does not specify the measurable attributes that characterize these objects, and
- does not describe the criteria on which the FDA staff will base their decision, in order to approve or reject the medical software [and, therefore, does not describe the measurement procedures to be used to ascertain the values of the relevant attributes of the objects being assessed].” See [9,10].

In fact, the focus of these documents is on the characteristics of a software development *process* that is likely to produce satisfactory software. It shares this approach and concern with almost all certification authorities’ requirements (as well as those of standards organisations and approaches based on ‘maturity’, such as CMM [17]). This seems to miss the point of the aim of certification, namely to ascertain whether the *product*, for which a certificate is being sought, has appropriate characteristics. Certification should be a *measurement* based activity, in which an objective assessment of a product is made in terms of the values of measurable attributes of the product, using an agreed objective function. (This objective function, defined in terms of the measurable attributes of the product, is itself subjectively defined; but once agreed, its use is completely objective, predictable and, perhaps most importantly, repeatable.) After all, we are not going to be happy if an avoidable disaster is down to a product being faulty, even though the process that produced it was supposed to deliver a sound product. A process can never provide this guarantee, if it does not actually examine relevant qualities of the product being certified. Even if the process is one that gives us correctness by construction (in the sense used in formal methods), mere correctness is not enough to convince us of the acceptability of the product. (For example, the specification on which the correctness assertion is based may be faulty. Or not all requirements have been taken into account. See [15,22,23].)

Hence, our hypothesis, boldly stated, is that process oriented standards and certification regimes will never prove satisfactory as ways of guaranteeing software properties and providing a basis for licensing, and we have to develop a properly scientific, product based approach to certification.

2 Process Oriented Standards and Certification

(See [9,10]:)

The Food and Drug Administration (FDA) is a public agency in the United States of America concerned with the validation of medical device software or software used to design, develop, or produce medical devices in the United States. In response to the questions about FDA validation requirements, the FDA has expressed its current thinking about medical software validation through guidance documents [6.7.8]. These documents target both the medical software industry and FDA staff. According to the FDA, validation is an important activity that has to be undertaken throughout the software development lifecycle. In other words, it occurs at the beginning, end and even during stages of software development.

[For example,] the FDA guidance documents recommend validation to start early while the software is being developed. In this sense, the FDA guidance document [6] considers other activities; like planning, verification, testing, traceability, configuration management; as important activities which all together participate in reaching a conclusion that the software is validated.

In essence, the FDA validation approach is a generic approach. It appears in the form of recommendations to apply some software engineering practices. These practices are considered to be working hand by hand to support the validation process. The reason behind FDA taking such a generic approach is due to the ‘variety of medical devices, processes, and manufacturing facilities’ [6]. In other words, the nature of validation is significantly dependant on the medical device itself. Examples of such validation determinant factors are [6]: availability of production environment for validating the software, ability to simulate the production environment, availability of supportive devices, level of risk, any prerequisite regulations/approvals re validation, etc.

[The recommendations in the FDA guidance documents aim to make it possible for the FDA to reach a conclusion that the software is validated. It applies to software] [6]:

- used as a component, part, or accessory of a medical device;
- that is itself a medical device (e.g., blood establishment software);
- used in the production of a device (e.g., programmable logic controllers in manufacturing equipment);
- used in implementation of the device manufacturer’s quality system (e.g., software that records and maintains the device history record).

[Having reached the conclusion that the software is validated increases the level of confidence in the software and, accordingly, the medical device as well.] In its guidance documents, the FDA recommends certain activities to be undertaken and certain deliverables to be prepared during the development of the medical software. These activities and deliverables are subject to [validation]. For [instance], validating the Software Requirements Specification (SRS), a [deliverable that contains all requirements], aims to ensure that there are no ambiguous, incomplete, unverifiable and technically infeasible requirements. Such validation seeks to ensure that these requirements [essentially] describe the user needs, and are sufficient

to achieve the users' objectives. In the same manner, testing is another key activity [that] is thoroughly described in the guidance. On the other hand, the guidance points out some issues that are interrelated as a result of the nature of software. Examples of such issues are: frequent changes and their negative consequence, personnel turnover in the sense that software maintainers might have not be involved in the original development. Moreover, the FDA guidance stresses the importance of having well-defined procedures to handle any software change introduced. Validation in this context addresses the newly added software as well as already existing software. In other words, in addition to validating the newly added pieces (components) of code, the effect of these new pieces on the existing ones has to be checked. Such a check ensures that the new components have no negative impact on the existing ones. Furthermore, the guidance highlights the importance of having independence in the review process, in the sense that the personnel who participate in validating the software are not the ones who developed it.

These are mainly the [kinds of] issues which the FDA guidance documents address with regard to software validation. In terms of software development[, the FDA approach does not favour any specific software development model.] [9] In other words, it leaves the choice of the approach to be used in developing the software to software producers themselves. This supports the fact that some organizations have their own policies, standards and development approaches [that] must be followed. [Furthermore, it] supports the fact that some approaches may well suit certain types of projects or software. Therefore, the FDA leaves the choice of the software development model to software producers, as long as it sufficiently describes the lifecycle of the software. However, [it is explicitly required] that validation occurs throughout all stages of the software [development model (approach)]. In this context, the guidance states that the magnitude of validation effort, expressed in terms of the level of coverage, is relative to the complexity and the safety risk which the medical software introduces.]”

In summary, the FDA guidance documents attempt to prescribe very detailed guidance on the nature of the software process to be used. This focus on process is sometimes lost and the guidance documents go into details of the products of the process. However, the nature of the product is highly underdefined, at least in terms of the requirements of measurement, and nothing is said about how the evidence submitted to the FDA will be evaluated. These characteristics make the FDA's certification process lengthy, costly, subjective and, therefore, highly uncertain.

2.1 Faking it

We have been criticizing process oriented methods of certification on the basis that the evidence about the product is indirect and offers no proper guarantees of the kind we actually need. One might then ask whether process based ideas are of any use at all in certification activities. In order to answer such a question, we need to look at what evidence about the product is required to make a proper assessment of its certifiability. Before doing that in the next section, we will discuss an idea due to Parnas [26], though implicit in the work of Dijkstra and many others, about *faking* the software process. The main point Parnas was trying to make was that actual instances of a development process are likely to be imperfect. There is a lot of backtracking, correction and work arounds that do not conform to the process definition. However, at the end of the project,

one can fake the ideal execution of the project, an execution in which there is no backtracking, no fixing, no work arounds, etc.

The problem with process based guidance for certification, as in that of the FDA [6,7] (see also [8]) or the Common Criteria [1,2,3,4,5], is that the prescription of the process ‘mandates’ (the FDA does not legally mandate!) many mechanisms and procedures that are purely to do with managing the imperfections of the process. So, for example, the FDA recommends the development and implementation of a configuration management plan, or a problem reporting plan. As a certifier, why should one care about configuration management during development? (In contrast, one might very well worry about configuration management of a product once it is out in the field.) What does it have to do with the properties of the product actually delivered for certification? Similar comments may be made about bug reporting and fixing mechanisms. The only thing that matters is the qualities of the final version of the product seeking certification. (Of course, if one knew that these matters were not handled well during development, then this might provide circumstantial evidence about the potential quality of the final product. This might then influence the certification authority to look more carefully at evidence about the product. But, this is a secondary effect and we leave it for future discussion.)

On the other hand, we need some products other than the one seeking certification as part of the evidence being assessed. An obvious example is a requirements specification. Other examples are: a design specification, a document describing validation of the design against the requirements, documents relating to testing, documents proving correctness, etc. We could organize these pieces of evidence, various products, in terms of some simple, idealized development process. One candidate for such an idealized process might be a simple version of the waterfall model well known from software engineering texts. In this version, there would be no backtracking, and every stage would have to be completed before moving on to the next one. And this is where the faking it comes in. Whatever actual process one follows, or does not follow, the onus is on the organization seeking certification for their product to map their documents/evidence onto the ideal model and its products. This gives the certification authority a standard product (consisting of the actual product and associated other products/evidence) to ‘measure’ and decide on certifiability. So, the certification authority should not ‘mandate’ any particular development process, or the necessity of having a configuration management plan, or whatever. This has the added benefit for the software developer that its internal processes are up to them, as long as they can effectively map their products onto the ones required by the much simpler, faked process. They can then manage their process without having to undertake the difficult job of redemonstrating conformance to the process ‘mandated’ by the certification authority or standard

Of course, the certification authority has to decide what evidence is required and design the idealized process to ‘deliver’ this evidence. Hence, there is a fine balance between the level of detail in the idealized process and the weight of evidence necessary to make the certification decision. Striking this fine balance is probably the most difficult job the certification authority has to perform. An example of this idea in action, though not mandated by the certification authority, is the model used in the Ontario Hydro redevelopment of the Darlington nuclear power station safety system [29]. See Figure 1.

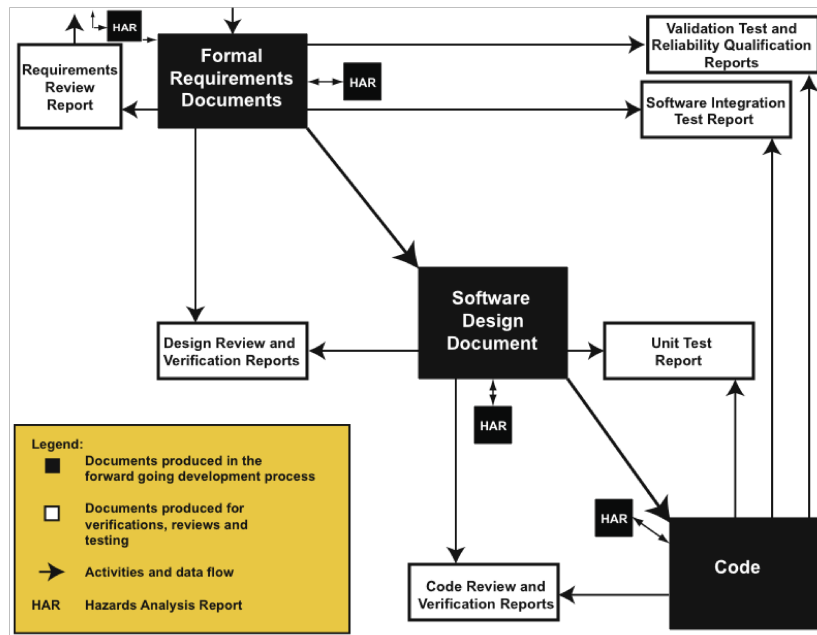


Figure 1: The Ontario Hydro idealized development process [29].

This is a simplified version of the figure that actually appears in [29] and is clearly insufficient, as it, stands to prescribe a real development process.

3 Measurement and Product/Process Modelling

“The “[Product/process (P/p)] methodology” [was introduced by Myers and Kaposi] in their book “A First Systems Book” ” [10] [18,24]. The purpose of this section is to provide a brief overview of this methodology. (See [20,21] for a more detailed description.) The language supporting the methodology provides abstractions for the two main constructs of the method: *products* and *processes*.

The basics of the method are described in [18,24], as is the relationship of the method to Measurement Theory. (See also [27,12] on measurement.) What we are modelling in many engineering problems is some real (existing or yet to be built) process and its associated products, technical or administrative (or both). Either this process is in place and we are modelling what we observe or we are intending, via the model, to prescribe what we eventually intend to observe in the organisation. The observed or intended phenomenon is called the *empirical referent* (and is to our endeavours what specific physical phenomena are to physicists attempting to understand the world by building scientific theories/models).

The language we use for modelling processes has three basic constructs via which the empirical referent must be modelled: products, processes and, an artefact of the method, gates. The first of these, *product*, is used to model the entities manipulated by processes. A product is an ‘instantaneous’ entity, in that it represents the measurable attributes of the entity at a specific moment in time. On the other hand, a process is an entity that relates to behaviour and so represents a phenomenon taking place over time. Gates are, to some extent, an artefact of the

method. Processes are modelled via ‘single input, single output’ transformers and so require products to be assembled (input from several preceding processes) and disassembled (to send parts of the output to separate subsequent processes).

One may recognise here strong relationships to business process modelling. Business process modelling is, of course, not a new subject! Many consultants make big money out of it! Our focus here is on the problem of modelling itself, with a particular focus on technical processes and requiring levels of detail in process definition which enable the methods and principles of measurement theory to be applied for the purposes of analysis, prediction and certification. A standard reference to approaches to business process modelling is [16].

The idea of characterising formally the process of software development is not new either. The early work of [25,19], etc is notable in this regard and relevant recent material on process modelling in software engineering can be found in [13]. The spirit of what we are attempting is very much in the style of [25], i.e., characterising processes as programs. We see the major difference as being the larger domain of processes being modelled, the much more powerful and expressive language being used and the focus on measurement and measurability. Also, software engineering has moved on since the time of this work and concepts such as patterns and software architecture enable a more sophisticated approach.

(From [9,10]:) Fenton and Pfleeger defined processes, [in the context of software engineering,] in [12], as ‘collections of software-related activities’. Hence the process usually has a time factor, i.e., it has a clear beginning and end. Attributes of the process are the inherent characteristics of the process. These attributes are meaningful descriptions over the process lifetime. Fenton and Pfleeger in [12] use the term “internal process attributes” to describe the attributes which can be measured directly by examining the process [definition] on its own. We will refer to these attributes as static attributes. Having identified these attributes, the evaluation [procedure, used by a certifier,] should [consider the procedures used] to measure each of these attributes. The evaluation may take place at predefined time checkpoints or even continue over a time interval.

In contrast, dynamic process attributes are those attributes that can only be measured with regard to the way the process relates to its environment, i.e., the behaviour of the process is the main concern of the measurement activity, rather than the process definition. The values of these process attributes are not meaningful outside their operating environment. Examples of such attributes are: quality and stability. The values of dynamic process attributes (external attributes in [12]) may depend on some values of the static process attributes.

On the other hand, Fenton and Pfleeger defined products in [12] as ‘any artifacts, deliverables or documents that result from a process activity’. We will refer to a product as a deliverable of a process. In a sense, we will overload the word deliverable to mean any outcome of the process. Examples of deliverables (products) are: Software Requirements Specification (SRS) document, [Software Design Specification document, software source code or some other] intermediate outcome of the development process. Products are atemporal in the sense that product attributes can be measured at any time instant (though the measured values may differ from one instant to another). The same notion of static and dynamic attributes is applicable to the deliverables (products). To be more concrete,” [9] “the version number (as a static attribute) of the software (as a product) is related to the product itself. Whereas, reliability (as a dynamic attribute) of the software (as a product) is relative to the way in which the product (software) behaves in its operating environment. [(There is a potentially very interesting discussion to be had here about a program or process being a *product*, e.g., the

code, and a *process*, e.g., the behaviour the program describes. But, we will not go there in this document.)]”

To achieve objective judgment of the evaluation evidence (deliverables), attributes have to be specified for products. This involves two main points: defining the *measurement scale* for the attribute, i.e., its type, and an effective *measurement procedure* for ascertaining the value of the type. Once the deliverable’s attributes are specified, an acceptability criterion for each attribute has to be established. Such a criterion will define the acceptable attribute values, from the point of view of the evaluation. Often, it is not the value of a specific attribute that determines the acceptability of the product, but the result of some utility function applied to some or all of the attribute measures of a product. Having both the measurement procedures and the acceptable values documented at the outset of any “deliverable (product) development” will facilitate the development, interim validation and the formal evaluation of these products. In the same manner, tools and machines that may be used in software production may have to be evaluated. Tools are considered as entities with attributes, exactly as for products (i.e., they are products), and hence their attributes have to be specified and ‘measured’.

(From [9,10]:) Having such a measurement framework defined will decrease [the level of] subjectivity in the evaluation activity. The FDA approach lacks such a measurement framework. The FDA approach has no explicit definition of what entities (processes or products) are to be measured. Hence the developers of the medical software and the evaluators from the FDA side share no common understanding about what evidence will be inspected, what attributes in this evidence will be measured, what values are acceptable and what values are not. [Similar comments may be made about the Common Criteria, though a better attempt is made in identifying products of development processes and determining what the evaluator must do, though not necessarily providing specific enough criteria and procedures to do it.]

[(Motivated by the idea that] processes and products are the key objects of measurement, Basili, Caldiera and Rombach in [11] developed the Goal-Question-Metric (GQM) [approach to help engineers develop models based on these kinds of ideas]. GQM is an engineering approach effective for the specification and the assessment of entities. The GQM model is hierarchical in the sense that it is layered into three main levels: the conceptual level (goals), the operational level (questions) and the quantitative level (metrics[, i.e., measurement]). At the conceptual level, goals “specify the objectives to be achieved by measuring some objects (products or processes). At the operational level, questions (what, who and how) should be derived from each goal. Answers to these questions will determine whether the goals are being met or not. Finally, the quantitative level describes what type of data has to be collected for every question in addition to the measurement mechanism to ensure a quantitative answer for the assessment. The key advantage of the GQM is that it enables us to identify the key attributes along with their measurement “scales and procedures. These attributes are the ones [that] are identified as being important for achieving the objectives and goals. Fenton and Pfleeger also considered process maturity models[, such as CMM,] to be used hand in hand with the GQM model. As the GQM helps to understand why we measure an attribute in a process, the process maturity model suggests whether or not we are capable of measuring the process in a practical way. Thus this supports the applicability of the GQM model. The main reason for considering this maturity model is that not all processes are at the same level of maturity, i.e., processes vary in terms of the visibility of input products and output products.

The model is described in detail in [12]. In this context, we want to emphasize that processes with [clearly defined] input and output [products] are our main concern.)

4 The Common Criteria (CC) for Information Technology Security Evaluation: A potential model?

(From [9,10]:) The Common Criteria (CC) for Information Technology (IT) Security Evaluation is an international standard for specifying and evaluating IT security requirements [and products]. This standard was developed as a result of a cooperation between six national security and standards organisations in the Netherlands, Germany, France, United Kingdom, Canada and the United States of America. This cooperation aims to define an international standard in order to replace the existing security evaluation criteria in those countries. [(The consortium has been significantly expanded since its inception.)] The main reason for considering the CC is the more systematic and consistent approach[, as compared to the FDA, that] the CC follows in specifying security requirements and evaluating their implementation. [As we will illustrate, the CC falls into the trap of prescribing in detail development process standards, but, on the other hand, it does provide a semblance of being product and measurement oriented.]

In the CC, IT [security requirements] can be [classified into Security Functional Requirements (SFRs) and Security Assurance Requirements (SARs)]. SFRs are mainly concerned with describing the functionalities to be implemented by the final product, whereas SARs are concerned with describing the properties [that] the final product should possess. As per the CC terminology, the final product [that] has to be developed, and after that evaluated, is called the Target-Of-Evaluation (TOE). The TOE is defined in [1] as ‘an IT product or system and its associated administrator and user guidance documentation that is the subject of an evaluation’ [1]. The requirements [that] describe the TOE are specified in the Security Target (ST). The Security target is a document [that] is similar to an SRS document, in which functional requirements are specified using [2], and assurance requirements are specified using [3]. These requirements are categorized in [2] and [3], according to the CC taxonomy, into classes, families and components. A class describes the security focus of its members. In other words, families of the same class share the same security concern, but each has a security objective [that] supports that concern. Components in the same family share the security objective of their family, but differ in the level of rigour in which the security objective is handled.

For instance, the security focus of communication class (Class CFO: Communication), a security functional class defined in [2], is to ‘assure the identity of a party participating in a data exchange’ [2]. This class has two families with two different objectives, yet they share the same security concern (communication). Non-repudiation of origin (FCO NRO: Non-repudiation of origin) is the first family in this class, which aims to ensure that the ‘originator cannot successfully deny having sent the information’ [2]. The other family (FCO NRR: Non-repudiation of receipt) aims to ensure that the ‘recipient cannot successfully deny receiving the information’ [2]. Components in the same family solve the security problem as described by their family, but with different levels of rigour. In this sense, “non-repudiation of origin” has two components. The first component (FCO NRO.1 Selective proof of origin) solves the “repudiation of origin” problem in the sense that it requires the “relied-on security enforcer (software or hardware or[, in the case of CC level 7], both)” to have entities ‘with the

capability to request evidence of the origin of information' [2]. [On the other hand,] the other component (FCO NRO.2 Enforced proof of origin) also solves the “repudiation of origin” problem, but it requires the “relied-on security enforcer (software or hardware or both)” to always “generate evidence of origin for transmitted information” [2].

In this context, CC part 2 [2] describes the following security functional classes with their families and components: security audit, communication, cryptographic support, user data protection, identification and authentication, security management, privacy, protection of the Toe Security Functionality (TSF), resource utilisation, and trusted path/channels. On the other hand, CC part 3 [3] describes the following security assurance classes with their families and components: Protection Profile (PP) evaluation, Security Target (ST) evaluation, development, guidance documents, life-cycle support, tests, vulnerability assessment, and composition. As previously described, components of the same family share the security objective of the family, but they differ in the level of rigour of the implementation of that objective. In other words, the level of rigour is essentially determined by the components that describe the level of confidence required for particular security issues. According to the CC, the Evaluation Assurance Level (EAL) determines the level of confidence required in the TOE. The CC defines seven evaluation assurance levels. These are [3]:

- EAL1: functionally tested
- EAL2: structurally tested
- EAL3: methodically tested and checked
- EAL4: methodically designed, tested and reviewed
- EAL5: semiformally designed and tested
- EAL6: semiformally verified design and tested
- EAL7: formally verified design and tested

Each evaluation assurance level requires certain components of particular assurance families to be implemented. The correspondence between the evaluation assurance level and the components of assurance families in a class appears in the following table as given in [3]. The “assurance class” column lists the security assurance classes as defined in CC part three (CC part 3: Security Assurance Requirements [3]). Each of these classes has security assurance families that share the security concern with other families in the same class. The assurance families of each class are listed under the “assurance family” column along the assurance class row. For instance, the “Development” assurance class (ADV) has ADV ARC, ADV FSP, ADV IMP, ADV INT, ADV SPM and ADV TDS as its assurance families. It is the CC convention for any family to start with the class symbol (ADV for DeVelopment Class) followed by an underscore () and then a family symbol (ARC for security ARChitecture family, FSP for Functional SPecification family, IMP for IMPLementation representation family, INT for tsf INTernals family, SPM for Security Policy Modelling family and TDS for Toe DeSign family). The development class symbol (ADV) and all other assurances classes’ symbols (AGD for Guidance Documents, ALC for Life Cycle, ASE for Security Target evaluation, ATE for TESts and AVA for Vulnerability Assessment) start with the letter “A” in order to differentiate them from functional classes.

As described in the “evaluation assurance level summary” table below, each family has a set of components that share the same security problem but differ in the level of rigour of the solution. For example, ADV FSP family of the assurance class “development” has six components. As the table [indicates], the first component (1) is necessary for evaluation

assurance level one (EAL1). Whereas the second component (2) of the same family is necessary for evaluation assurance level two (EAL2). In the same manner, EAL5 and EAL6 both require the implementation of the fifth component (5) of this family, and so on. Finally, some cells in the table are left blank. [This] means that it is not required to implement any component from the given family in order to achieve that evaluation assurance level. Security functional and assurance requirements are specified in [2,3], respectively. They are specified in a generic way that [enables] customization.

Developing the security target starts with defining the level of confidence required in the product (software product in this case) as per the evaluation assurance levels. Having the level of confidence determined, the evaluation assurance level summary table imposes the specification and implementation of the security components in the ST and the TOE respectively. The requirements written in the security target, as taken from [2] and [CC 2006c], can then be customized to reflect some restrictions, such as organisation-specific or product-specific issues. Developing the security target using only the security functional requirements of [2] results in a CC part 2 conformant product. However, adding extra requirements to the security target, which are demonstrated to be needed by the ST developer, results in a CC part 2 extended product. The same concept applies to CC part 3 security assurance requirements.

| Assurance class | Assurance Family | Assurance Components by Evaluation Assurance Level | | | | | | |
|----------------------------|------------------|--|------|------|------|------|------|------|
| | | EAL1 | EAL2 | EAL3 | EAL4 | EAL5 | EAL6 | EAL7 |
| Development | ADV ARC | | 1 | 1 | 1 | 1 | 1 | 1 |
| | ADV FSP | 1 | 2 | 3 | 4 | 5 | 5 | 6 |
| | ADV IMP | | | | 1 | 1 | 2 | 2 |
| | ADV INT | | | | | 2 | 3 | 3 |
| | ADV SPM | | | | | | 1 | 1 |
| | ADV TDS | | 1 | 2 | 3 | 4 | 5 | 6 |
| Guidance documents | AGD OPE | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | AGD PRE | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Life-cycle support | ALC CMC | 1 | 2 | 3 | 4 | 4 | 5 | 5 |
| | ALC CMS | 1 | 2 | 3 | 4 | 5 | 5 | 5 |
| | ALC DEL | | 1 | 1 | 1 | 1 | 1 | 1 |
| | ALC DVS | | | 1 | 1 | 1 | 2 | 2 |
| | ALC FLR | | | | | | | |
| | ALC LCD | | | 1 | 1 | 1 | 1 | 2 |
| | ALC TAT | | | | 1 | 2 | 3 | 3 |
| Security Target evaluation | ASE CCL | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | ASE ECD | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | ASE INT | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | ASE OBJ | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| | ASE REQ | 1 | 2 | 2 | 2 | 2 | 2 | 2 |
| | ASE SPD | | 1 | 1 | 1 | 1 | 1 | 1 |
| | ASE TSS | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Tests | ATE COV | | 1 | 2 | 2 | 2 | 3 | 3 |
| | ATE DPT | | | 1 | 2 | 3 | 3 | 4 |
| | ATE FUN | | 1 | 1 | 1 | 1 | 2 | 2 |
| | ATE IND | 1 | 2 | 2 | 2 | 2 | 2 | 3 |
| Vulnerability assessment | AVA VAN | 1 | 2 | 2 | 3 | 4 | 5 | 5 |

We can observe that, though not cast in the terminology of product/process modelling, defining relevant attributes and measurement procedures, this looks very close to what we have been describing.

(From [9,10]:) The security assurance requirements are organized into action elements for the developer, action elements for the content and presentation of the submitted deliverable, and action elements for the evaluator.

The taxonomy of the CC describes Security Assurance Requirements (SARs) in terms of action elements for the developer, action elements for the “content & presentation” of the submitted evaluation evidence and as action elements for the evaluator. Each evaluator action element in [3] corresponds to an action that is detailed into work units in the Common Evaluation Methodology [4], a companion document to the CC documents, which describes the way in which a product specified using the CC requirements is evaluated. The work units describe the steps that are to be undertaken by the testing laboratory in evaluating the ST, TOE and all other intermediate products. If these products passed the evaluation of a testing laboratory authorized by the CC, they would be submitted to the national scheme in that country to be certified.

In this context, the CC requires that the developer *shall* provide the TOE for testing. The TOE *shall* be suitable for testing and the evaluator *shall* examine sources of information to support the identification of potential vulnerabilities in the TOE. After that the evaluator *shall* conduct penetration testing to “confirm that the potential vulnerabilities cannot be exploited in the operational environment for the TOE” [3]. The CC uses the auxiliary verb *shall* to refer to mandatory work that has to be undertaken to ensure the correctness of evaluation and, accordingly, the verdicts assigned to products. On the other hand, the CC uses the auxiliary verb *should* to mean “strongly preferred”.

As an example, we include a small fragment of the CC that relates to testing:

ATE_FUN.1 Functional testing

Dependencies: ATE_COV.1 Evidence of coverage

Objectives: The objective is for the developer to demonstrate that the tests in the test documentation are performed and documented correctly.

Developer action elements:

ATE_FUN.1.1D The developer shall test the TSF and document the results.

ATE_FUN.1.2D The developer shall provide test documentation.

Content and presentation elements:

ATE_FUN.1.1C The test documentation shall consist of test plans, expected test results and actual test results.

ATE_FUN.1.2C The test plans shall identify the tests to be performed and describe the scenarios for performing each test. These scenarios shall include any ordering dependencies on the results of other tests.

ATE_FUN.1.3C The expected test results shall show the anticipated outputs from a successful execution of the tests.

ATE_FUN.1.4C The actual test results shall be consistent with the expected test results.

Evaluator action elements:

ATE_FUN.1.1E The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.

So, we are a long way from the ideal described in the product/process modelling approach, but we see elements of the approach in the above description. There are lacunae, such as a definition for “shall confirm”, which is clearly referring to a (measurement) procedure that is supposed to

determine whether “information provided meets all requirements for content and presentation of evidence”.

5 Conclusions

Software certification is starting to appear on the agenda of various groups: governments, industry and consumers/citizens. Although it has existed as a requirement in some critical areas, the practice of certification still leaves a lot to be desired. Certification regimes tend to be focused on process oriented standards, expecting that good processes will produce artifacts with the right attributes for certification. But, at best, this is attempting to evaluate the worth of the artifact by using what lawyers might call circumstantial evidence. Lawyers and juries are rightly wary of convicting people for serious crimes based only on circumstantial evidence. This is more so when the crime involved is of a more serious nature, entailing more serious punishment. We should follow suit and be more and more wary of certification by circumstantial evidence when the artifact involved may have more serious consequences for society, individuals or organisations.

The concepts of measurement theory and the traditional engineering idea of modelling problems by using transfer functions, aka the product/process modelling ideas described above, provide a basis for defining much more rigorous standards for evidence and for the process of evaluating the evidence to make a certification decision.

There is much research to be done to enable us to put these ideas into action. Obvious questions include:

- Is there a generic notion of certification, valid across many domains?
- What, if anything, needs to be adapted/instantiated in the generic model to make it suitable for use in a particular domain?
- What simple process model is sufficient to enable the “faking” of real processes and providing a platform for evaluation by certification authorities?
- What is the difference between software quality, of a certain level, and certifiability?
- In what situations can we safely use process based properties as a proxy for product qualities?
- If we assume that both formal approaches and testing are necessary for demonstrating evidence of certifiability, what mix is to be used when? If we have levels of certifiability, as in the Common Criteria, how does this mix change with level?
- Since evaluating evidence about software is an onerous task, how can we assist an evaluator to perform their task by providing tools? (Amongst examples of such tools may be proof checkers (to check proofs offered in evidence), test environments (to re-execute tests offered in evidence), data mining tools to find “interesting” patterns in artifacts, etc.)

There are also cultural and political issues for software certification to deal with. Many software producers find the idea of software certificates anathema: witness the move in various jurisdictions to lower the liability of manufacturers from even the abysmal levels in place today. Governments are woefully ignorant of the dangers represented by the low levels (or non existent levels) of regulation in some industries, such as those producing medical devices, cars and other vehicles, financial services, privacy and confidentiality issues in many information systems, etc. However, the issue is much too large for us to ignore any longer.

Acknowledgements. The Natural Sciences and Engineering Research Council of Canada, McMaster University's Faculty of Engineering and the Department of Computing and Software provided support for this research. The work of the author's Masters student Marwan Abdeen contributed some material to this work. The author would like to thank many of his colleagues, in particular Alan Wassying and Mark Lawford, for fruitful discussions about certification.

6 References

1. Common Criteria for Information Technology Security Evaluation, part 1: Introduction and general model, version 3.1, revision 1, <http://www.commoncriteriaportal.org/public/files/CCPART1V3.1R1.pdf> (September 2006).
2. Common Criteria for Information Technology Security Evaluation, part 2: Security Functional Requirements, version 3.1, revision 1, <http://www.commoncriteriaportal.org/public/files/CCPART2V3.1R1.pdf> (September 2006).
3. Common Criteria for Information Technology Security Evaluation, part 3: Security Assurance Requirements, version 3.1, revision 1, <http://www.commoncriteriaportal.org/public/files/CCPART3V3.1R1.pdf> (September 2006).
4. Common Methodology for Information Technology Security Evaluation, version 3.1, revision 1, <http://www.commoncriteriaportal.org/public/files/CEMV3.1R1.pdf> (September 2006).
5. Common Criteria for Information Technology Security Evaluation, User Guide, <http://www.commoncriteriaportal.org/public/files/ccusersguide.pdf> (October 1999).
6. General Principles of Software Validation; Final Guidance for Industry and FDA staff, <http://www.fda.gov/cdrh/comp/guidance/938.pdf> (January 2002).
7. Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices; Guidance for Industry and FDA staff, <http://www.fda.gov/cdrh/ode/guidance/337.pdf> (May 2005).
8. Guidance on General Principles of Process Validation, May 1987, Reprinted, <http://www.complianceassociates.ca/pdf/Guide%20-%20Process%20Validation.pdf> (1993).
9. Abdeen, M. M., Kahl, W. and Maibaum, T.: FDA: Between Process & Product Evaluation, HCMDSS/MD PnP, to be published by the IEEE Computer Society Press (2007).
10. Abdeen, M. M.: A Model for the FDA General Principles of Software Validation, Masters of Applied Science Thesis, Department of Computing and Software, McMaster University (2007).
11. Basili, V. R., Caldiera, G., Rombach, H. D.: The Goal Question Metric Approach, Encyclopedia of Software Engineering, Wiley&Sons Inc. (1994).
12. Fenton, N. Pfleeger, S. L.: Software Metrics (2nd Ed.): a Rigorous and Practical Approach. PWS Publishing Co (1997).
13. Finkelstein A, Kramer J, Nuseibeh B.: Software Process Modelling and Technology. John Wiley (1994).
14. Haerberer A, Maibaum T.: The Very Idea of Software Development Environments: A Conceptual Architecture for the ARTS Environment Paradigm. In: Proceedings of ASE'98. Redmiles D, Nuseibeh B, eds. IEEE Computer Science Press (1998).
15. Haerberer, A., Maibaum, T.: Scientific Rigour, an Answer to a Pragmatic Question: A Linguistic Framework for Software Engineering. Proceedings of the International Conference on Software Engineering, ICSE 21, IEEE CS Press (2001) pp463-472.
16. Hammer M, Champy J.: Re-engineering the Corporation: A Manifesto for Business Revolution. Nicolas Brealey Publishing (1993).
17. Humphrey, W.: Managing the Software Process. Addison Wesley Professional (1989).

18. Kaposi A, Myers M. Systems, Models and Measures. Springer-Verlag, London, Formal Approaches to Computing and Information Technology (1994).
19. Lehman M. Process Models, Process Programs, Programming Support. Proceedings of 9th International Conference on Software Engineering . IEEE Computer Society Press (1987) pp14-16.
20. Maibaum T.: The Mensurae Language: Specifying Business Processes. Technical Report, King's College London, Department of Computer Science (1999).
21. Maibaum, T.: An Overview of The Mensurae Language: Specifying Business Processes. Invited talk, Proc. Third Workshop on Rigorous Object-Oriented Methods, York University, BCS EWICS (2000).
22. Maibaum, T.: Mathematical Foundations of Software. Future of Software Engineering, ICSE 2000, IEEE CS Press (2000) pp161-172.
23. Maibaum, T.: The Epistemology of Validation and Verification Testing. In: Testing of Communicating Systems, 17th IFIP TC6/WG 6.1 International Conference, TestCom 2005, Khendek F & Dssouli R (eds), LNCS 3502, Springer-Verlag (2005) pp1-8.
24. Myers, M., Kaposi, A.: A First Systems Book, Imperial College Press (2004).
25. Osterweil L.: Software Processes are Software Too. In: Proceedings of 9th International Conference on Software Engineering. IEEE Computer Society Press (1987) pp2-13.
26. Parnas, D.L., Clements, P.C.: A Rational Design Process: How and Why to Fake It. In: Proc. TAPSOFT Joint Conference on Theory and Practice of Software Development (1985) pp25-29. Also published as University of Victoria/IBM Technical Report No. 3, February 1985, 18 pgs.
27. Roberts F.: Measurement Theory, with Applications to Decision-Making, Utility and the Social Sciences. Addison-Wesley (1979).
28. Suppe F.: The Structure of Scientific Theories. University of Illinois Press (1979).
29. Wassyng, A., Lawford, M.: Software Tools for Safety-Critical Software Development. International Journal of Software Tools for Technology Transfer, Special Section The Industrialisation of Formal methods: A View from Formal Methods 2003. Vol. 8, Number 4-5, Springer-Verlag (2006) pp337-354.