Software Certification: Is There a Case against Safety Cases?

Alan Wassyng*, Tom Maibaum*, Mark Lawford*, and Hans Bherer*

McMaster Centre for Software Certification
Faculty of Engineering
McMaster University, Hamilton, Ontario, Canada L8S 4K1
{wassyng,lawford,bhererh}@mcmaster.ca, tom@maibaum.org

Abstract. Safety cases have become popular, even mandated, in a number of jurisdictions that develop products that have to be safe. Prior to their use in software certification, safety cases were already in use in domains like aviation, military applications, and the nuclear industry. Argument based methodologies/approaches have recently become the cornerstone for structuring justification and evidence to support safety claims. We believe that the safety case methodology is useful for the software certification domain, but needs to be tailored, more clearly defined, and more appropriately structured in analogy with regulatory regimes in classical engineering disciplines. This paper presents a number of reasons as to why current approaches to safety cases do not satisfy essential attributes for an effective software certification process and proposes improvements based on lessons learned from other engineering disciplines. In particular, the safety case approach lacks the highly prescriptive and domain specific nature that can be seen in other engineering specialities, in terms of engineering and analysis methods to be applied in generating the relevant evidence. Safety case approaches and corresponding methods should aim to achieve the levels of precision and effectiveness of engineering methods underpinning regulatory regimes in other engineering disciplines.

1 Introduction

Software certification is in the news; see, for example, [1] for an academic account, and the following online discussions for "popular" reaction: an infusion pump that had to be removed from the U.S. market [2]; a software defect that prevented the emergency stop on the Gamma-Knife [3]; and two instances in which software failure resulted in horrific damage caused by radiation machines [4,5]. From automotive recalls to radiation device malfunctions, actual and potential deaths caused by faulty software have woken people up to the fact that software embedded in devices of all kinds has the capability of both helping us

* Supported by the Ontario Research Fund, and the National Science and Engineering Research Council of Canada.

R. Calinescu and E. Jackson (Eds.): Monterey Workshops 2010, LNCS 6662, pp. 206–227, 2011.

and killing us, or, less dramatically, having serious consequences for individuals and the environment. It is quite obvious to many, if not most, people that software is an incredible enabling technology. It is so good, in fact, that there is now almost no new device/technology on the market that does not depend on software in some way, either for its function or for its design. We have also been remarkably successful in building huge numbers of software enabled devices, with a rather limited number of known serious problems. However, this is quite misleading, and has resulted in severe over-confidence, both on the part of manufacturers and the public at large. As software and devices become increasingly complex and safety features get further intertwined with functional features, the chance of creating serious disasters also dramatically increases. Every now and again, just to remind us, software faults in critical applications feature prominently in the world's news. In contrast, establishing software certification and regulation as the norm is not on most people's horizon, never mind establishing real improvements in the regulation of software based devices.

Motivated by the developments above, we have become interested in promoting the concept of licensing for software and systems containing software in order to provide the public and governments with greater confidence in the safety (and efficacy) of such products. We want to put software certification on a proper engineering footing, analogous to other engineering domains. A reasonable characterization of *Software Certification*, in the context we are referring to, is that it is a demonstrated assurance that the system has met relevant technical standards and specifications designed to ensure it will not endanger the public, that it can be depended upon to deliver its intended service safely and securely and that it is effective [6]. We also contend that the goal of certification is to systematically determine, based on the principles of science, engineering and measurement theory, whether an artifact satisfies accepted, well defined and measurable criteria [6].

Although there are some regulatory regimes applicable to software, experience has shown that most of these regimes have struggled with how software can and/or should be regulated. These existing regimes are often based on assuring the quality and safety of the software based on the fact that the producer of the software adheres to some defined process, often one that is realized in an international standard for development processes. Recently, we have detected growing awareness that this so-called *process based software certification* cannot deliver the assurance we need, and *product focused certification* is viewed as a significant advance.

In the past few years, we have seen *safety cases* assuming a more prominent role in regulatory regimes for software based systems. They are touted as *the* way to go for certifying the safety of systems, especially those enabled by software [7]. A good introduction to safety cases, their structure and to a methodology for their development can be found in [8]. Safety cases are touted as *the* way to go for certifying the safety of systems, especially those enabled by software [7]. Accordingly, recent years have seen tremendous effort expended on safety cases. For example, a start has been made to develop systematic processes and

formalisms that would help increase the level of confidence in the soundness of a safety case [9]. There is also interesting work on how to characterize the chain of evidence, and how to produce it, in making the argument in a safety case [10]. Work is also being done to extend safety claims to more general software properties [11], and a number of researchers have broadened the safety case into an *assurance case*, see [12] as well. To complete this brief review, approaches for using safety/assurance cases to certify adaptive systems [9], as well as generic software based systems [13], have also emerged in recent years.

We were impressed by what we read about safety cases and started looking for more information on them: theory as well as their application in practice. While doing this we started developing some misgivings about them. This paper re-examines the safety case approach, using lessons we should learn from other engineering disciplines as a criterion for evaluation - and we take as our example, Civil Engineering. This examination leads us to identify a number of potential problems in the use of safety cases, so that we can make suggestions on how to modify this approach so that it can be even more useful in certifying systems containing software.

2 Software Certification Approaches

In many cases, when we speak about *software certification*, we often mean that, in certifying the system behaviour of a software driven device, we need to pay specific attention to the behaviour of the software components and their interfaces to the physical device. In other cases, it may be that the software to be certified runs on generic hardware rather than embedded in a device.

Different software certification approaches exist and are usually characterized by their underlying philosophy as being either *process oriented* or *product oriented*. Nevertheless, the process versus product classifications need not be mutually exclusive, and attempts to partition software certification approaches based solely on the two properties could be misleading. Nothing prevents the use of both in a certification process. For example, the certification process could check that an approved process was followed and also provide an argument based on product evidence. Since process oriented certification regimes usually require compliance with some set of established standards, they are sometimes referred to as *standards based approaches*. Again, this could be misleading because nothing precludes a standard from being product focused. Similarly, it is not unusual to refer to product oriented certification regimes as *argument based approaches* because one has to come up with a convincing argument about one's product, often in the context of an argumentation theory based framework for presenting evidence; but, again, nothing *a priori*, excludes process considerations in the arguments.

In the context of argument based approaches, the way to construct the argument is surely not unique and is still under intense debate. In this context, we see recent proposals for goal oriented and template based presentations of arguments [14] (methods for realizing argumentation based approaches). Whether

the argument should be product focused, goal based, or evidence based is still an open question and this issue has given rise to more specific argument based approaches. In the balance between process and product emphasis in a given certification approach, we believe that the certifying agent should increasingly favour the product, as the criticality of the software increases, not because processes are irrelevant for the development of the product, but because they are not sufficient to prove anything definite about safety, effectiveness or correctness.

2.1 Process Based Approaches

Process based certification is common because we have been "able to do it": it gives us the illusion, rather than a guarantee, that we have produced a good quality product and, therefore, a safe system. It is a lot easier to evaluate conformance to a development process than it is to decide on attributes that distinguish between dependable and unreliable software products and to be able to measure the relevant attributes effectively. Because process based standards model the products of the process superficially, if at all, it is impossible to characterize properly the properties of the entity we are actually interested in, namely the application we have built. Any process based definition of quality ends up guaranteeing only that certain steps and activities were undertaken, but does not offer direct evidence of the presence of desirable properties. A high quality process is not necessarily a reliable indication of a high quality product. Nevertheless, it is important to note that we believe that it is essential that the product be built by an organization that has excellent processes and excellent people, because this is likely to result in good products. It follows that certifying agents will be interested in these aspects, but they can usually be audited in a straightforward manner, often by a third party with no specific knowledge about the products developed by the audited organization, or its potential problems in relation to safety. The fact that a process standard has been adopted by some regulators may be evidence that the regulator believes that the standard is based on an implicit (generic) safety case that justifies the quality attributes of products produced using the process. However, the fact that this (generic) safety case is implicit contributes directly to the problems regulators are experiencing in evaluating applications based on adherence to the process standard. The developer and the regulator end up talking at cross purposes because of the lack of this common understanding. See below for further discussion of this point.

As a result, because process based certification approaches are inherently unable to guarantee the quality (in many cases, safety) that a regulator requires in the actual product, we need to focus on evaluating attributes of the product itself. This has generated interest in product focused certification approaches.

2.2 Product Focused Approaches

We might characterise process based approaches as providing indirect justification of a product's required attribute values, as needed by regulators. In contrast

to the indirect justification of process based standards, the product(s) of the process contain the basis for the direct justification of claims to regulators. We refer to this focus on the attributes of the product, providing the basis for the direct justification of safety claims, as *product focused*, in contrast to the process based approaches discussed above. After all, it is the safety of the final product in which we are interested and not the efficacy or otherwise of the process used to produce the product. A product focused approach requires that the relevant attributes of the product can be modeled and that there are precisely defined measurement procedures in place to determine the values of these attributes for a specific instance. As we will see below, typical engineering domains have a very product focused approach to regulation, concentrating on the product itself, or on design artifacts directly related to the final product. This use of models and other design artifacts is made possible by the direct relationship of such design artifacts to the final product, underpinned by physical laws and the predictability inherent in these laws, as well as by the predictability of engineering methods in ensuring that crucial design properties are maintained.

It is often said that classical engineering is underpinned by the continuity principle, enabling the predictability of the artifact's properties from those of the models and design artifacts. While this may well be the case, the predictability of the typical engineering method is at least as crucial in ensuring that properties predicted on the basis of design artifacts will be realized in the engineering product. In this regard, the process standards often used in regulatory settings related to software are not proper engineering methods. They do not have the required detail and prescriptive qualities to support the requirement for maintaining essential properties of design artifacts through to final product. (An international process standard does not an engineering method make!)

3 What Do Civil Engineers Do?

For many years a debate has raged as to whether software engineering is really *engineering*. That debate still continues. We are convinced that software engineering should be a real engineering *profession*. It should be a *speciality* within engineering. Why? To answer that we have to understand a little about the differences between engineering and science - and the very practical reasons that led to the creation of engineering.

There are a number of differences between science and engineering that are of particular importance to any discussion related to software engineering [15].

- Scientists learn and/or discover what is true of our physical world. They learn how to confirm that these truths are, indeed, valid. They also learn how to extend this knowledge. Engineers also need to learn what is true of our physical world. They often learn about this from scientists. They learn how to apply that knowledge in the construction of artifacts that are useful, effective and safe for the general population to use. This often requires additional knowledge from many different domains. They also need to learn

the *engineering principles* applicable to their speciality that then enables them to build safe and effective products.

- Scientists have the validity of their work judged by their peers. Scientific results may be used by other scientists or by engineers who use the results to build something. The results on their own usually do not represent a danger to the public. On the other hand, one of the primary roles of the engineer is to build products that will be used by other people. In many cases, these products have to be constructed so that they do not pose undue risk to the public.
- A direct result of the danger inherently posed by the construction of engineered artifacts was the idea of the *licensed professional engineer*. Society recognized the problem of having unqualified people (people who did not know enough about the specific engineering domain) build potentially dangerous products. There is no similar regulation of scientific personnel.
- In many jurisdictions, not only are the people who build the products regulated (by law), but the products themselves may be subject to regulation that requires "proof" that the product is *safe for its intended use*. In addition, some jurisdictions (such as medical device regulation in the United States) require "proof" that the product is *effective*, i.e., it works as advertised.

It is also apparent that software products, both embedded in devices, and standalone applications, may pose significant risk to society. It is also obvious that software is pervasive, and the modern world depends on software for the functioning of commerce, transportation, entertainment, medicine, and the generation of electricity. Thus, an engineering speciality, called *software engineering*, built on knowledge from *computer science* and other domains makes perfectly good sense. So, why the debate? We believe that the debate is largely fuelled by our failure to model software engineering on other engineering specialities. This is not surprising since software engineering is relatively new, and the fundamental science it is dependent on is also relatively new. However, we have managed to make incredible progress in building software products that have literally changed the world we live in. So, we should not use its "newcomer status" too much as a reason not to have more consensus on the fundamentals of software engineering.

To help the discussion along, it will help to look at an engineering speciality that has a long history of building artifacts that are useful/essential to society, have to be built at reasonable cost, and have to be safe. We will regard this engineering speciality as representative of other engineering domains. So, after centuries of experience, how do *Civil Engineers* manage to build the modern infrastructure required by our civilization, protect us from environmental hazards, and do this at reasonable cost with an excellent (modern) safety record?

Well, Civil Engineers are famous for their development and use of *Engineering Codes*. For example, the *CSA Standard CAN3-A23.3, Design of Concrete Structures for Buildings* [16] (in conjunction with other standards) is used by Civil Engineers to ensure that concrete structures are safe for use. Before we look at some of the details regarding concrete structures, it is interesting and important to note a reminder in Section 1, General Requirements: *"(3) The*

National Building Code of Canada requires that certain reviews be carried out by the designer or another suitably qualified person to determine conformance with the design." This is clearly a built-in certification/verification.

The following examples, taken from *CAN3-A23.3*, are instructive because they enable us to highlight essential differences between typical engineering regulatory practice (as exemplified by civil engineering) and typical software regulatory practice.

Example 1 refers to Section 15 which relates to *Footings*. In particular, isolated footings and (some) combined footings and mats. In Section 15.4.1 we find guidance on loads and reactions, and details relating to shaped columns and pedestals. Rules for determining the moment in footings are included. The rules are clearly prescriptive, and have been calculated to be conservative. They also allow for simplified calculations in the case of more complex shapes - also pre-determined to be conservative. Section 15.3 is note worthy, in that it clearly does not preclude a more specific analysis. A more specific analysis may be less conservative and allow for a more "creative" approach. There are other examples in which the more conservative approach is mandated, even if a more "accurate" method is available. By comparison, software regulation is hardly ever prescriptive with respect to analysis methods. This is partly due to the lack of consensus with regard to *standard* analyses in software engineering.

Example 1

- Section 15.4.1 *"The external moment on any section of a footing shall be determined by passing a vertical plane through the footing and computing the moment of the forces acting over the entire area of the footing on one side of that vertical plane."*
- For example: 15.3 *"In lieu of detailed analysis, circular or regular polygon shaped concrete columns or pedestals may be treated as square members with the same area, for the location of critical sections for moment, shear, and development of reinforcement in the footings".* □

Example 2 refers to Section 19 which relates to *Shells and Folded Plates*. Interestingly, as we can see in this section, the standard is also prescriptive with respect to the analysis assumptions that must be used. It also includes specific requirements on materials. By comparison, if software regulation does not prescribe analysis methods, it is certainly not likely to prescribe assumptions related to analyses. There are a number of software analogies we could use that relate to material. For instance, we could think of the programming language(s) used in an application as a fundamental component in much the same way as material may be thought of in constructing something physical. Using this analogy, it may be surprising that although most software regulation may not prescribe a programming language, many jurisdictions and manufacturers have imposed restrictions on programming languages to protect against known unsafe programming constructs.

Example 2

- Section 19.2.1 *"Elastic behaviour shall be an accepted basis for determining internal forces and displacements of thin shells. This behaviour may be established by computations based on an analysis of the uncracked concrete structure in which the material is assumed linearly elastic, homogeneous, and isotropic. Poisson's ratio of concrete may be assumed to be equal to zero."*
- 19.3.1 *"The specified compressive strength of concrete, t'_c, at 28 days shall be not less than 20MPa"*; and,
- 19.3.2 *"For nonprestressed reinforcement the yield strength used in calculations shall not exceed 400 MPa."* □

Finally, Example 3 refers to Section 21 which relates to *Special Provisions for Seismic Design*. This rather lengthy section starts with three columns of notation and definitions, indicative of more complexity. Our motivation for selecting this example is that one of the reasons given for treating software differently from other engineering specialities, is the fact that software systems are typically quite complex. In fact, software systems are frequently amongst the most complex of human endeavours. Section 21 now reflects the additional complexity required to deal with constructing buildings that can withstand seismic events. For example, Section 21.4.4 deals with *Transverse Reinforcement*, and Section 21.4.4.2 specifies very prescriptive compliance requirements, for a problem that is inherently complex and could be solved in a large number of ways. In addition, the product can be checked for compliance both during and after implementation.

Example 3

Section 21.4.4.2

Transverse reinforcement, specified as follows, shall be provided unless a larger amount is required by Clause 21.7:

(a) the volumetric ratio of spiral or circular hoop reinforcement, ρ_s, shall not be less than given by

$$\rho_s = (0.12 f'_c / f_{yh}) \tag{21-2}$$

and shall not be less than that required by Equation (10-7);

(b) the total cross sectional area of the rectangular hoop reinforcement shall not be less than the larger of the amounts given by Equations (21-3) and (21-4)

$$\mathrm{A_{sh}} = 0.3 \frac{\mathrm{sh_c f'_c}}{\mathrm{f_{yh}}} \left(\frac{\mathrm{A_g}}{\mathrm{A_{ch}}} - 1 \right) \tag{21-3}$$

$$\mathrm{A_{sh}} = 0.12 \left(\frac{\mathrm{sh_c f'_c}}{\mathrm{f_{yh}}} \right) \tag{21-4}$$

(c) transverse reinforcement may be provided by single or overlapping hoops. Cross ties of the same bar sizing and spacing as the hoops may be used; and

(d) if the factored resistance of the member core is greater than the factored load effect including earthquake, then Equations (10-7) and (21-3) need not be satisfied outside the joint." □

The above examples show that *CAN3-A23.3*, like most engineering standards, imposes constraints and requirements on the product, and is prescriptive on the analysis process too. In comparison, most standards found in the software engineering domain adopt a risk-based approach and define a process-based generic approach for the software development lifecycle phases. One such standard that is emerging as a key standard for the functional safety of electrical/electronic/electronic programmable (E/E/EP) safety-related systems is IEC-61508, specifically parts 3 and 7, which relate to software requirements and techniques, respectively [17]. This standard assumes that it is not possible to prescribe techniques and measures that will be correct for any given application. It does, however, state that a primary objective of the IEC 61508 series is to facilitate the development of product and application focused standards, perhaps quite similar to *Engineering Codes* such as the one discussed above. Since testing is the most common way in which software engineers evaluate their products, let us examine what IEC 61508 says regarding the *Requirement for Software module testing* (IEC 61508-3, Section 7.4.7), which is part of the *Software design and development phase* (IEC 61508-3, Section 7.4). First, *outputs* of the given phase (or activity) are identified, such as the *test result*. Second, properties are identified:

(1) completeness of testing with respect to the software design specification;
(2) correctness of testing with respect to the software design specification;
(3) repeatability;
(4) precisely defined testing configuration.

Next, a list of techniques is *proposed*. For example, *dynamic analysis and testing* (DAT) is proposed. Then, the links between the technique and the properties are highlighted. In this case, it identifies that DAT will positively contribute to properties (1) and (2) but not to properties (3) and (4). This is followed by a more detailed description of suggested techniques related to DAT. For example, *test case execution from boundary value analysis* is proposed. Finally, specific guidelines drawn from experience, are given in Part 7 of IEC 61508. For example, *"the use of the value zero, in a direct as well as in an indirect translation, is often error-prone and demands special attention to zero divisor, blank ASCII characters, empty stack or list element, full matrix, or zero table entry."* As this example shows, the evolution of IEC 61508 is a small step towards more prescriptive and product focused standards or codes. However, as can be seen in the above discussion, even though this modern software standard sets out to be product and application focused, it falls far short of the type of prescription we see in other engineering disciplines.

The next section stresses some lessons we can learn from these examples.

4 What Can We Learn from Civil Engineering?

The kind of domain specific, tightly prescribed standard described above is essentially unfamiliar territory for software engineers and regulators working in

the area of software. We want to understand why analogous standards do not exist for software and we want to make the case that they should.

The following points relate directly to lessons we can learn from: i) the standard described in the previous section (and whenever the "standard" is mentioned in this section, it is that CSA standard on concrete design that we are referring to); and ii) civil engineering practice in general. In many cases throughout this discussion, Civil Engineering simply stands as an example of typical engineering disciplines. We also include a brief but pointed comparison with software engineering practice. The point of the comparison is to promote the idea that we can do better in the software domain and that, if we really want to have proper safety standards in software engineering, we should do better.

- In the balance between *safety* and *creativity/efficacy*, safety always wins. Architects and engineers are constrained, generally by statute, in what they are allowed to do. This is simply accepted as a way of life for everyone in the profession. This prescriptive regulation is also updated frequently to keep up with advances in the field. We should not underestimate the positive effect of smart prescriptive regulation. To comply with Canadian nuclear regulations, developers have to separate control and safety functions. The safety system must be completely independent of the control system. This has the effect of severely constraining the complexity of the safety system, leading to systems that can be mathematically defined and analysed. Contrast this with the almost complete absence of control of creativity in software and in safety cases! (See Bloomfield in Section 5.1.)
- The standard imposes constraints and requirements on the product. Compliance with these constraints and requirements can be determined objectively during and after completion of the implementation. This is because "compliance" is defined in the context of the standard scientific/engineering measurement framework (i.e., the MKS system of measures). There are well defined measurement procedures that can be used to determine whether or not some constraint or requirement has been met. Engineers typically do use them. Software professionals typically do not. More importantly, software standards impose much more stringent requirements on the development process than they do on the product.
- Although the standard is predominantly product focused, it is sometimes unashamedly prescriptive on the analysis process as well. In software we seldom encounter anything similar. There are some exceptions, as usual. For example, Modified Decision/Condition Coverage testing is mandated by a civil aviation software standard.
- The standard applies to all (concrete) products.
- Although this is not obvious from the referenced standard, Civil Engineering (like other engineering disciplines, e.g., Electrical Engineering) defines very limited interfaces. Even when very general interfaces are quite possible, standard interfaces cut the complexity of designs and implementations tremendously. Those simple interfaces allow technicians to assemble products without full-time guidance of the engineers. Software Engineering has

> done almost nothing to limit our freedom to create interfaces. There has been excellent work done to cope with the complexity of software interfaces, but it may have been wiser to look also for ways of defining more standard interfaces. This may still be worth examining.

To some degree, in contrast to the lessons we have been trying to learn from the classical engineering disciplines, safety (and assurance) cases have been put forward as a kind of panacea for software engineering. We look more closely at safety cases in the next section.

5 Safety Cases

One approach that exhibits elements of both process based and product focused certification, depending on the nature of the evidence being adduced in the case, is that of the safety case. A safety case provides a structure in which the producer makes *claims* related to the safety of the product, and presents a *justification* as to why the claims are valid, using *evidence* related to and/or derived from the product. Latterly, argumentation theory has been proposed as a way of better presenting/structuring the safety case. This may make the safety case more comprehensible, but there is no reason to believe that it adds anything to its success in demonstrating safety or efficacy. This last claim requires justification. Before going any further, we should clarify what we mean by safety case (or assurance case, as for the moment we will not distinguish them). The literature on safety cases unfortunately exhibits a confused usage of the phrase. A safety case can obviously refer to an artifact, or product in the sense we use it in this article, namely the document containing the safety claims and the justification of those claims. The phrase can also be used as a name for an approach to demonstrating safety claims, as in the "safety case approach". This usage refers to a methodology, i.e., a set or system of methods, principles, and rules for regulating a given discipline, as in the arts or sciences. For the safety case approach, the elements include hazard analysis, the use of claims, adducing evidence, argumentation principles, and so on. Finally, a well defined and prescriptive engineering method for constructing a safety case in a well defined domain may qualify as a method that falls within the safety case approach. An example of this usage could be the method of goal structured argumentation using argument templates, were it not for the fact that we do not believe that the existing literature describes anything amounting to a proper engineering method.

The certification goal in the Introduction points to what is missing from safety cases at all levels. The safety case approach, i.e., the methodology, is not sufficiently clear on the scientific and measurement foundations of the methods and principles of safety cases. The specific goal structured, argumentation method for safety cases is also not well founded in terms of scientific and measurement principles. Furthermore, it is not an effective engineering method, whose aim is to engineer safety cases in a reliable and predictable manner. Of course, the safety case artifact is itself deficient as it will not contain the relevant measurements and scientifically based justifications required to make the safety case

"sound". It is this lack of proper scientific and measurement principles underlying safety cases, at all three levels, which justify our scepticism about efficacy of safety cases as noted above. We would make the same comments about assurance cases, which subsume safety cases and address issues other than just the safety of the system under consideration.

5.1 Why Safety Cases?

In stark contrast to the evidence from the classical engineering disciplines, Bloomfield and Bishop present five points as to why their notion of the safety case approach is to be preferred to more prescriptive regulation [13]:

- To prevent safety from being seen as the responsibility of the regulator rather than the service provider.
- Prescriptive regulation typically comes from past experience and this may be inappropriate in technically innovative industries.
- Prescriptive regulation encodes best practice at the time it was written, and may eventually prevent developers from adopting best practice.
- Overly restrictive regulation may be viewed as a barrier to open markets.
- Prescriptive regulation can adversely affect the cost and technical quality of products developed to comply with that regulation.

We will consider these points in turn.

"To prevent safety from being seen as the responsibility of the regulator rather than the service provider." We have no quarrel with the first point at all! It is absolutely clear to us that the developer of a safety critical artifact must take responsibility for its safety. In fact, legally the developer has no choice, whether regulated or not. However, we do have a quarrel with the role that this statement plays in support of the argument against overly prescriptive standards. Prescriptive standards are perfectly consistent with developers taking responsibility for the safety of their product. We do not see why this is different for software compared with other specialties. Regulation occurs in all disciplines (and it is the system that gets certified, not the software). Prescription should cover a minimum safe set deemed necessary for ensuring the safety of the product.

"Prescriptive regulation typically comes from past experience and this may be inappropriate in technically innovative industries." Most engineers would be offended at the accusation that, just because they have highly prescriptive regulatory regimes, they do not innovate. However, as Vincenti points out [18], there is innovation, and then there is innovation. In the distinction he makes between normal design and radical design, we see an inherent distinction between controlled and predictable innovation in normal design, and some or all bets are off innovations in radical design. Normal design embodies the experience based prescriptive design principles beloved of engineers. They are beloved exactly because they are reliable and predictable. However, as Vincenti points out, Just because they are prescriptive does not mean that innovation is stifled. The kind of innovation that is well supported in normal design is incremental, controlled innovation. For example, improving engine performance in a car or an airplane

by 2% is almost always an example of such controlled innovation. Replacing the usual car engine with a Wankel engine is not. This is an example of radical design and the predictability of the innovation's properties is much lower than for normal design.

So, in actual fact, engineers should avoid innovation, at least of a certain kind, exactly because they want predictability and safety. Radical innovation should have a very high price attached to it and should be stifled to some degree so as to prevent tragic accidents.

"Prescriptive regulation encodes best practice at the time it was written, and may eventually prevent developers from adopting best practice." The remarks made above about technical innovation in products apply equally to the engineering methods used to produce them. The guarantees offered by normal engineering methods are well worth their value in providing support for safety engineering and do not prevent normal design innovation in the method. The engineering ethos encourages such improvement. However, the price of radical innovation in engineering methods is made clear; the burden of responsibility for radical changes, with respect to safety properties of systems, is clearly placed on the innovator. As it should be.

"Overly restrictive regulation may be viewed as a barrier to open markets." As argued above, there should be some restrictions and barriers on open markets to make sure that the chances of damage to individuals and society are minimised. One has only to point to financial regulation, or the lack thereof, that was the major contributor to the world's recent economic catastrophes. The whole purpose of certification is to prevent the open markets from foisting dangerous products on us.

"Prescriptive regulation can adversely affect the cost and technical quality of products developed to comply with that regulation." In the interest of keeping our response polite, we will only say that safety trumps productivity, and that prescriptive regulation does not automatically imply bad or overly prescriptive regulation. Also, as noted often, other engineering specialities are usually regulated through much more prescriptive standards than is software.

5.2 Some Weaknesses of Safety Cases

The freedom inherent in safety cases is appealing to software experts. It is appealing for exactly those reasons we believe that software engineering has resisted the constraints and discipline imposed in other engineering domains. As Vincenti [18] points out, engineers classically rely on established and recognised methods for designing artifacts, as the established method offers certain assurances about efficacy and safety. These assurances are backed up by standard analyses and measurement procedures associated with the relevant, specific engineering method being used for the design Vincenti calls this *normal* design, in contrast to *radical* design, where some element of a normal design method is absent, say because untried technology is being used. Software engineers have made very little attempt to develop a normal design culture, valuing their freedom to make the same errors over and over again! And to not learn from other software

engineers' experiences! From the regulators' point of view, this also makes their job well nigh impossible. Every submission is different, and so very difficult to evaluate in a systematic way. Regulators also need "normal evaluation" methods to work effectively and efficiently. Just as manufacturers have to use normal design principles for predictability of design, so regulators need to be able to apply normal evaluation procedures to reach reliable evaluation outcomes. It is not that radical designs cannot be submitted, but the evaluation process is then much more complicated and less certain in its judgement (see the discussion on stifling innovation in [13]). There is an obvious and important role for the safety case approach to play. In fact, we believe that safety cases should play this role in the certification of all products, be they software-enabled or not. This role is primarily as an overseer and organizer of safety-related principles and methods relating evidence and artifacts. Individual components within the safety case method should be domain specific and prescriptive certification strategies. For example, a safety case document for a medical device is likely to contain components that are software certification specific, electronic certification specific, etc.

A typical safety case document would be extremely time and resource consuming and potentially costly for certifying agents to evaluate. The fact that each safety case may be a *one-off* example means that certifiers would have to spend considerable time understanding the evidence and the importance of the evidence in each individual case. That is, the certifiers are left with the problem of rationally reconstructing the safety case method and the methodology/approach used in creating the safety case document. So, each safety case presents the certifier with a double scientific induction problem: one from safety case document to safety case method and the other from safety case method to safety case approach. This kind of induction problem is one of the most difficult kinds of problems for scientists to solve! They also have to spend significant amounts of time understanding the safety case structure, in each individual submission. Although ideas from argumentation theory have been proposed as a way of structuring safety cases, this does not actually address the issue at hand, as there is no concept of a "normal" or standard argumentation structure for a particular class of safety cases, domain specific or not. This may also result in an unpredictable submission process, much as we have at this time in many regulatory regimes. This difficulty may be ameliorated somewhat, but not completely, with the advent of safety case templates [19,13]. One might add that in those cases where a regulator relies on a process or quality standard, or both, as in the EU and Canada for medical devices, there is a third scientific induction problem faced by developers and regulators. The process/quality standard used is a stand in for a safety justification that motivated the structure and content of the standard. The idea is that if the standard is followed, then all the requirements of this implicit safety justification would be met. However, both the developer and the regulator have to guess why elements of the standard are required in relation to this unstated safety justification. This results in the need for the implicit safety justification to be induced by the developer and the regulator separately, possibly resulting in diverging interpretations of the implicit

case. This may explain to some degree the need for regulators, e.g., the U.S. FDA, to issue guidelines to accompany these standards - one can interpret the role of the guidelines as narrowing the gap between the interpretations of developer and regulator in relation to the implicit safety case. However, the present process/quality standards based approaches leave both regulator and developer with a triple scientific induction problem! How bad is that?

It is not good enough that the producer of the product supplies the evidence and the argument(s) in the safety case. What does matter is that the certifying agent cannot expect the same type of evidence and the same type of argument each time. The certifying agent thus has less chance of building expertise that will help in future submissions. This lack of expertise, or lack of appreciation of some of the finer points perhaps in the argument, may easily lead to the certifying agent not detecting a subtle flaw. Again, safety case templates could help alleviate this problem .

Argumentation frameworks, suggested for use in presenting safety cases, have some inherent flaws. An argument is not a derivation, as in logic. Hence, the methods of logic cannot be used, on their own, to decide whether an argument "demonstrates" its intended result. An argument has to be refined from its informal presentation in the safety case into a form that can be formally analysed [20]. There may be several or even many such refinements - there are inherent ambiguities in natural language based presentations of arguments. Does one check all of these refinements for "soundness" of the argument? If not all, then how do we choose which one? If we manage to analyse the refinement(s) and determine that we have a sound argument, how can we be sure that the argument formulation is appropriate to support the guarantee of safety? How do we judge that the safety of the artifact has been established by the argument? It seems to us that argumentation is essentially value free; here "value" is used in the moral or subjective sense. It may be that the concept of "explanation" from the epistemology of science may be a better basis for presenting safety cases. In scientific explanation, the issue is how to present a case for establishing in a scientific manner that some observed phenomenon can be logically demonstrated by certain known assumptions, certain scientific laws and scientifically understood procedures. On the other hand, a scientific explanation can also be used predictively. This may be the role related to safety cases - predicting a significant property of our software artifact and relying only on scientific/engineering principles.

Safety cases were designed to present compelling evidence of safety. In many instances, efficacy is also extremely important. The U.S. FDA, requires medical devices not just to be safe, but at least as effective as a similar device already on the market. The initial such device has to demonstrate, through clinical trials, that it does what it claims to do. The relationship between demonstrating these two properties is an interesting issue. There is almost always some tension between efficacy and safety, and safety cases were not designed to deal explicitly with this complication. In developing the shutdown systems for the Darlington Generating Station, demonstrating both properties was aided tremendously by

the separation of safety and control functions, significantly reducing the complexity of the safety system.

5.3 Standards Combining Process and Product Focus with Implicit Safety Cases

Another example that exhibits elements of both a process based and a product focused approach is the *Common Criteria* [21], developed for security certification. In the Common Criteria approach there are product definitions of varying degrees of exactitude. There are also definitions of measurement and evaluation procedures. What is missing is a proper product focus, due care and attention to the demands of measurement theory, and the demands of scientific explanation. Taking into consideration its included *evaluation methodology* [22], it may be regarded as an implicit safety case approach. However, the Common Criteria is more prescriptive than a typical assurance case approach, in that it specifies what products have to be produced for certification at a particular level, and provides the certifier with an evaluation methodology in [22] as part of the standard. Unlike engineering standards though, the evaluation methodology states that the "target audience . . . is primarily evaluators applying the CC [Common Criteria] and certifiers confirming evaluator actions; . . . developers . . . may be a secondary audience." The developer can infer from [22] what evidence to produce, though not necessarily how it should be produced. To the developer, the safety case for the product behind the standard remains implicit and is effectively done by the evaluator as part of the evaluation process.

An approach that we have championed is an extension of some of the ideas and approaches used in licensing the Darlington Nuclear Generating Station's Shutdown Systems in Ontario, Canada. The work done on the Darlington Shutdown Systems has actually been quoted as an example for what safety cases are designed to prevent [13]. In reality, the Darlington licensing activity that Bloomfield and Bishop refer to, was the original licensing of the systems in 1989/90. The licensing process was difficult for many reasons. Prime amongst these was that this was the first software based nuclear safety system to be built and licensed in Canada, and that both the regulators and the manufacturer had not developed a plan to deal explicitly with software issues. It thus transpired that regulator involvement in the software verification activities started after the system, including all the software, had already been developed. This initial licensing of the Darlington Shutdown Systems was described from the point of view of the regulators [23], and also by some of the team who performed the verification [24]. Our contention is that a safety case approach introduced at that stage would have fared no better. In fact, this is a good example of why we believe that of all the safety case approaches we have seen, the *assurance based* approach advocated by Graydon, Knight and Strunk [12] is likely to be the most successful. The difference here is that the safety case is used to drive development as well as build the safety argument. Nowadays, when we refer to the Darlington approach, it is to the methodology that was researched and implemented subsequent to the original verification. The software development (and verification) approach was

described briefly in [25]. As that approach was developed, it was discussed in detail with the regulatory authority in Canada. The resulting methodology enabled the regulators to conduct audits of evidence produced during the development process, significantly simplifying the certification process, and making it much more predictable for the manufacturer.

The foundations of that approach were laid in a *standard*, called the "Standard for Software Engineering of Safety Critical Software" [26]. This standard is reasonably prescriptive in defining the (quality) attributes that must be present in each of the major software documents that are mandated. This effectively defines what major steps in the software development process must occur, as well as how to judge whether the required attributes are present in the specified documents. Although it does not specify/mandate the actual process to be used, it does mandate that specific processes must be described in *lower-level process documents*. For example, [26] has a list of process documents that must be produced, that then govern the production of a specific project output. A small sampling of these includes: the *Software Requirements Specification (SRS)*, the *Software Design Description (SDD)*, the *Design Review Report (DRR)* and the *Design Verification Report (DVR)*. Each of these lower-level "standards" then mandates both a process to be followed, as well as the documentation that has to be produced. The lower-level standards were created so that the relevant software document governed by that standard would possess the quality attributes described in [26]. All of these standards together embodied an implicit safety case approach, although it was not viewed that way at the time.

The implicit safety case approach for the development of safety-critical software at Ontario Power Generation and AECL, in this case, is as follows:

1. The requirements are specified mathematically and checked for completness and consistency. A hazards analysis is required to document risks and especially to identify sources of single point failures. These hazards have to be mitigated in the specified requirements.
2. Compliance between requirements and software design is mathematically verified. This includes a software design *review*, that evaluates how well the design exhibits mandated attributes. A prime example of such a design attribute is that of *maintainability*. Specific criteria are used to evaluate this, based on *information hiding*. This means that designers have to be able to demonstrate why they think that anticipated changes will be accomplished by changes to single design modules.
3. Compliance between the code and software design is verified through both mathematical verification and testing. Compliance between code and requirements is shown explicitly through testing. However, there is an implicit argument of compliance between code and requirements through the transitive closure of the mathematical verification - code to design, and design to requirements.

It should be noted though that the Darlington Shutdown System was a safety system that under normal circumstances did not intervene in operation of the plant. The system's safe operation is a substantial part of its efficacy. The other

part of its efficacy is related to providing acceptable production, i.e. *not* intervening unless it is really required. It is also important to note that the implicit safety case approach actually includes more goals than does a typical safety case approach. The CANDU standard [26] was designed to demonstrate *correctness* of the implementation with respect to its requirements. In the example of the Shutdown Systems, we believe that demonstrating *correctness* is mandatory, since the complete system is a safety system. The modern movement away from trying to show correctness (primarily because we know that it cannot be achieved 100%) disturbs us. It is still not a bad goal!

5.4 Safety Case Improvements

As Bloomfield and Bishop [13] point out, there are a number of directions for the future development and improvement of safety cases. Mainly, they have identified the following directions:

- Safety Case Methodology Enhancement;
- Extension to Other Areas;
- Safety Case Structuring;
- Confidence.

Regarding methodology enhancement, we believe that the emphasis should be on prescriptive engineering methods that are domain specific, on domain and method specific analysis methods, and on well defined methods for combining analyses, i.e., the result of the combination defines some enhanced level of confidence over and above that engendred by the parts. Moreover, we need to examine how to strike an appropriate balance between analysis methods of different strengths/forms of guarantee, on the one hand, and a balance between direct product based evidence versus process based evidence, on the other hand. Another important issue is the problem of incremental certification, of which the COTS problem is just one instance. For safety case structuring, Bloomfield and Bishop [13] have identified interesting future directions. One of them is the use of diverse arguments and evidence. We believe that the diverse arguments idea is a mistake if it means that we should provide multiple arguments related to the same evidence: it confuses defence in depth within the system with different ways of arguing about the system. The latter is just confusing to the evaluator; it is demonstrating something by the bludgeon method!. If what was meant was, in reality, defence in depth, so that multiple justifications and evidence are provided to back up a single claim, then we agree wholeheartedly.

Different structuring methods for safety cases will certainly be appropriate in different domains and for differing levels of criticality. In each case, the structuring of the case must be derived from the requirements of a safety case for the combination of domain and level of criticality. This question of structure should be decided once and for all by the regulator so that there is a uniform understanding by applicants of what is required in an application for a licence.

Establishing levels of confidence goes to the very heart of the problems with safety cases as they are currently defined. The key ingredient missing, as noted

above, is the ability to make an objective, repeatable assessment of the confidence one should have in the safety of the system. The assessment procedure for determining one's confidence in a product's safety is no more and no less than a proper engineering method. This means that the main attribute we are interested in, the product's safety, must be a measurement based result, derived from directly measurable attributes by well defined functions appropriate for the purpose, i.e., the function is the result of a scientific analysis of the value to be attached to the various forms of evidence and to the functions used to combine them. This issue of determining levels of confidence in safety cases is "arguably" the most important issue for future investigation.

Early gains in improving safety cases could be made by removing the guesswork in using process based standards and actually reverse engineering the implicit safety case behind the use of the standard. Presumably, the regulators "trust" the process standard because they feel that it implicitly supports some sort of safety case; making this explicit would help all parties in standardizing requirements and expectations. The result will not be the ideal regime, but will be a big step towards it. At the very least, it removes one of the induction exercises identified in Section 5.2.

6 Conclusion

The safety case approach, and safety cases in particular, will probably play an important role in software certification. A concern is that current safety case approaches lack proper scientific and measurement principles, and it seems that many proponents of safety cases are overselling the method and its tools. One can look at our concerns and point to specific instances where the safety case will drill down to a level at which domain specific certification results are used, and quote these as an indication that safety cases do not preclude other certification processes at various steps in the safety case argument. However, if the safety case approach does not impose more prescriptive software dependent requirements for certification, we are not solving the basic software certification problem, or for that matter, the problem for embedded devices. It will still be possible, and maybe easy, for people to present apparently convincing arguments to substantiate their claims using evidence that they are free to choose. Of course, certifying authorities do not have to accept the validity of the evidence or of the argument - but that is no better, and in fact no different, from many certification regimes in existence today. Most of the arguments for less prescriptive regulation, some of which are presented in Section 5.1, seem unconvincing to us - and reminiscent of similar arguments over the years regarding software. Most of the arguments tend to favour creativity and "progress" over safety, which is exceedingly strange for safety cases. Nevertheless, an argument that seems to have some real merit is the one concerning responsibility. However, that has been dealt with adequately in other engineering jurisdictions and we see no reason why it should not be possible in software certification as well.

Moreover, the name *safety* case indicates the focus of the approach. However, in many domains it is not enough to produce a *safe* product. Other product

characteristics may compete with safety and therefore should be addressed in the certification process. For example, the FDA regulations explicitly state that medical devices must be proven to be both safe and *effective* and there seems always to be a tension between safety and efficacy. Another example comes from the nuclear industry where the nuclear power plant has to be proven *productive* in the sense that the generating station must not only be safe but it must also be cost effective in producing power. Therefore, concentrating on only one aspect certainly gets easier but it is also not realistic. If we take that thought to its logical extreme, the shutdown system for a nuclear generating station would be trivial to construct. It should always just shut down the plant. So, how do safety cases help us determine that the system is both safe and effective? We are not convinced that they can do this effectively.

Under the safety case approach, different improvements have been proposed recently. One of them is the generalization of safety cases to assurance cases where the latter can consider claims that not only relate to safety but could, for example, address effectiveness or productiveness. We believe that it is a step in the right direction, however, their scientific and measurement principles are, as in the case of safety cases, an open issue.

In conclusion, we believe that, in the future, an efficient certification process could be based on argument based evidence like safety cases or more generally assurance cases. But, similar to what is found in Civil Engineering, a "code" for building them must be defined. This code should be prescriptive, product focused, and should not only help and guide the product developers, but should also focus, reduce and clarify the audit process for the regulators.

References

1. Kornecki, A., Zalewski, J.: Certification of software for real-time safety-critical systems: state of the art. Innovations in Systems and Software Engineering 5, 149–161 (2009), http://dx.doi.org/10.1007/s11334-009-0088-1, doi:10.1007/s11334-009-0088-1
2. CBC Staff: Infusion pumps recalled in U.S. but not Canada. CBC News Online (May 2010), http://www.cbc.ca/news/health/story/2010/05/04/con-baxter-pump.html
3. Poulson, K.: Known software bug disrupts brain-tumor zapping. Wired (October 2009), http://www.wired.com/threatlevel/2009/10/gamma
4. Bogdanich, W.: Radiation offers new cures, and ways to do harm. The New York Times Online (January 2010), http://www.nytimes.com/2010/01/24/health/24radiation.html
5. Bogdanich, W., Rebelo, K.: A pinpoint beam strays invisibly, harming instead of healing. The New York Times Online (December 2010), http://www.nytimes.com/2010/12/29/health/29radiation.html
6. Software Certification Consortium: Software Certification Consortium Charter (Draft) (2010)
7. Jackson, D., Bloch, J., Dewalt, M., Gardner, R., Lee, P., Lipner, S.B., Perrow, C., Pincus, J., Rushby, J., Sha, L., Thomas, M., Wallsten, S., Woods, D.: Software for Dependable Systems: Sufficient Evidence? National Academies Press, Washington (2007)

8. Bishop, P., Bloomfield, R.: A methodology for safety case development. In: Redmill, F., Anderson, T. (eds.) Industrial Perspectives of Safety-critical Systems: Proceedings of the Sixth Safety-critical Systems Symposium, Birmingham, UK, pp. 194–203. Springer, Heidelberg (1998)
9. Rushby, J.: A safety-case approach for certifying adaptive systems. In: Proceedings of AIAA Infotech@Aerospace, Seattle, WA, American Institute of Aeronautics and Astronautics (April 2009)
10. Panesar-Walawege, R., Sabetzadeh, M., Briand, L., Coq, T.: Characterizing the chain of evidence for software safety cases: A conceptual model based on the IEC 61508 standard. In: 2010 Third International Conference on Software Testing, Verification and Validation, pp. 335–344. IEEE, Los Alamitos (2010)
11. Fong, E., Kass, M., Rhodes, T., Boland, F.: Structured assurance case methodology for assessing software trustworthiness. In: 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement Companion (SSIRI-C), pp. 32–33. IEEE, Los Alamitos (2010)
12. Graydon, P.J., Knight, J.C., Strunk, E.A.: Assurance based development of critical systems. In: DSN 2007: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 347–357. IEEE Computer Society, Washington, DC, USA (2007)
13. Bloomfield, R., Bishop, P.: Safety and assurance cases: Past, present and possible future - an adelard perspective. In: Dale, C., Anderson, T. (eds.) Making Systems Safer, Proceedings of the Eighteenth Safety-Critical Systems Symposium, Bristol, UK, pp. 51–67 (February 2010)
14. FDA Staff: Guidance for Industry and FDA Staff Total Product Life Cycle: Infusion Pump - Premarket Notification [510(k)] Submissions DRAFT GUIDANCE. U.S. Department Of Health and Human Services: Food and Drug Administration, Center for Devices and Radiological Health (April 2010)
15. Parnas, D.L.: Software engineering programs are not computer science programs. IEEE Software 16(6), 19–30 (1999)
16. Canadian Portland Cement Association Ottawa, Ontario, Canada: CSA CAN3 A23.3 M94 Concrete Design Handbook (1994)
17. IEC 61508: Functional safety of electrical/electronic/programmable electronic (E/E/EP) safety-related systems: Parts 3 and 7, International Electrotechnical Commission (IEC) (2010)
18. Vincenti, W.G.: What Engineers Know and How They Know It: Analytical Studies from Aeronautical History. The Johns Hopkins University Press, Baltimore (1993)
19. High, K.M., Kelly, T.P., Mcdermid, J.A.: Safety case construction and reuse using patterns. In: 16th International Conference on Computer Safety and Reliability (SAFECOMP 1997), pp. 55–69. Springer, Heidelberg (1997)
20. Parsons, T.: What is an argument? The Journal of Philosophy 93(4), 164–185 (1996)
21. Common Criteria for Information Technology Security Evaluation: Part 1: Introduction and general model. CSE, SCSSI, BSI, NLNCSA, CESG, NIST, NSA, Version 3.1 Revision 3 (July 2009)
22. Common Criteria for Information Technology Security Evaluation: Evaluation methodology, Version 3.1, Revision 3 (July 2009)
23. Parnas, D.L., Asmis, G.J.K., Madey, J.: Assessment of safety-critical software in nuclear power plants. Nuclear Safety 32(2), 189–198 (1991)

24. Archinoff, G.H., Hohendorf, R.J., Wassyng, A., Quigley, B., Borsch, M.R.: Verification of the shutdown system software at the Darlington nuclear generating station. In: International Conference on Control and Instrumentation in Nuclear Installations, Glasgow, UK, The Institution of Nuclear Engineers (May 1990)
25. Wassyng, A., Lawford, M.: Lessons learned from a successful implementation of formal methods in an industrial project. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 133–153. Springer, Heidelberg (2003)
26. Joannou, P., et al.: Standard for Software Engineering of Safety Critical Software. CANDU Computer Systems Engineering Centre of Excellence Standard CE-1001-STD Rev. 1 (January 1995)