Electronic Communications of the EASST
Volume 46 (2011)

Proceedings of the 11th International Workshop on Automated Verification of Critical Systems (AVoCS 2011)

Positioning Verification in the Context of Software/System Certification

Marc Bender, Tom Maibaum, Mark Lawford, Alan Wassyng

15 pages

Guest Editors: Jens Bendisposto, Cliff Jones, Michael Leuschel, Alexander Romanovsky
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Positioning Verification in the Context of Software/System Certification

Marc Bender, Tom Maibaum, Mark Lawford, Alan Wassyng

McMaster Centre for Software Certification (McSCert), McMaster University

Abstract: Formal verification applied to software has been seen as an important focus in research for determining the acceptability of that software for use. However, in examining the requirements for determining the safety of a software intensive system for use in critical situations, it is quite clear that verification plays a role, but not necessarily a *central role*. It is entirely possible that a piece of software satisfies its specification, but is unsafe to use. (The first and foremost reason for this is that the program satisfies an unsafe specification.) In this paper we will address the nature of certification in the context of critical systems, decomposing it, by means of a new philosophical framework, into four aspects: *evidence, confidence, determination* and *certification*. Our point of view is that establishing the safety (in a very general sense) of a system is a confidence building exercise much in the same vein as the scientific method; our framework serves as a setting in which we can properly understand and develop such an exercise. We will then place formal verification and assurance cases in this setting, discussing their roles and limitations.

Keywords: Software certification, System certification, Formal specification, Verification, Critical systems, Safety, Assurance cases, Safety cases

1 Introduction

"Beware of bugs in the above code; I have only proved it correct, not tried it."
– Donald Knuth

Formal (mathematical) verification has been an important and dominating focus in Computer Science and Software Engineering since the days of Turing [Tur49]. The main motivation for this is that formal verification applied to software has been seen, in academia, as the main focus in determining the acceptability of that software for use, particularly in relation to software that is used in critical applications. The purpose of verification in this context is the demonstration of the correctness of the software in relation to a mathematical specification. (A related and later notion is that of model checking: build a model of the intended software in the language of a model checker and then check whether the model enjoys identified properties or not.) Verification has been technically a highly challenging research area, and remains so to this day (as the attendees at this conference well know!). We might call this emphasis on the paramount importance of verification in guaranteeing the correct behaviour of software the *verificationist* view.

This perspective may be said to have reached its apotheosis, from the point of view of engineering of critical systems, in Shao's paper [Sha10], which appeared last year in CACM. In it, he states that software should be certified in isolation of its operating environment, and that proof is

tantamount to certification. We feel that this definition of "certified software" is seriously flawed. Our intent in this paper is to refute this definition as unsound in the context of software and systems engineering of critical systems and to provide a more satisfactory definition of *software (and system) certification* and the role of formal verification within this scope.

1.1 Software, engineering, and certification

In examining the requirements for determining the safety of a software intensive system for use in critical situations, it is quite clear that verification plays a role, but not necessarily a central role. It is exceedingly possible that a piece of software satisfies its specification, but is unsafe to use. The first and foremost reason for this is that the program satisfies the wrong specification. A second, and equally important reason, is that the formal models may not take into account important details of the platform on which the corresponding program executes or other important information about the environment of the program that may influence its behaviour. This information may include details of the operational procedures related to the system controlled by the software, the man/machine interface, hardware behaviour, timing requirements, including failures, etc. It is also a fact that verification technology is limited in scale, and it is infeasible to consider the verification of a complete system of any reasonable size (see challenge 2 in §2). (What this feasibility limit is at any one time is uncertain and the limit is expanding, perhaps even quickly. However, even if the complete verification of a system is possible, it may be economically infeasible.)

There are alternatives, involving ideas explored, for example, in the development of the safety systems for the Darlington nuclear power station built by Ontario Hydro (OH), now Ontario Power Generation (OPG) [WL03]. This involved a complete verification of the implemented code for the shutdown system with respect to its formalised requirements. The shutdown system was both physically and logically separated from the control system, which was certified to a significantly lower level of criticality. The shutdown system was tens of thousands of lines of code and was completely verified. The control system was around half a million lines. This separation of concerns made verification feasible, even with the technology of the time (early 1990s). Nevertheless, though verification of the shutdown system was an important part of the safety case made to the regulators, there was a lot more to it than that!

In this paper we will address the nature of certification in the context of critical systems, widely defined, and its reliance on concepts such as evidence, measurement, assessment criteria and the notion of confidence. The last is derived from the concepts discussed by epistemologists when they try to understand how the results of experiments designed to test a hypothesis derived from a scientific theory affect the strength of our belief in the validity of that theory. Our hypothesis is that establishing the *safety* (in a very general sense) of a system is a confidence building exercise very much in the context of the scientific method. We will then place verification of software in this context, discussing its role and its limitations.

2 Software Certification

For the moment, let us take certification simply to be some activity that involves the *evaluation* or assessment of engineering artifacts. (We will define exactly what we mean by *software* in §2.1 and *certification* in §3.) What makes evaluating software challenging? On this question, Shao makes three important points with which we strongly agree.

1. Software is everywhere. Software is being used in more and more safety-critical settings. The need to develop and apply dependability criteria to critical software cannot be overstated. How can we deal with the huge number of domains that we interact with?

2. Software is complex. Our ability to reason about the dependability of production scale software is already seriously lacking, and in the absence of a consensus on certification, we are only falling further behind. How can we make certification scale?

3. We lack metrics for software dependability. As Shao says, this makes it difficult to compare different techniques and make steady progress in the field. How can metrics be discovered and developed, and how can we incorporate them when they exist?

Shao's proposal for certified software addresses these challenges thus:

1. Ignore the domains; *certify the software in isolation*. Shao explicitly states, as his first key insight, that "the dependability of a software system should be treated separately from its execution environment", a claim he repeats twice in the paper. He also explicitly states that the execution environment "consists of not just hardware devices but also human operators and the physical world".

2. Shao is somewhat pessimistic, saying the following: "Constructing large-scale certified software systems is itself a challenge. Still unknown is *whether it can be done at all* (emphasis ours) and whether it can be a practical technology for building truly dependable software."

3. Consider the formal "dependability claim" as a metric. (He states that the dependability claim is a specification expressed in a general purpose mathematical logic.) He gives no indication of how this is to be done.

In our opinion, these views are all unacceptable. They highlight the problems with Shao's definitions of software, with which we are in some disagreement, and certification, with which we are in complete disagreement: briefly, he sees takes software to mean (machine) code and certification to mean formal proof. In the next section, we will present our own perspective on software certification as contrasted with Shao's. We will come back to the three challenges afterward.

2.1 What should *software certification* really certify?

Before we present our position on what software certification should be, let us look at what software comprises. There is a bit of a problem with nomenclature: the word 'software' is often taken to mean actual code on the one hand, and the field of endeavour on the other. By way of

analogy, it is as if we used the same word for edifice as we do for architecture. Shao's definition of software is definitely the former. But when we talk about software as an *engineering* artifact, we believe that the correct definition stretches much further into the latter, especially in the context of certification. A *software artifact* is the collection of all documentation involved in the production of the actual code (requirements, specifications, design descriptions), the code itself, and the executable machine code; analogously, an *architectural artifact* could be said to contain a list of materials, a blueprint, and an actual structure. Making claims about the dependability of a structure in the absence of these extra pieces of information is ludicrous. Our view is that the same is true of software: *software certification should certify software artifacts*, not just the code.

But accepting that formal documentation is a part of the software artifact means that certification must take into account the software's environment. Certifying the artifact should include checking the requirements, which means validating them against (a model of) the real world. As Lawford, Maibaum and Wassyng say in [WML10], this includes epistemological questions that are (generally) not subject to proof, but only to argument. The certification of software as an engineering artifact must take the imperfections of these aspects into consideration. We agree with Shao that software dependability becomes a much more difficult question once we take this step. We strongly disagree that the solution is to remove all but the sterile notion of formal proof from the factors that contribute to dependability.

Once we accept that dependability does not rely only on simple yes/no questions, certification becomes quite an interesting and challenging problem. This is the motivation for our *certification framework*, to which we now turn our attention.

3 A Certification Framework

What is certification? As mentioned above, it includes evaluation of engineering artifacts; furthermore, a successful evaluation results in a certificate being bestowed upon the artifact. This is a reasonable starting point but it is somewhat vague. To really understand what we are trying to do, a better question is: what is the *goal* of certification? Following [HHL+09], we take the goal of certification to be *to systematically determine, based on the principles of science, engineering and measurement theory, whether an artifact satisfies accepted, well defined and measurable criteria*. The challenge, then, is to develop a certification process that achieves this goal.

Our approach goes roughly along the following lines: we must evaluate pieces of *evidence* about the artifact, which systematically increase our *confidence* that it is satisfactory, until the point where we make the *determination* that the artifact is acceptable and assign a *certification* to it.

We use this simplified process as the starting point for our certification framework. It was while attempting to develop an idealized process of certification that this decomposition was elucidated; however, it gradually became clear that developing a satisfactory idealized process was very difficult. The main reason for this is that it is not at all clear what exactly we are *working with* when certifying a system: what are the raw materials? To answer this question, we took a step back and designed a certification framework with the goal of identifying and organizing the *aspects* of certification. One of the primary aims of the framework is to provide a setting that can

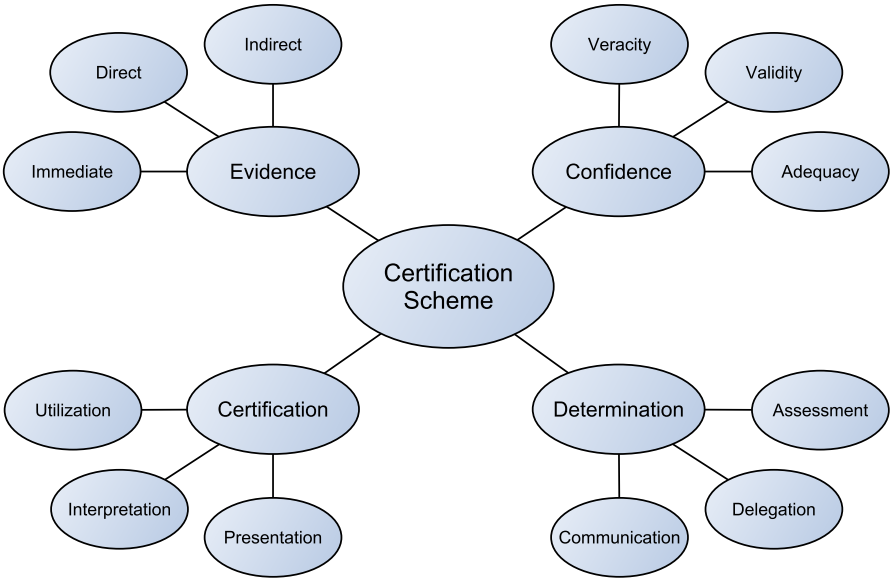

Figure 1: The framework

accommodate all approaches to certification, from current in-use standards based approaches to our own vision for software certification. In fact, there is no reason that the framework could not be applied to *any* setting in which evaluation takes place.

The framework decomposes any certification scheme into four aspects, indicated by the italicized words in the simplified approach above: *evidence*, *confidence*, *determination* and *certification*. The remainder of this section is devoted to discussing each of the aspects in detail. For each, we give its philosophical basis, explain what it is meant to include and to exclude, give examples of the kinds of things that it indicates, and provide a (provisional) further decomposition into useful sub-categories. Figure 1 presents the decomposition in graphical form.

3.1 Evidence

Evidence embodies the *empirical* part of the certification effort. It consists of the *things under consideration*: the real-world objects and documents that form the informational *foundation* of evaluation. Examples of evidence specific to the software setting include source code, requirements, specifications, machine executables, models, test results, proofs of correctness, real-world trial results, etc. Evidence can also include things like personnel qualifications/certifications and documented adherence to development processes.

When we discuss evidence in isolation from the other aspects, we are discussing how to identify, observe, measure, classify and organize items of interest. We are *not* talking about their trustworthiness, accuracy, relevance, adequacy, or any other such interpretation: these issues are epistemic and therefore fall into the confidence aspect. We are also not talking about the actual performance of any observations or measurements, as this is a pragmatic consideration which is

a part of determination. Finally, we do not treat certification artifacts—that is, items and documentation that are created during the certification activity—as evidence within the framework; from the "internal" point of view of certification, these are pragmatic side-effects and so we see them as part of the determination aspect (specifically, communication). (However, if we were later evaluating the certification activity itself, then these would certainly be seen as evidence.) Issues that pertain directly to evidence are reproducibility, traceability and identifiability.

We propose that evidence can be broken into subcategories based on how far removed it is from the product that is being certified: these subcategories consist of *immediate*, *direct* and *indirect* evidence. A helpful analogy to motivate the categorization comes from the legal setting: immediate evidence corresponds roughly to the defendant, direct evidence to material evidence, and indirect to circumstantial evidence.

1. *Immediate evidence*: evidence which is itself being evaluated; the parts of the candidate artifact: machine executables, source code, specifications, requirements documents.

2. *Direct evidence*: evidence which presents properties of the candidate; things that are directly about the immediate evidence: test results, proofs of correctness, static analysis results, hazards analyses, real-world trials, model checking results.

3. *Indirect evidence*: evidence which describes the circumstances relevant to the creation of the candidate; information about the development of the artifact: development processes, personnel qualifications, tool qualifications, content management systems.

In this context, our thesis can be restated more exactly: immediate evidence should include more than just the machine code; as a result, direct evidence must include much more than formal proofs. In fact, if we look carefully at Shao's approach from the perspective of our evidence categorization, our point makes itself. For besides the machine executable there is also the dependability claim (specification), which falls into the immediate evidence category and is therefore itself subject to evaluation. But meaningful examination of the dependability claim can only be carried out in the larger (non-mathematical) context of the system, where formal proof simply cannot be carried out. We will continue the discussion of where and how verification fits into our framework in §4.1.

A point which we have previously made is that current standards tend to overemphasize indirect evidence. Our papers [MW08, WML10] present the view that direct evidence should be the primary focus when certifying any system. We believe that the reason this is not the case is related to the third challenge: we lack *metrics* that can serve as the direct evidence we need. However, we do not make the claim that indirect evidence should be ignored altogether, as it can definitely be of use in building our confidence in a system. We advocate the middle ground: consider all the available evidence; incorporate metrics when they are or become available; allow for progressive refinement of the incorporation and treatment of evidence.

3.2 Confidence

Confidence is the *epistemic* aspect of certification. It is about our knowledge and judgements, how we value and reason about the evidence, consider and apply criteria, weigh the importance

of pieces of information, and combine pieces of information. Its focus is the *rationale*, ranging from qualitative and imprecise to quantitative and exact, about the evidence that is presented. As such, confidence represents the "measuring stick" of the certification effort, and embodies the reasoning behind the evaluation of the candidate product.

The confidence aspect of certification separates out the pure reasoning from the rest of the considerations. As such, issues relating to the expression and communication of that reasoning are outside of the scope of confidence; specifically, they fall under the communication component of the determination aspect. Examples of confidence specific considerations are: arguments, professional judgements, assumptions, uncertainty, weighting, inference (deductive or inductive), criteria, probability, trust and trustworthiness, soundness, relevance, tolerance.

We see confidence as divided into 3 sub-areas: *veracity*, *validity* and *adequacy*.

1. *Veracity*: about the *sources* of our knowledge; deals with the level of trustworthiness or accuracy of pieces of evidence: measurement tolerance, precision, trust, reliability, reputability.

2. *Validity*: about the *interpretation* of the evidence; encompasses the inferences and logical steps that we make: soundness, relevance, consistency, justifiability, defensibility, reasonableness.

3. *Adequacy*: about the *sufficiency* of our knowledge; focuses on what is required of a system and the presented evidence to achieve certification: criteria satisfaction, completeness, comprehensiveness, sufficiency, conclusiveness, acceptability.

Confidence is a very difficult aspect of certification. There are many fundamentally different kinds of knowledge as outlined above; developing a systematic treatment of confidence in which these can all be incorporated is hard to envision, let alone accomplish. The problem of combining pieces of information in a consistent and meaningful way has no clear solution. Existing logics and frameworks of uncertainty, though interesting and relevant, universally do not allow us to treat some of the less mathematical sources of knowledge that we must deal with in an engineering setting.

Our current research is centred on using *assurance cases* as the generic setting for representing confidence issues (see §4.2). We have also explored formally modelling confidence in terms of a complete upper semilattice, with joins representing combinations of evidence/confidence. This general setting allows for the incorporation of other confidence metrics when we have them (along with their respective operations), while also allowing for non-logical qualitative forms of confidence such as *judgements* by professional engineers. Though there is still a great deal of exploration and development to be done in this direction, we believe that our work shows promise as a generic backdrop for reasoning about engineering confidence.

Besides the classification of confidence presented above, we have also developed a hierarchy of types of *support* provided for claims. From weakest to strongest, support can be stratified into belief (unsupported), judgement (statement by some (reputable) party), rationalization (provision of argument or logic behind claim), substantiation (provision of evidence supporting the claim), and demonstration (showing a model or example of what is claimed). We do not delve more deeply into this here, but only mention it to help support our statement that engineering confidence is a tricky beast with many facets that make it difficult to comprehend and manage.

3.3 Determination

Determination encompasses all of the *pragmatic* facets of the certification process (as distinguished from the *development process* which we see as indirect evidence). It denotes all of the "feet-on-the-ground" aspects of certification as an activity: who does what, how measurement and evaluation takes place, how things are recorded and communicated, what is produced.

Having determination as a separate aspect allows us to focus on and talk about the real-world issues involved in certification in isolation, thus drawing a clear line between theory and practice. We believe that in the engineering setting such a line can be of great value. Engineering is ultimately about making real products, and since our framework is intended to present a comprehensive decomposition of certification theory and practice, determination is a very important aspect. Examples of some of the topics that determination encompasses are: examination, assessment, spot-checking, inspection, professionals and professionalization, roles and responsibilities, certificate-granting authorities, certificate creation and management, dialogue between the developer and certifier, and certification maintenance and expiration.

Our tentative decomposition of determination breaks it into three parts: *assessment*, *delegation* and *communication*. We make no claim that this breakdown is comprehensive but it serves to help illustrate some of the various issues that come into play in practice.

1. *Assessment*: about how evidence is examined, verified and evaluated, and how rationale is developed and scrutinized: inspection, re-running tests, checking sources, verifying authenticity, appraising rationale, identifying problems, auditing processes.

2. *Delegation*: about the human/social aspects of certification; the roles, responsibilities and functions of the parties involved: regulators, certifiers, independent evaluators, domain experts, professionals (engineers and otherwise).

3. *Communication*: about expressing, relating, recording, tracking and archiving certification activities: document formats, mathematical knowledge management, media, security and encryption, clearance levels ("social information hiding").

To those attending this conference, determination is perhaps the least interesting aspect of certification; after all, practical considerations are often seen as uninteresting from a theoretical point of view. Perhaps this is the reason that Shao's scheme eschews practical questions altogether. Our view, which has developed over time both through academic research and through industry collaboration, is perhaps best put by Yogi Berra: "In theory there is no difference between theory and practice, but in practice there is." It has been our experience that theoretically reasonable ideas are often not useful to engineers because they are quite simply not practicable, for one reason or another; on the other hand, tools that are theoretically unattractive sometimes become quite popular in practice. This disconnect alone is, to us, reason enough to devote an entire branch of our framework to pragmatics.

Even the simple question of how to manage certificates for software (reminder: we do not subscribe to the view that software can be certified in isolation) is difficult: there has been an entire workshop devoted to the problem [DFHJ05]. Interestingly, some of the papers in said workshop were by researchers who we would classify as verificationists, but even from their perspective that certificates are just proofs, managing certificates is quite complicated. Shao

himself says that a general framework in which we can manage and connect "proof certificates" would be a (so-far nonexistent) rich second-order logic. Once we widen our view to encompass other pragmatic considerations, it becomes clear that we have a long way to go to understand how determination fits into the overall picture of certification.

3.4 Certification

A *certification* is the result of a successful evaluation activity. It is a "marker" or designator that results from the bid by a developer to certify or have certified their candidate product. We see it as a *signifier* of that success; as such, certification in our framework embodies the *semiotic* aspect of the certification scheme.

Before we explain the certification aspect further, it is important to distinguish the specific usage of the word certification that is being adopted here. Note that certification can be given both a "means" and "ends" reading—the word can be taken to mean both the process of certifying on the one hand, and the resulting achievement on the other. The question might be asked as to why we do not just use the term "certificate" instead to get rid of this problem. The reason is that we see certificates as concrete entities, whereas certifications are abstract and can be represented by many different kinds of certificate. A useful analogy might be the distinction between a degree and a printed diploma: it is entirely possible to obtain multiple diplomas (one for home, one for the office, etc.) for a single degree; more importantly, if diplomas are lost or destroyed they can simply be reprinted, based on one's academic records. Such academic records are in fact just another representation of the degree, as we will explain

Examples of topics that are part of the certification aspect are: certificates, records of certification, standards, legal implications, contexts of applicability, limitations on use, and expiration/reevaluation indications.

Following the traditional decomposition of semiotics, we break certification into three parts: syntactics, semantics and pragmatics. Note that we do not adhere seriously to the semiotic interpretation as presented in the literature, but rather adopt it as a helpful guide for how to treat an entity whose primary purpose is to signify. To actualize this, we choose to call the syntactic part *presentation*, the semantic part *interpretation* and the pragmatic part *utilization*.

1. *Presentation*: how a certification is presented, realized or actualized; the real-world representation of the certification: certificate, database entry, proofs of certification, naming/numbering (e.g. IEC 61508).

2. *Interpretation*: about the meaning of a certification to various parties; the denotations and connotations of certification: records of certification, engineering log books, list of products certified, reputability of a certification.

3. *Utilization*: the implications, restrictions and limitations on the use of a certification or standard; contexts of use and applicability: legal ramifications, liabilities, issues relating to international use/interpretation.

We see the interpretation of a certification as being loosely divided into its *vindicative* and *indicative* reading. The vindicative interpretation consists of (a presentation of) all of the evidence, evaluation activities, involved parties, rationale, etc. that took place while obtaining the

certification—the denotations of the certification, representing all of the actual facts that stand behind it. The indicative reading is what the certification entails to the "outside world"—the connotations of the certification, including any interpretation that goes beyond the established facts of the certification activities involved. For example, an ISO 9001 certification vindicatively means that the process and documentation put in place by a company has been evaluated and assessed as conforming to the respective standard; indicatively, it can be interpreted as showing the maturity of an organization, its trustworthiness, etc. If it were not for the indicative aspects, why else would businesses hang the ISO 9001 banner outside their buildings?

There are many other interesting questions to discuss about certifications and what they mean. The semiotic view allows for what we believe is a clean decomposition while at the same time affording a novel interpretation of what a certification really *is*. The exploration of the semiotic approach to certifications is still underway. Some other interesting questions are these: How do we treat certifications of people, tools, other systems, etc. as *evidence* within another certification scheme? What is the right way to incorporate the vindicative and indicative aspects in this setting? What is the exact difference between indicative interpretation and utilization? These topics are the subject of active research.

4 Applying the Framework

To help understand the framework and put it in context, we now turn our attention to how it handles and organizes things, particularly verification and assurance cases. At the same time, we will exploit the framework to help us understand and put into perspective those notions. To achieve this, we will look at each and break it down into aspects by placing it into the framework, and then discuss the interesting questions, challenges and issues that arise.

4.1 Incorporating verification

Generally speaking, verification consists of proofs that a program satisfies its specifications. In Shao's view, the specification of interest is a dependability claim, and the program is taken to be a machine executable. So where does everything fit within the framework?

First, we classify the evidence that verification is concerned with.

- Code, specification: *immediate evidence*. These are all a part of the artifact that is being certified (according to the arguments we put forward in §2.1), and are therefore treated as immediate evidence in the framework.
- Proof: *direct evidence*. Proof is directly about both the code and specification which are immediate evidence, and so proof is classified as direct evidence.

Elaborating on the points made in §3.1, when we place the evidence that is considered from a purely verificationist stance into our framework, it can immediately be seen to be incomplete. First of all, with specifications as a part of the immediate evidence, direct and indirect evidence can only be comprehensive if it covers the specification as well. Secondly, it is more than likely that the development effort should produce many more than these items of evidence; surely it

would be prudent, from an engineering point of view, to consider them *all* in order to paint a complete picture of the candidate. After all, software, being in itself abstract, can only be a *part* of an engineered artifact. To understand its role in an overall system requires that we look at the whole picture, in which verification plays but a limited part.

In order to fully understand this issue, let us now address the confidence-related (epistemic) aspects of verification. It is from this point of view that the incompleteness of verification-specific evidence can fully be appreciated.

1. Veracity. The veracity issues that are directly related to verification are straightforward: how do we ensure that the proof indeed matches the given specification and code, and that it is in fact *correct*? Thus, the pertinent issues are *traceability* and *proof checking*. A possible source of evidence to support the first is checksumming; for the second, using qualified (certified) proof-checking tools; indirect evidence pertaining to the personnel involved can potentially be used to support both.

2. Validity. Here lies the real strength of verification. Proof presents, in a sense, the highest form of validity: the unconditional *truth* of a statement. In the presence of such a proof, the code is perfectly tied to the specification. But when we question the validity of the specifications themselves, the question is far from answered. Are the specifications themselves valid, i.e., are they relevant and reasonable as a representation of the system? This is an epistemic question, not subject to proof but to less formal kinds of reasoning like argumentation.

3. Adequacy. Formal proof is more than adequate to show that specifications are met by the code, but this is *not* the notion of adequacy that is being discussed here. What adequacy addresses is whether or not the evidence and supporting rationale is sufficient to back up the claim that is being made about the software/system. *Is proof enough?* We say definitely not. Let us look at Shao's dependability claim: what should we have in hand to make a statement about dependability? Dependability, in an engineering setting, is a *systemic* property that cannot be reduced to the reliability of individual components like software. The issue of adequacy, in our view, is the final nail in the coffin for the verificationists. There are simply (and obviously) too many other questions that need to be addressed before an engineering artifact can seriously be called dependable.

We now turn to the practical issues relating to verification, that is, the aspect of determination. Proofs can be mechanically checked—this is a real benefit to the evaluator (keeping in mind that we must have established that the proof-checker itself is reliable). However, the task of evaluating the specifications themselves cannot be automated. If the evaluator comes to the conclusion that the specification must be changed, then the proof is essentially void and the job must be redone. This is (in general) an expensive proposition, and one that should not be taken lightly: in real-world engineering, resources and cost are of paramount importance. The verificationist approach requires significant commitment to the specifications, but from a holistic, system-wide view, this may well be unwise as during the course of the certification process the specification might change. Although we agree that for highly safety-critical parts of the system such a commitment is called for, this is not always the case. *When is proof worth the effort?*

Finally, we look at the question of the certification. In the general setting, verification would be a part of the evidence that was considered and would fall into the vindicative aspect of the interpretation of a particular certification. This is how we see things. Shao, on the other hand, sees the proof as the certificate. We believe that this is fundamentally untenable because, in practice, certifications mean much more. How can a proof reflect, for example, legal implications? There are large pieces missing from the vision of certification presented by Shao. Questions about how a certification is interpreted are also very important—a proof does not reflect the evaluation effort that checks that same proof for correctness, or that ascertains that the specification that was proved against was adequate, as discussed above.

All in all, our point of view is simple and in line with [FL07]: verification is a *powerful* tool for certification, but it is one that has a significant practical *cost* and must be carefully positioned when it is placed in the setting of certification.

4.2 Using assurance cases

An assurance case is a document (or a collection of documents) that presents, in a systematic, structured form, the rationale or *argument* that some *claim*—usually a claim about a property, say safety, of a system. An assurance case that makes that specific claim is commonly called a *safety* assurance case or just a safety case; it is the most common kind of assurance case. They have been widely employed, particularly in Europe, as an important component of the engineering methodology used for safety-critical systems. Examples of organizations that mandate the use of (safety) assurance cases are the UK Office of Rail Regulation [UK 00], the UK Ministry of Defence [UK 07], and Eurocontrol [Eur01].

Assurance cases provide a setting in which to *document, structure, archive, reuse, scrutinize and communicate* rationale [KW11]. Historically, (safety) assurance cases have been presented, essentially, as a normal (textual) collection of documents. More recent work, however, has centred around representing rationale/argumentation *graphically* using *Goal Structuring Notation* (GSN). This development is due in large part to work of Tim Kelly and colleagues at the University of York; Kelly's dissertation [Kel99] contains the first fully elaborated goal-structured notational system for assurance cases. The main benefit of a graphical representation of rationale is that the structure of the rationale is reflected in the structure of the *documentation* of the rationale, thereby facilitating its comprehension and navigation; however, such graphical notation can be problematic because of the amount of space it often requires. GSN also suffers from a complete lack of any rigorous semantics, although we are actively developing ways to incorporate them when we can (see below).

How do assurance cases fit into our framework? Our perspective is this: they should *not* be treated as a simple piece of evidence that is provided by the developer, but rather as a part of the communication aspect of determination. More exactly, assurance cases provide a setting in which the developer and the certifier can engage in and record the dialogue about the candidate's safety, efficacy, fitness for purpose, etc. This point of view is perhaps surprising, so it merits a bit of explanation.

In the traditional view, assurance cases would fit into the framework as immediate evidence, being but one of the components of the artifact under consideration (albeit quite an important one). The developer would record their rationale for the design decisions, provided evidence,

etc. in the submitted assurance case. The certifier could then scrutinize the other evidence provided by the developer in the context of the assurance case, that is, the context of the developer's rationale about their product. Though we see the traditional approach as valuable, it also unnecessarily restricts the role of the assurance case in the certification setting and misses a tremendous opportunity for supporting the certification process itself. A new approach, advocated by John Knight [KW11], positions the assurance case as a *living document* which plays a part during both the development and certification of a system, thus facilitating and recording the dialogue just mentioned.

The primary purpose of the assurance case in our setting is as a tool for expressing, developing and communicating *confidence*. Because GSN allows us to express generic arguments (rationale), it is well-suited to handling the qualitative, non-mathematical aspects of confidence that are involved in certification. On the other hand, we are currently developing a method for incorporating mathematical, calculational aspects of confidence into assurance cases when we have them; for example, embedding deductive logical arguments into GSN without discarding their semantics, and bringing in probabilistic forms of evidence with corresponding probability operations encoded as GSN primitives.

5 Conclusion

In this paper, we have presented a new framework for certification, decomposing it into four fundamental philosophical aspects: evidence, confidence, determination and certification. We used our framework to position and analyse verification in the context of certification. We have argued that verification is an important and powerful piece of *evidence* in a certification scheme, but that it is not *central* to such a scheme. In particular, we have examined Shao's [Sha10] approach to software certification and presented our view that it is essentially and fundamentally flawed.

To complete our argument, let us return to Shao's three challenges faced by software certification. We argue that our framework provides a setting in which we can *gradually* attack the challenges instead of trying to solve them all at once, which we believe is practically impossible.

1. Software is everywhere. How do we manage the huge number of domains we have to interact with?

   Shao's solution, to ignore the domains and certify the software in isolation, we have already addressed: we believe that this position is untenable. We must respect practice if our approach to certification is to be taken seriously. To advocate verification as the solution to all problems, while ignoring existing—though admittedly flawed—standards and certification approaches is harmful to the already strained relationship between theory and practice in our field. There are sound practical reasons for the evidence that is being considered, as we write this, in real-world certification activities. We believe that it is important to consider, however inexactly, *all of the available evidence*.

   Considering, as we do, software as but one aspect of an engineering system, we need to tie in to existing certification efforts and to continue to develop metrics and confidence measures for software. Our certification framework is designed to allow for new measure-

ments and analyses to be incorporated as they become available, but also to make use of more qualitative evidence like process adherence and personnel qualifications, etc., for the time being.

2. Software is complex. How can certification scale?

   Rather than throwing up our hands in despair, we firmly believe that the problem can be confronted gradually. Not all software systems need to be certified with the same level of rigor; not all development activities must be treated in the same way. Highly critical systems can be more strictly regulated and we can limit the complexity of the software there deployed, just as Ontario Hydro did with their nuclear shutdown system [WL03]. For systems at a lower level of criticality, we can relax the requirements somewhat and employ less severe, but still meaningful, forms of evidence and confidence measures. From this stance we can progressively tighten the certification scheme as appropriate methods become available, while still maintaining an approach that can essentially be applied to any system containing software.

3. We lack metrics for software dependability. How can metrics be discovered and developed, and how can we incorporate them when they exist?

   Shao's idea, that is to view the dependability claim as a kind of metric, seems entirely to miss the point of a metric as it does not present something that is measurable. However, we do strongly agree that metrics are scarce and that, even when they do exist, they do not provide particularly good information (e.g., lines of code as a measurement of complexity). Nevertheless, this will not necessarily be the case forever, and until more mathematical metrics become available we can still make use of engineering judgement, best practices and other qualitative forms of evidence. This problem just reflects the nature of software engineering: as a young and relatively underdeveloped branch of engineering, it is what Vincenti calls *radical engineering* [Vin90], but as time passes it will become normal, just as other engineering disciplines have. As this process takes place, our framework provides a setting to support certification through the transition. This is the best we can do at present, but we believe that it is the only responsible, professional approach that engineers can adopt.

Acknowledgements: The authors would like to acknowledge various members of McSCert for valuable discussions and helpful pointers relating to the certification framework; in particular, weekly discussions with Gord Uszkay and Valentin Cassano about tools and techniques for expressing and managing confidence have been an invaluable source of insight. We would also like to thank Paul Joannou, Zarrin Langari, Ali Taleghani for their helpful comments on an earlier version of this paper, and the members of the Software Certification Consortium for their feedback on a version of the framework that was presented at their May 2011 meeting.

Bibliography

[DFHJ05] E. Denney, B. Fischer, D. Hutter, M. Jones. Software certificate management (SoftCeMent'05). In *Proc. of the 20th IEEE/ACM international Conference on Automated software engineering*. ASE '05, pp. 463–463. ACM, New York, NY, USA, 2005.

[Eur01] Eurocontrol. *The EUR RVSM Pre-Implementation Safety Case, Ver. 2.0 (RVSM 691)*. Aug. 2001. http://www.eur-rvsm.com/safety.htm#precase.

[FL07] J. Fitzgerald, P. Larsen. Balancing insight and effort: the industrial uptake of Formal methods. In *Formal methods and hybrid real-time systems*. Volume 4700, pp. 237–254. 2007.

[HHL+09] J. Hatcliff, M. Heimdahl, M. Lawford, T. Maibaum, A. Wassyng, F. Wurden. A Software Certification Consortium and its Top 9 Hurdles. *Electron. Notes Theor. Comput. Sci.* 238:11–17, September 2009.

[Kel99] T. P. Kelly. *Arguing safety – a systematic approach to managing safety cases*. PhD thesis, Department of Computer Science, University of York, 1999.

[KW11] J. Knight, K. Wasson. Safety Assurance Cases for FDA 510(k) Submissions. Waltham, MA, USA, June 2011. Unpublished course materials.

[MW08] T. Maibaum, A. Wassyng. A Product-Focused Approach to Software Certification. *Computer* 41:91–93, 2008.

[Sha10] Z. Shao. Certified software. *Commun. ACM* 53:56–66, December 2010.

[Tur49] A. M. Turing. Checking a Large Routine. In Anonymous (ed.), *Report on a Conference on High Speed Automatic Computation, June 1949*. Pp. 67–69. University Mathematical Laboratory, Cambridge University, Cambridge, UK, 1949.

[UK 00] UK Office of Rail Regulation. *The Railways (Safety Case) Regulations 2000*. Oct. 2000. http://www.legislation.gov.uk/uksi/2000/2688/contents/made.

[UK 07] UK Ministry of Defence. *Safety Management Requirements for Defence Systems*. 2007. Def Stan 00-56, Issue 4.

[Vin90] W. G. Vincenti. *What engineers know and how they know it: Analytical studies from aeronautical history*. Johns Hopkins University Press, Baltimore, 1990.

[WL03] A. Wassyng, M. Lawford. Lessons Learned from a Successful Implementation of Formal Methods in an Industrial Project. In Araki et al. (eds.), *FME 2003: International Symposium of Formal Methods Europe Proceedings*. Lecture Notes in Computer Science 2805, pp. 133–153. Springer-Verlag, Aug. 2003.

[WML10] A. Wassyng, T. Maibaum, M. Lawford. On Software Certification: We Need Product-Focused Approaches. In Choppy and Sokolsky (eds.), *Monterey Workshop 2008*. LNCS 6028, pp. 250–274. Springer, 2010.