NASA/TM–2015–218804

Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software

Patrick J. Graydon and C. Michael Holloway
Langley Research Center, Hampton, VA

September 2015

NASA/TM–2015–218804

Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software

Patrick J. Graydon and C. Michael Holloway
Langley Research Center, Hampton, VA

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

September 2015

Acknowledgments

We thank Rob Ashmore, John Knight, John McDermid, and Drew Rae for their helpful suggestions regarding this paper.

We thank Rob Ashmore, Dewi Daniels, Mario Fusani, Ibrahim Habli, Chris Johnson, Giuseppe Lami, Bev Littlewood, Drew Rae, John Rushby, Lorenzo Strigini, and Virginie Wiels for submitting position papers to AESSCS 2014.

We thank John McDermid and Pippa Moore for their keynote addresses.

We thank Sami Alajrami, James Armstrong, Rob Ashmore, Fatemeh Ayatolahi, Dewi Daniels, Jean–Charles Fabre, Camille Fayollas, Jane Fenn, Mario Fusani, Omar Jaradat, Nathan Kennedy, John Knight, John McDermid, Pippa Moore, Nadir Murru, Drew Rae, John Rushby, Miruna Stoicescu, Lorenzo Strigini, Virginie Wiels, and Wen Zeng for their participation in the workshop and their contribution to the discussion.

We thank the AESSCS 2014 program committee, Sushil Birla, Darren Cofer, Mike DeWalt, John Knight, Alan Wassyng, and Virginie Wiels, for their efforts.

We thank EDCC 2014 general chair Alexander Romanovsky, workshop chair Andrea Bondavalli, and the other EDCC 2014 organizers and volunteers for their support of AESSCS 2014.

Some participants' attendance of AESSCS 2014 was funded by the Swedish Foundation for Strategic Research (SSF) as part of the SYNOPSIS project.

The use of trademarks or names of manufacturers in this report is for accurate reporting and does not constitute an official endorsement, either expressed or implied, of such products or manufacturers by the National Aeronautics and Space Administration.

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Abstract

We need well-founded means of determining whether software is fit for use in safety-critical applications. While software in industries such as aviation has an excellent safety record, the fact that software flaws have contributed to deaths illustrates the need for justifiably high confidence in software. It is often argued that software is fit for safety-critical use because it conforms to a standard for software in safety-critical systems. But little is known about whether such standards 'work.' Reliance upon a standard without knowing whether it works is an experiment; without collecting data to assess the standard, this experiment is unplanned. This paper reports on a workshop intended to explore how standards could practicably be assessed. Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software (AESSCS) was held on 13 May 2014 in conjunction with the European Dependable Computing Conference (EDCC). We summarize and elaborate on the workshop's discussion of the topic, including both the presented positions and the dialogue that ensued.

1 Introduction

We need well-founded means of determining whether software is fit for use in safety-critical applications. While software in industries such as aviation has an excellent safety record, the fact that software flaws have contributed to deaths and injuries (e.g., the Therac-25 accidents [1] and the in-flight upset of an Airbus A330-303 [2]) illustrates the importance of having *justifiably* high confidence in safety-critical software. Despite the dangers of relying on software having properties we can't guarantee, developers are building ever-more-complex safety-critical software, sometimes for good reasons. Today, it is often argued that software is fit for safety-critical use because it conforms to a standard for software in safety-critical systems such as RTCA DO-178C[1] [3] for avionics, ISO 26262 [6] for automotive[2], or IEC 61508 [7] for protection and control systems[3]. But little is known about whether such standards 'work.' (What it means for a standard to work is also less clear than it could be.) To rely upon a standard without knowing whether it will work is to conduct an experiment; to do so without carefully planning data collection to assess the standard in practice is to conduct an *unplanned experi-*

[1] RTCA DO-178C [3] is functionally identical to EUROCAE ED-12C [4]. The same is true of their predecessors, RTCA DO-178B [5] and EUROCAE ED-12B. For simplicity, we refer only to DO-178B and DO-178C in this paper.

[2] ISO 26262 "is intended to be applied to safety-related systems that include one or more electrical and/or electronic (E/E) systems and that are installed in series production passenger cars with a maximum gross vehicle mass up to 3,500 kg" [6].

[3] IEC 61508 "applies to all types of [electrical, electronic, and programmable electronic] safety-related systems, including protection systems and control systems" [7].

ment. This paper reports on a workshop intended to explore how the community might do better by fostering discussion about how standards could practicably be assessed. *Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software* (AESSCS) was held on 13 May 2014 in Newcastle upon Tyne, UK, as part of the European Dependable Computing Conference (EDCC). Papers from the workshop are available online [8–15]. In this paper, we summarize and elaborate on the workshop's discussion of the topic, including both the presented positions and the dialogue that these positions prompted on the day. The authors of this paper are reporting positions expressed in the submitted position papers and by participants in discussion at the workshop. Unless specifically indicated in the text, readers should not assume that any given person (or that person's employer) endorses a given position.

In Section 2, we present the motivation for the workshop. In Section 3, we present proposed definitions of what it means for a software safety standard to 'work.' In Section 4, we outline strategies for assessing standards. In Section 5, we define research questions related to the efficacy of standards for safety-critical software, illustrating the variety and number of such questions. In Section 6, we relate some of the studies proposed by participants and elaborate on how they might be conducted and address the research questions presented in earlier sections. In Section 7, we present observations raised by workshop participants. In Section 8, we discuss related work. Finally, we conclude in Section 9.

2 Motivation

Standards such as RTCA DO-178C [3], ISO 26262 [6], and IEC 61508 [7] present rules and guidance for developing and assessing software for use in safety-critical applications: developers conforming to them must meet a series of *assurance requirements*[4] related to development process, development artifacts (including inspection, review, analysis, and test results), and software. Assurance requirements are typically expressed at a high level that admits multiple means of conformance. For example, ISO 26262 requires developers to conduct unit testing to demonstrate that software exhibits specified functionality but no unintended behavior (amongst other things) [6, §9.4.3.c–d]. To achieve this, it *recommends* 'requirements-based test[ing],' 'interface test[ing],' 'fault injection test[ing],' and 'resource usage test[ing],' in some cases requiring developers to justify a decision *not* to use some of these techniques.

Most current standards for software for use in safety-critical applica-

[4] Assurance requirements—requirements that must be satisfied in order to claim conformance—are known by different names in different standards. For example, RTCA DO-178C refers to its assurance requirements as 'objectives' [3]. Other standards' assurance requirements are sometimes known as 'clauses.'

tions are built around variations of a *recipe* for providing safety assurance for such software. The recipe—implicit in the standards' assurance requirements and recommendations—calls for the provision of several pieces of evidence that are each, by themselves, not a direct assessment of software's comprehensive contribution to system safety. Developers separately assess the highest-level software safety requirements, lower-level requirements, the software architecture and design, the source code, execution time, and whether the object code satisfies its requirements (typically by means of testing at the unit, integration, software, and system levels).

The standards' recipe(s) purportedly represent expert consensus on good development practice. But experts (and their consensus) might be wrong, a whole might be less than the sum of its parts, and there is little or no direct evidence that the recipe(s) *work*. It is critical that we determine whether or not the recipe(s) work, and, if so, how and why: we are relying on the recipes to develop and assess software for use in safety-critical applications. To the degree that we do not know that these recipes work, our trust in software might be misplaced. Moreover, even if what we are doing is effective—and it seems to be—if we do not know *how* and *why* the recipe works, the slightest change to practice risks undermining whatever effect standards now have [9].

Some argue that history shows the standard recipes work [16]. That is, because we have experienced few software-caused fatalities, the standards must either cause software to be fit for use or confirm that it is. But this line of reasoning is weak for several reasons:

- Many applications do not offer enough exposure to make history statistically significant.
- Correlation is not causation [17]: observing standards conformance and safe operation together cannot show that one causes the other. Safe operation might plausibly result from a third factor such as the care developers lavish on safety-critical systems [18].
- Measuring the correlation is difficult. Multiple versions of multiple standards have been used, and anecdotes about lenient assessors suggest differences in how standards are interpreted and applied.
- Standards might work better for the systems and applications of the past than with some new systems applications (e.g., those featuring more complex microprocessors, more integrated and networked platforms, or more autonomy).

A well-planned, carefully controlled, long-term study might be useful, but none has been done [19].

Standards reflect the consensus of the committees that wrote them. The rationales for standards' recipes are rarely published. Even if they were, the scientific literature offers little evidence to support the logic they would likely contain. For example, there is little evidence that testing to Modified Condition/Decision Coverage[5] confirms correct software. Experiments show that code inspections reveal many interesting defects but miss some; no one knows how many [21]. Some coding standards limit code complexity [Section 5.4.7] [6], but experts are divided on whether lower complexity causes lower dangerous failure rates and the experimental evidence is limited and conflicting [22].

It is tempting to say we should not build software without knowing how to establish that it is fit for use. However, we continue to do so, sometimes for good reasons. While it is easier to reason about the safety of simple hardware, software can implement more complex behavior. For example, simple hardware cannot automatically brake a car to avoid hitting a pedestrian. The inability to guarantee that automatic braking will always work will not keep people from driving cars. Arguably it should not: fitting imperfect automatic braking systems might result in fewer deaths and injuries than fitting no automatic braking system. We do not seek to stop innovation, but rather to increase both safety and our confidence in safety assessments.

We are relying on standards' recipes in matters where lives are at stake. Even when developers use an argument-based standard rather than a so-called prescriptive standard, they tend to follow the same recipe [23, 24]. The people who develop and use standards have the best of intentions. Nevertheless, there is little evidence to either support or rebut the claims that following a standard's recipe ensures or confirms that software is fit for safety-critical use. We urgently need to study both whether the recipe works and why it does or does not work.

3 What it Means For a Standard to 'Work'

It was the workshop organizers' sincere hope that participants would bring evidence to challenge the key premises of the workshop, namely that (a) it is crucial that we understand how and why standards work and that (b) we don't know as much about this as we ought to. But all in attendance agreed with these premises and concluded that more research on this topic is urgently needed. However, discussion revealed several different definitions of what it might mean for a standard for software in safety critical applications to 'work.'

Ideally, each standard should define what it means for the standard to 'work.' This is crucial information; it is difficult to see how standards committee members could decide whether a draft standard is acceptable without knowing what the standard is meant to do. Some standards do include statements about the standard's aims or contributions. But these

[5] Modified Condition/Decision Coverage (MC/DC) is a structural test coverage metric [20]. RTCA DO-178C (and its predecessor, RTCA DO-178B) require software testing of the most critical software (i.e., level A software) to achieve MC/DC [3, 5].

are not generally testable hypotheses about the effects on safety. Some are too vague, others are belied by later text of the standard, and others sidestep important issues. For example, the introduction to ISO 26262 promises that it delivers 'guidance to help avoid [risks due to failing electrical or electronic components in automobiles] by providing appropriate requirements and processes' [25]. But what does it mean to 'help avoid' risk? The statement does not promise that risk will be reduced to or by any specific, testable amount or that the result will satisfy any of the usual theories of residual risk acceptability [26]. It is perhaps unfair to expect that it would: ISO 26262 is an industry consensus document aimed at achieving an approach that will work across a diverse supply chain. Nevertheless, if we are to determine whether a standard works, we need a better definition of what it means to work. Must following the standard always (or with a specified likelihood) result in software that is fault free or meets a reliability target? Must it do these things in a cost-effective manner?

Participants in the workshop identified both (a) a crucial distinction between the goals of software *correctness* and software's contribution to system safety and (b) several distinct definitions of what it might mean for a standard to 'work' in achieving one of those goals. The definitions can be broadly organized into four categories: (1) bounding uncertainty in a claim about software behavior or contributions to system safety, (2) addressing the risk from the most likely and/or consequential mistakes, (3) promoting developer competence, and (4) giving certifiers a tool to improve safety. In this section, we will elaborate on the distinction of safety versus correctness, on each of these potential definitions, and on the broader issue of cost-effectiveness.

3.1 Safety Versus Correctness

Some standards for software in safety critical applications are focused on correctness rather than safety. That is, the majority of their assurance requirements are more clearly about whether the software behaves as specified (and *only* as specified) rather than how the software's behavior affects system safety. For example, RTCA DO-178C [3] assumes the existence of software requirements derived from a system-level safety analysis (e.g., in accordance with SAE ARP4754A [27]). The standard requires developers to show that software high-level requirements refine the given requirements, including safety requirements, but does not distinguish between software safety requirements and non-safety-related software requirements. It specifies a mechanism for specifying a software development assurance level (and ultimately a concrete development and assurance plan) from the consequences of software failure, but does not discuss the human-factors impacts of software human interface design choices[6]. Most of the standard's assurance requirements are related to demonstrating that the delivered software meets the given requirements. Developers demonstrate this by providing evidence for a chain of refinement from the given requirements through low-level requirements and source code to the delivered object code.

Many other standards include parts describing both system-level safety analysis and software-level safety activities. For example, ISO 26262-3 defines a mandatory hazard and risk analysis process, while ISO 26262-6 gives software verification and validation requirements [6, 28]. But the focus of ISO 26262 is limited to the safety impact of the failure of individual electrical and electronic systems. Like RTCA DO-178C, ISO 26262-6 contains many assurance requirements related to software correctness. But it contains no requirements related to, for example, designing a car as a whole in order to achieve adequate safety. The standard's assurance requirements would apply to the design and implementation of an electronic stability control system *if* the vehicle manufacturer fits one, but does not offer any guidance on *whether* such a system should be fitted.

Correctness is certainly helpful in achieving safety, but correctness is not a sufficient condition for safety. Safety is also impacted by decisions about what to build as well as how well systems are built. Perhaps more so: there is some evidence to suggest requirements defects are a bigger problem than implementation defects [29, 30]. While we can name a few well-published examples of accidents ascribed to software implementation mistakes, these appear to be few and far between. Problems related to how computer systems automate tasks (as opposed to how correctly the software implements the automation as designed) appear, at first glance, to be at least as big a problem in the aviation domain [31]. New pilots are not necessarily taught to do things the old-fashioned manual way, and thus may not be able to cope when automation fails. This problem will only grow as developers face pressure to automate more functions (e.g., to reduce crew workload, add capability, or increase fuel economy), and trends towards assisted and even autonomous driving in the automotive domain[7] suggest that automation accidents on the road are not far in the future.

[6] Even if software engineers in some domains are responsible to some extent for user interface design, DO-178C relegates consideration of the safety impact of human interfaces to system safety engineers following separate processes. ARP4754A requires the specification of 'operational requirements' that 'define the interfaces between the flight crew and each functional system' [27, §5.3.1.2.2]. If, while designing software to satisfy these requirements, software engineers see the need for a requirement that is 'not directly traceable to higher-level requirements' or 'specify behavior beyond that specified by the system requirements,' they must submit 'derived requirements' to the safety engineers for consideration [3, §5.1.1.b, §5.2.1.b, glossary].

[7] For example, Nissan plans to demonstrate prototype remotely-supervised autonomous taxis within the next two years [32]. Regulators are aware of the difficulties that such automation might pose and are working to draft suitable policies [33].

Achieving adequate system safety, as opposed to only achieving software that performs as specified, requires making appropriate decisions about what software should do. Developers and regulators must consider many aspects of what makes software design and implementation fit for use, including the effects of software design choices on human performance and security implications.

In some cases, this is an issue of who takes ownership for the correctness of software requirements. For example, suppose that embedded aircraft software meets its requirements but operates in a way that creates additional challenges for aircraft maintainers. The software might be correct, yet lead to a higher risk of maintenance error—and thus of an accident—than an alternative design.

In other cases, this is an issue about how computer-based systems are designed. For example, it might lower software development cost to implement an avionics function in a system unit that is thematically unrelated: the target unit might have surplus processing power, or better access to the key data needed to perform the function in question. But when the automation fails, human flight crew members might struggle to understand the error messages they are receiving about a unit that has no obvious connection with the failure symptoms they are observing.

Building automobile radios with speed-sensitive volume controls increases passenger comfort and might decrease driver distraction by eliminating a reason to turn the radio down at traffic lights and up again at speed. But such integration necessitates a data connection between the entertainment system and the relevant engine control unit(s). That link is a potential vector for deliberate attack: the complex and seemingly-unimportant entertainment system might be an attacker's means of gaining access to computers that control lighting, braking, and other safety-relevant functions [34]. While most safety standards address security concerns only indirectly (if at all)[8], the potential for deliberate attack to compromise safety cannot be ignored.

It might be the case that, despite any new dangers that it introduces, increased reliance on computer-based automation will reduce deaths and injuries in some applications. (Autonomous automobiles might prove to

[8] For example, RTCA DO-178C does not mention security except to say that system requirements allocated to software might include security requirements [3, §2.1]. ISO 26262 makes no mention of security whatsoever [6,25,28,35–41]. IEC 61508:2010 does mention security, but stops short of directing engineers to achieve an appropriate balance of safety and security (let alone identifying a means for doing so) [42]. The standard calls for a security threats analysis when safety analysts performing hazard analysis happen to notice "that malevolent or unauthorized action, constituting a security threat, as [sic] reasonably foreseeable," directing readers to ISO/IEC 13335 (withdrawn, not revised) and IEC 62443 for guidance [43,44]. "If security threats have been identified, a vulnerability analysis needs to be undertaken," again according to ISO/IEC 13335 and IEC 62443. But IEC 61508:2010 does not identify the possibility that safety and security goals might conflict, provide guidance for resolving such conflicts, or specify how the satisfaction of security requirements should be verified.

be an example of this.) But a complete safety standard should help developers to make and regulators to assess such choices about what software should be relied upon to do. To focus exclusively on correctness is to focus on one threat to safety to the exclusion of others.

In any case, whether rightly or wrongly, most standards for software in safety critical applications are focused more on correctness than on safety. Accordingly, some of the definitions of 'works' discussed below need to be considered in two forms: one related to correctness, the other to software's impact on system safety.

3.2 Bounding Uncertainty in Behavior or Safety Claims

A standard for software in safety critical applications might be said to work if its use reduces and bounds the uncertainty in a behavior or safety claim of interest. For example, RTCA DO-178C claims to "provid[e] the aviation community with guidance for determining, in a consistent manner and with an acceptable level of confidence, that the software aspects of airborne systems comply with airworthiness requirements" [3]. This definition implicitly acknowledges that it is impossible for conformance to a standard to *entail* a behavior or safety claim (unless that claim is trivially true, in which case the standard is irrelevant). Nevertheless, reducing or bounding the likelihood that a behavior or safety claim is false is useful.

This view is nearly equivalent to the view that standards for software in safety critical applications are human factors standards meant to increase the likelihood of spotting any mistake that would compromise system safety before systems are deployed. If we presume that mistakes are addressed in a way that improves system safety (or correctness), however slightly, and if conformance to the standard does not cause developers to make additional mistakes, then spotting more mistakes would result in greater confidence in a claimed level of safety (or correctness).

This view is also related to the *filter model* of safety-critical system certification [45]. In the filter model, a certification procedure comprises a *filter* and a *decision procedure*. Each assurance requirement or group of requirements is a filter that is used to identify faults in a system. As in the Swiss Cheese accident model [46], each filter might catch defects that other filters in the same standard miss. The overall goal of the filters is to allow certifiers to determine 'whether faults remain that would subject the public to an unacceptable level of operational risk' [45].

The *bounding uncertainty* definition of what it means for a software safety standard to 'work' has the virtue of being the most directly relevant to decisions about whether to deploy, continue operating, or improve software in safety critical applications. However, this definition is sometimes disparaged on the grounds that we cannot, at present, show that use of a standard has this effect. The present state of knowledge is too limited to claim that a standard either works or does not work using this definition. Some standards might not. However, this definition might be useful even if it is untestable: it seems to captures an important aspect of what many speakers mean when they say that a standard 'works.'

3.3 Addressing the Risk From Specific Mistakes

A standard for software in safety critical applications might be said to work if its use reduces the incidents of mistakes that are known to threaten safety. Accident and incident investigations often report 'lessons learned'; we might say that a standard 'works' if following it makes developers less likely to make one of a set of mistakes of interest or more likely to catch such a mistake before it can impact safety. This set might be drawn from research or even intuition rather than from history alone. Ideally, the set would be chosen to include the mistakes that give rise the greatest risk, i.e. those that would occur the most frequently without intervention or that lead to the most severe or likely accidents.

This definition differs from the previous one in that its focus is not on the total uncertainty in a safety or correctness claim, but rather on specific contributions to that uncertainty. Using this definition, we might say that a standard works even if it is possible for systems to contribute to an accident in ways that the standard does not address. To the degree that a particular system or its development is like prior systems, addressing the ways in which those systems went wrong should, in general, make the system safer. However, this definition raises the possibility that a standard might have far less of an effect on safety or correctness (or confidence in these) if the application or development techniques are novel. For example, a rule prohibiting most backward branches might have significantly reduced the incidence of control flow problems when most embedded software was written in assembly language, but such a rule is not applicable to software written in most high-level languages.

3.4 Promoting Developer Competence

A standard might be said to work, not because it (only) allows assessors to detect and filter out unsafe systems, but because it promotes developer competence. It might promote competence by teaching good practice to developers, acting as a barrier to the entry of incompetent developers or organizations, or both.

One function of a standard might be to teach developers how to build acceptably safe systems. We are not aware of standards committees explicitly writing a standard as a pedagogic tool and would not recommend that developers learn how to build critical systems *solely* by reading standards, but a standard might (in part) have this effect. Developers using a standard read it, come to understand its assurance requirements, seek information about the recommended means of satisfying these, and possibly create their own concrete development or assurance plan to satisfy those assurance requirements. These activities might impart knowledge and habits that would have a beneficial effect even if conformance were not required in the future.

A related function might be to serve as a barrier to entry that precludes the development of safety-critical systems by people who lack the requisite skills and experience or organizations that do not follow appropriate practices. In this view, a standard—or perhaps a compliance enforcement regime related to that standard—poses a difficult-to-surmount barrier to the people whose work might pose the most danger but a manageable hurdle to the people who are likely to produce acceptably safe systems with or without the standard.

Several workshop participants opined that if you took the standards away from the best development teams, they would carry on building acceptably safe software[9]. Both of these views of what it means for a standard to work are consistent with that opinion.

One workshop participant observes that standards might work in a related way, namely by giving developers a way to resist pressure to cut corners. In that view, a standard doesn't promote developer competence, but it gives developers an excuse to practice competently even if budget pressures result in management suggesting that more expensive safety techniques could be dispensed with.

3.5 Giving Certifiers a Tool for Improving Safety

A standard might be said to work, not because of what developers reading it learn, but because it gives certifiers[10] a tool for improving safety or blocking the deployment of unsafe systems. In this view, a standard is simply the necessary foundation of a safety assessment regime[11]. Devel-

[9] One workshop participant opined the opposite, namely that if you took the standards away from the best development teams, their software would become unsafe.

[10] The mechanisms of conformance assessment vary from standard to standard and, sometimes, jurisdiction to jurisdiction. For example, in the United States of America, the Federal Aviation Administration allows aircraft developers to submit the life cycle data specified in RTCA DO-178C [3] to one of its certification offices as a means of "showing compliance with the applicable airworthiness regulations for the software aspects of airborne systems and equipment certification" [47]. Developers in the automotive domain *may* choose to have a third-party company assess their conformance to ISO 26262 [25], but we do not know of a domain where independently-certified conformance to ISO 26262 is mandated by law or regulation. For the purpose of this definition of what it means for a standard to 'work,' a certifier is any organization that assesses conformance to a standard and can either certify conformance or refuse to do so. Legislation, regulation, and policy might impact the business importance of gaining such certification—and thus the efficacy of a standard under this definition—but the basic idea behind this definition applies to some degree even where standards conformance is purely voluntary.

[11] For example, a standard gives certifiers and developers a common terminology and framework, both of which are necessary for productive discussions about safety. The standard also forces developers to give certifiers access to artifacts such as requirements, source code, test results, and traceability information.

opers are motivated by the possibility that assessors will refuse to certify compliance with the standard, either because compliance is mandated by law or regulation or because a mark of compliance is perceived to add value to the system in question. This motivation might encourage developers to follow better practices than they otherwise would, thus improving their effective competence. But where it doesn't, assessors might agree to certify systems only if developers agree to use a safer design or better tools or practices.

In this view, it isn't necessary for a standard's assurance requirements to be completely effective as a means of bounding confidence in safety, filtering out unsafe systems, or teaching developers best practice. The assurance requirements needn't necessarily be unambiguous. If assessors are capable of determining whether a given system is acceptably safe or whether an alternative would improve safety, the leverage afforded by the certification regime could allow them to enforce or improve safety.

3.6 Cost-Effectiveness

The definitions of 'work' given in Sections 3.2–3.5 focus on the safety-related effects of standards. But standards also impact the cost of development and certification. Decisions about whether to use a standard, which standard to use, or what assurance requirements to include in a standard might turn not only on whether a standard works, but whether it is *cost-effective*.

While decisions about whether a safety improvement is worth its cost are unavoidably political, a scientific assessment of standards might nevertheless produce data that could be useful to those making such determinations. For example, we might assess the feasibility and cost of applying a standard and present this information to the relevant decision-makers alongside any assessment of whether that standard works.

If assessments of standards are to inform decisions about cost-effectiveness, researchers must plan their studies with the needs of the relevant decision-makers in mind. This is because it may not be sufficient in all cases to simply supplement assessments of safety-related effect with information about cost. For example, consider a study that compares the effect of two competing standards on confidence in software correctness. If the study design can answer the question, *Which works better?*, but not the question, *By how much?*, the results would lack information needed to make a decision based on cost-effectiveness.

4 Assessing Standards

Knowing what it means for a standard for software in safety critical applications to work would be only the first step in assessing that standard. In this section, we outline strategies for assessing standards. Answering the question of whether or not a standard is effective will likely require multiple studies; these over-arching strategies define what individual studies must show and how their results relate to the overall conclusion. We will turn to the issue of specific study methodologies in Section 6. Here, we outline four distinct strategies: (1) measuring the effect of using a standard, (2) assessing the standard in parts by assessing the efficacy of each assurance requirement, (3) assessing necessary conditions for efficacy, and (4) assessing how well a standard solves the problems of the past.

4.1 Measuring the Effect of Using a Standard

Suppose that we accept the view that a standard works if it bounds uncertainty in a claim about system safety or software behavior (see Section 3.2). We might then try to assess the standard by directly measuring its effect on accident rates, incident rates, or post-release defect density. For example, we might measure the number of safety-relevant defects found in software after deployment[12] and compare these numbers across software developed in conformance with different standards or to different integrity levels of the same standard.

The intuitive appeal of this approach is its directness: rather than rely on the results of many studies, each of which assesses only one aspect of what it means to work or one contributing factor to how a standard is thought to work, we might be able to assess the standard with a single study. Such directness has the potential to reduce epistemic uncertainty in a claim about whether a standard works.

However, there are at least three problems with this approach: (1) accidents might be too few and far between to yield useful measures, (2) it is difficult to hold all other things equal in comparison studies, and (3) the necessary data is difficult to acquire. If the study is ongoing, there might also be pressure to change policy based on preliminary results, effectively terminating the study before strong conclusions can be drawn. Suppose a string of incidents is reported. These might reflect either (a) an effect that could be used immediately to save lives or (b) statistical chance and reporting bias [48]. In the former case, it is more important to make a change in policy or practice than to gather more data, but making such a change would halt collection of the data that could tell us which interpretation is correct.

In some industries, our aim is to develop systems that make a defined category of accidents unlikely over the entire operating lifespan of all

[12] If, all other things being equal, fewer defects are discovered in software developed to standard X than to standard Y, conformance standard X results in lower uncertainty in a safety claim than conformance to standard Y. Note that the phrase 'all other things being equal' is crucial here. Correctness is not the same as safety: many factors affect the safety impact of each defect. We note that it might be difficult to adequately hold all other things equal when comparing the effect of two standards in this way.

systems of that type[13]. If we actually achieved that—or very nearly achieved it—there would be insufficient data in the operational history of such a system from which to draw a statistically-valid conclusion about whether such a goal had been met.

A correlation between standard type or level and accident or defect rates does not conclusively demonstrate a causal link between the two. However, an observed *lack* of correlation would demonstrate the absence of a useful causal effect. But for such a study to be valid, it would have to control for any other factor that might plausibly affect the measured outcome. This would be very difficult to do for standards. For example, we do not know whether differences between industries or application areas would affect the efficacy of software development, verification, and validation practices. This makes it difficult to compare, say, aviation software certified to DO-178B [5] with industrial control software certified to IEC 61508 [51] or rail software certified to EN 50128 [52]. We do not know whether advances in software development practices not discussed by the standards would affect the number or type of defects introduced by the software. Since such practices change over time, it might be similarly difficult to compare older aviation software certified to DO-178B [5] with newer software certified to DO-178C [3].

Any study of post-deployment defect density would require access to reliable information about *all* safety-relevant defects discovered in the specimen software. Such information might be very difficult to obtain for research purposes. While some organizations that build and maintain software keep such records, they may not be willing to disclose them for fear of legal ramifications, loss of prestige or sales, or disclosure of proprietary information about how their products work.

4.2 Assessing a Standard by Parts

Again supposing that we accept the view that a standard works if it bounds uncertainty in a claim about system safety or software behavior, we might try to assess a standard by parts. That is, we might try to separately assess the effectiveness of each of its assurance requirements and then infer an assessment of efficacy of the standard as a whole. (This is analogous to many standards' recipe for evaluating software's contributions to safety, in which many pieces of interrelated evidence are collected separately.)

[13] U.S. regulations require that transport category "airplane systems and associated components, considered separately and in relation to other systems, must be designed so that ... the occurrence of any failure condition which would prevent the continued safe flight and landing of the airplane is extremely improbable" [49]. The phrase 'extremely improbable' is interpreted to mean that extremely improbable events "are those so unlikely that they are not anticipated to occur during the entire operational life of all airplanes of one type" [50]. For software-based systems, developers might choose to conform to RTCA DO-178C [3] as a means of showing conformance with relevant regulations [47].

One approach to doing this is to capture and then assess a standard's implicit argument [53]. That is, we might read a standard's overall objectives, internal rationale (if any), and assurance requirements and document the argument connecting satisfaction of the requirements (as evidence) through the rationale (as intermediate claims and argument structure) to the conclusion that the objectives are satisfied (e.g., software defects do not cause the system to pose unacceptable risk). (Such an argument has been extracted from the Common Criteria for Information Technology Security Evaluation [54]. A similar argument is being constructed for DO-178C [55].) If we could separately evaluate whether each assurance requirement (or the typical means of satisfying it) accomplishes what the standard requires it to, we might be able to assess confidence in the argument's main claim and thus the efficacy of the standard. The complete evaluation of a standard by parts would require many experiments to assess the efficacy of individual techniques. (Section 6.1 gives an example of one such experiment [11].)

One potential problem with that approach is that it might be difficult to identify (or obtain agreement on) the standard's argument. Michael Holloway observed that SC-205—the committee charged with updating DO-178B [5] to produce DO-178C [3]—tried but failed to construct such an argument for the standard they were drafting. John McDermid noted that some standards, such as the UK Ministry of Defence's safety management standard [56], have assurance requirements so broadly defined and broadly applicable that it would be difficult to draw strong conclusions using this approach.

Another difficulty is that there might not yet be broad agreement on the premises for such an argument. For example, there are differing opinions about the efficacy of Modified Condition/Decision Coverage testing [20] and little empirical data in the public domain. Evaluating a standard in this way might first require empirical assessment of a great many safety-related techniques.

A final difficulty is that the value of a whole might not depend solely on the individual values of its parts [11]. If the efficacy of a technique depends in part on whether it is deployed in conjunction with other techniques, then we would need appropriately specific data about its efficacy in order to evaluate a standard in this way.

4.3 Assessing Necessary Conditions for Efficacy

One general approach to assessing whether a standard works is to identify ways in which it might not and assess whether it fails to work in each such way. Stating this approach slightly differently, we might identify necessary conditions for a standard to work and assess whether each has been met.

For example, we might suppose that, for a standard to work, its assurance requirements must be unambiguous. (See Section 7.3.) We might then perform an experiment using human subjects to determine whether assessors interpret objectives in the same way.

At the workshop, Virginie Wiels proposed an *abstraction view* model of certification that defines necessary conditions that might be evaluated in this way [15]. In her model, certification is analogous to verifying software by machine checking a formal abstraction of it. For example, we might determine whether an applicant has met an assurance requirement of DO-178C as an assessment of an abstract view of the software. If the abstraction has the required properties, making such conformance determinations effectively assesses whether the software is correct. According to this model, if a standard is to work, its abstract view of the software must have three properties: (i) soundness, (ii) tractability, and (iii) precision. If the abstraction is sound, i.e. "it defines more behaviors than the actual behaviors of the program," then when the assurance requirements are met, the software is correct. The abstraction is tractable if it is practicable to determine whether or not the assurance requirements are met. If the abstraction is too imprecise, checking reveals failure to meet assurance requirements that do not correspond to ways in which the software is incorrect.

The virtue of the necessary conditions approach is that it might be easier to assess whether a specific condition is met than to assess the overall effect of a standard. For example, it is clear that configuration consistency is a necessary condition: if we do not know which version of the source was used to compile a given test binary or the delivered executable code, any conclusions derived from testing might not apply to the software that will actually be used in the field. The meaning of configuration consistency is more concrete than that of whether a standard works and configuration consistency could—at least in principle—be assessed by an auditing process. The disadvantage of this approach is that it is more useful for revealing problems that need to be addressed than for showing that a standard works: because we might not know all of the necessary conditions, we might find that all identified conditions are satisfied even if the standard in question does not work.

4.4 Assessing How Well a Standard Solves the Problems of the Past

One other general approach to assessing a standard is to determine how well it solves the problems of the past. For example, suppose that several systems in one application area fail (in part) because software failed to meet a response time deadline. Suppose also that a new version of the relevant standard has been introduced and that this includes new assurance requirements meant to improve confidence that deadlines will be met. In such a case, we might monitor software that conforms to the new version of the standard and, if fewer overruns are seen in practice, conclude that it works better than the old version.

This approach is a variant of assessing the effect (see Section 4.1) and so will suffer from similar problems, e.g., the observed problems might be too rare to support statistically strong conclusions or the data might be difficult to acquire. It has the virtue of focusing on aspects of what it means for a standard to 'work' that experience has shown us are important. (It might be that some things that we might not know about a standard simply don't matter in practice for reasons we do not yet know.) However, it also has a corresponding limitation: as new applications are tackled and new development techniques introduced, software might begin to fail more in one way than another, or even in ways not seen in the past. This limitation might be more problematic in domains where practice is changing rapidly than in domains where practice is more settled.

5 Research Questions

In the submitted papers, authors identified several research questions related to whether or not a standard could be said to work. In the workshop discussions, participants identified several more. In this section, we present a selection of the most relevant research questions identified.

While these questions are not presented in any well-defined order, readers should note that many are interrelated. For example, if the answer to RQ5 or RQ7 is 'No,' then many of the other questions might not be worth answering. If it is not practicable to answer RQ5, it would very likely be impracticable to answer RQ4.

We do not claim that it is practicable (or even possible) to answer all of these questions. In Section 6, we present six proposed studies and discuss challenges related to each. The listed challenges illustrate the difficulty of answering research questions like these.

RQ1. *Does conformance with a given standard mean that it is acceptably likely that a system is acceptably safe (for a given kind of application)?*

This question, while difficult to answer, seems the most straightforward way of capturing the reasoning behind decisions to accept or reject a complete system based on conformance verification. 'Acceptably safe' might be defined more concretely in terms of a rate of dangerous failures or a rate of accidents or incidents.

RQ2. *Does conformance with a given standard mean that the software's contributions to system hazards and their management are acceptable?*

Some standards (or parts of standards) focus on software only. Since software in isolation cannot be said to be 'safe' or 'unsafe,' we must ask instead about software contributions to system safety.

RQ3. *Does conformance with a given standard mean that it is acceptably likely that the software satisfies its requirements?*

Some standards—including DO-178C—focus mainly on software correctness, excluding from its scope the effect of the given software requirements on system safety. (See Section 3.1.) This formulation addresses only the issue of correctness. Note that the degree to which software satisfies its requirements might be expressed in several different ways, e.g. in terms of defects per line of code, reliability, or availability.

RQ4. *Does conformance with standard* X *mean that, all things being equal, a system is safer (or software more likely to meet its requirements) than if it conformed instead to standard* Y *or to no standard at all?*

If we can't determine whether conformance is enough to demonstrate adequate safety, perhaps we can show that conformance *improves* safety relative to plausible alternatives. An answer to this question might inform a decision about which potentially-applicable standard to require, use, or meet. An answer to this question might better inform such decisions if it also answers the follow-up question *By how much?* However, it is possible that a useful answer might be context-specific. For example, systems conforming to standard X might be safer under some circumstances and less safe under others, or provide more protection against some types of risk and less protection against others.

RQ5. *Does conformance with standard* X *at level* Y *mean that, all things being equal, a system is safer (or software more likely to meet its requirements) than if it conformed at level* Z*?*

If conformance to a standard at a higher integrity level produced no effect, the extra effort spent would be wasted. To inform cost-benefit decisions, we might also need to know the magnitude of the effect.

RQ6. *Does the number or distribution of mistakes that developers make change when a standard (or a new version of a standard) comes into widespread use?*

If conformance to a standard makes developers less likely to make one of a selected set of mistakes (see Section 3.3), then the introduction of a standard targeting those mistakes should result in a change in the rates at which those mistakes are made.

RQ7. *Can we reliably determine whether a system (and its development process and development artifacts) conforms to a given standard?*

Conformance judgments must be repeatable if conformance is to *cause* a large, reliable safety effect. If conformance judgments were unrepeatable and yet conformance produced a substantial safety effect, we might suspect that the effect was due less to filtering out unsafe systems than to educating developers or encouraging developers to be more self-consciousness.

RQ8. *Do a given standard's assurance requirements feature vagueness or other features that might give them multiple plausible interpretations?*

This question addresses a necessary condition of a necessary condition (see Section 4.3) for bounding uncertainty (see Section 3.2): a standard might bound uncertainty well if conformance judgments are not reliable, and conformance judgments might be unreliable if assurance requirements are not clear and unambiguous. The same question also addresses a necessary condition for a standard to work as a teaching tool (see Section 3.4): ambiguity might lessen a standard's educational benefits.

RQ9. *Can we formalize standards' assurance requirements? Would this significantly reduce variance in conformance evaluations?*

'Formalization' might mean many things, but we might hypothesize that some forms of formalization might make assurance requirements less ambiguous and conformance judgments more repeatable. As with any proposed enhancement, we might ask whether formalization in a proposed way has the intended effect and whether this effect is worth the cost.

RQ10. *Does a standard permit innovation? That is, does it allow developers to use techniques that are not yet widely used in safety-critical applications yet produce systems that are just as safe?*

Computers, software, and software engineering tools and techniques have changed rapidly. If a standard does not permit innovation without changes to its assurance requirements and can't be revised (and then re-assessed) rapidly enough, developers might be unable to take advantage of advances that would lower development cost, increase confidence in software behavior, or enable the implementation of complex software-based safety features that lower overall risk. Note that information about the effectiveness of new tools and techniques is critical to ensuring that their use maintains safety. However, reliable studies demonstrating the efficacy of a new technique might lag its introduction by many years—if such studies are done at all.

RQ11. *How do factors such as degree of flexibility or prescriptiveness or type of submission documentation (e.g., safety case or accomplishment summary) affect acceptance rates and the number of safety-relevant problems discovered during assessment?*

Assessors' acceptance rates for similar systems might vary for several reasons. For example, a more flexible standard might cause assessors to see more novel forms of safety evidence, which they might either be more likely to reject (on the grounds that novel types of evidence are 'unproven') or more likely to accept (because they do not know the weaknesses of these new forms). The rate at which assessors discover safety-relevant problems during assessment might also vary for several reasons. For example, different forms of documentation might be more or less revealing of developer competence or the safety implications of design decisions. Understanding the effect of these factors on assessment outcomes could help to design a standard that facilitates accepting more safe systems and fewer unsafe systems. Studies to address concrete instances of this research question would have to control for other factors that might affect the measured property, e.g., whether conformance is purely voluntary or an accepted means of compliance with regulation, whether the assessor is chosen and paid by the developer or a government agency, etc.

RQ12. *Do factors such as the type of submission documentation affect developers' understanding of the safety impact of their actions?*

Anecdotes from developers who have constructed safety arguments suggest that the process forced them to think more carefully than they had done before about the purpose and meaning of the safety-related development activities they undertook[14]. If better education leads to better safety outcomes, then, all other things being equal, a form of submission that better educates developers should lead to better safety outcomes.

RQ13. *What sorts of defects does a given design and construction approach introduce? What sorts of defects does a given verification and validation approach find?*

One way of evaluating the effect of a standard is in terms of the *types* of defects that might be introduced during design and development and the effectiveness of verification and validation in detecting these. For example, if two standards permit the use of a given design and construction approach, and that approach is known to introduce a certain type of error, then a standard would be better if it caused the use of a verification and validation approach that finds those types of errors than if it did not. Answers to this question are unlikely to produce a total order of how well standards work, but might be useful for more specific decisions

[14] One of the authors, Graydon, has heard this from developers at two separate organizations he has assisted to develop safety arguments. Similar anecdotes have been related by other researchers who have worked with industrial developers building safety arguments for the first time.

such as whether to allow a proposed alternative means of conformance to an assurance requirement. The question presumes that it is possible to identify categories of defects that reveal clear differences between design, construction, verification, and validation approaches; this might prove difficult.

RQ14. *Are there assurance requirements in a standard that are always satisfied (without remediation)?*

If (nearly) every product and development process is found to meet a given assurance requirement, we might ask whether it is superfluous. Universal or near universal conformance *might* be a symptom of a requirement addressing a problem that does not appear frequently in practice. But there might be another cause or causes for a high rate of conformance. For example, if conformance with the assurance requirement is easily and reliably checked, applicants might be in the habit of self-checking before submission to reduce the time and cost penalty of re-submission. An answer to this question does not identify assurance requirements that can be removed, but it might identify assurance requirements that are worthy of greater scrutiny.

RQ15. *How likely is it that each of the various software development and safety assessment techniques fails to serve the safety purpose it is meant to serve? (For example, how likely is it that hazard analysis would fail to identify a hazard?)*

In cases where a standard mandates or highly recommends the use of a given technique, we might ask how effective that technique is. All things being equal, the more effective the required techniques are, the more effective the standard should be. Answers to this question might also inform judgments of whether an alternative technique might be more effective and the adequacy of a proposed alternate means of compliance with a given assurance requirement. However, answers to this question will only be useful if we know the precise purpose that each technique is meant to serve. Current standards describe this poorly (or not at all), and consensus about the purpose might prove illusive.

RQ16. *What factors influence the efficacy of each safety technique and by how much? (For example, will an outstanding code reviewer find more defects than two mediocre reviewers?)*

It is possible that the efficacy of a given safety technique is variable, e.g., that hazard analysis might find a greater proportion of a system's hazards if performed by people who have substantial experience with similar systems and applications. If we knew which factors affected efficacy, we might be able to improve standards and certification by providing more targeted guidance. A more ambitious goal would be to build a model of how *measurable* factors impact the efficacy of a safety technique. If such a model could account for most of the variance observed in efficacy, it would make judgments of sufficiency more repeatable and facilitate recommending concrete improvements.

RQ17. *Would conformance judgment be more reliable, or standards conformance yield a better balance of operational risk and expense, if standards expressed the degree to which each assurance requirement was meant to address operational and development risks?*

Some assurance requirements seem to target both *operational risk* (i.e., risk to humans or the environment that arises from the system's behavior or is meant to be addressed by the system) and *development risk* (i.e., the risk that developers will fail to construct an acceptable system). Safety regulators might justifiably be more concerned with the former than with the latter (although there is some cause for concern that late discovery of a safety problem will lead to pressure to make exceptions). Knowing what type of risk each assurance requirement is meant to address might plausibly help assessors to focus on factors that affect operational risk while allowing developers to take on the development risk that the developers and their customers are comfortable assuming.

The number of questions identified is itself striking: there are many distinct aspects of what it means for a standard to 'work' and many factors that would affect how well it does. Moreover, it is noteworthy that some questions apply only to some kinds of standards. For example, RQ1 applies only to system-level safety standards, while RQ3 applies only to standards for software used in the context of a broader safety process that provides software safety requirements. Where different questions apply to two standards, it might be difficult for a regulator to use answers to them to decide which standard to accept as a means of satisfying applicable legal or regulatory requirements.

6 Proposed Studies

The aim of the AESSCS workshop was to *plan* the unplanned experiment by soliciting ideas for empirically assessing the efficacy of standards for safety-critical software. The accepted position papers identify hypotheses of interest, propose study or experiment strategies, and identify threats to validity. During the workshop, participants discussed the proposals, offering suggestions to further refine them. In this section, we briefly outline the six most concrete research proposals discussed during the workshop. These six proposals illustrate the difficulty of empirically assessing the efficacy of standards.

6.1 What Is The Value of Formalizing Requirements?

Habli and Rae propose a concrete experiment that is part of the general approach of assessing a standard by parts (see Section 4.2) [11]. They propose addressing RQ15 and RQ16 in the specific case of formalization of software requirements. Their proposal begins with the observation that, in the course re-writing natural-language software requirements in a formal notation, people have identified requirements errors. While this might lead us to hypothesize the there would be fewer requirements errors if a standard required developers to express software requirements in a formal language, the anecdotes alone cannot support such a hypothesis. The authors propose rigorous experiments to do so.

Hypotheses. Habli and Rae give a set of alternate hypotheses to explain the observed ability of requirements formalization to uncover previously-unknown requirements defects. It might be: (i) that the errors are revealed by the increased precision of the formal notation, (ii) that the errors are revealed by the process of re-expressing the requirements (regardless of the notation used), (iii) that the errors are being discovered because the people doing the formalization are experts, (iv) that any effort would find errors regardless of method or expertise, (v) that formal notations enable the use of automatic tools that are finding the errors, or (vi) some other explanation best explains the observation.

Method. An experiment on human subjects to measure the number of defects found by participants. Independent variables represent the key parameters for each alternative explanation: (i) formality of notation used, (ii) difference between source and target notations, (iii) expertise of subject, (iv) task duration, and (v) degree to which automated analysis is enabled. Vary independent variables singly and in combinations; the best explanation is that associated with the variables that, when manipulated, produce the biggest gain in number of defects found.

Threats. Habli, Rae, and the other participants identified several potential threats to feasibility and validity:

1. The type or form of requirements might affect the ability of the formalization process to identify requirements defects (thus making the results dependent on the subject requirements used in the experiment).

2. Some independent variables might have a non-monotonic effect on the dependent variable.

3. The discovered defect might be the result of systems engineers asking for the wrong thing instead of the requirement not being

communicated clearly (in which case having *them* formalize the requirement might not help).

Some workshop participants expressed a worry that the results of such an experiment might not be generally accepted. If the experiment showed that engineers expressing requirements in a given formal notation are less likely to introduce errors in those requirements than engineers expressing requirements in natural language, skeptics might question whether the effect was worth the cost. If the experiment did not show that effect, some researchers might move the goalposts and claim that formality is still valuable because it enables techniques that find errors in formalized requirements. Such behavior would be consistent with Kuhn's description of the way scientific communities prefer to refine models and adjust auxiliary hypotheses rather than discard cherished beliefs [57]. Such refinement is valuable, and an accumulation of anomalies and inconsistencies through experiment is a necessary pre-condition if the current models are eventually to be discarded.

6.2 Can Mere Mortals Assess Safety Arguments?

Holloway and Johnson observe that "a common criticism of [argument-based] standards is that using them will increase substantially the intellectual burden on individuals within regulatory authorities by requiring unrealistic general levels of competence" [12]. Anecdotes, for example from the UK Civil Aviation Authority's transition to safety-case based submissions, suggest that this might not be a problem in practice, but anecdotes are no substitute for data from well-constructed studies. Holloway and Johnson propose to determine whether evaluating an argument about safety requires knowledge and skills (e.g., those a logician or philosopher might have) that typical safety assessors do not have.

Hypothesis. Safety assessors are incapable of evaluating the adequacy of a safety argument, particularly one that does not rely on conformance to a process-based standard [12].

Method. Experimentation in which human subjects—safety assessors—would evaluate the adequacy of a safety argument.

Threats. Holloway, Johnson, and the other participants identified several potential threats to feasibility and validity:

1. It might be difficult to create a specimen system and documentation that is both realistic and sufficiently simple that (a) ground truth can be determined and (b) participants can complete the experiment during the time they are willing to spend as subjects.

2. The amount of training subjects receive (if any) might affect the outcome.

3. The presentation of the argument(s) might affect the outcome.

4. Differences between domains (e.g., rail versus aviation) might affect the outcome.

5. The study itself might be viewed as impugning the competence of safety assessors, reducing participation and perhaps biasing results.

6. It might be difficult to recruit an appropriate 'representative sample' of assessors. For example, if one recruits readily-available subjects such as students, the results might not generalize to safety assessors. If one recruits professionals by advertising at a conference, any self-selection bias in the conference attendees becomes a bias in the sample. If one cannot recruit enough participants, the study results might not achieve statistical significance.

Related Studies. One factor that might make it difficult to assess a safety case is that while more prescriptive standards constrain a developer's choice of tools, techniques, and design features, a safety argument could, in principle, include *anything.* If assessors are routinely presented with claims about unfamiliar tools, techniques, or design features, they might have difficulty assessing those claims. But it is not yet known that developers do, in practice, create arguments around unfamiliar evidence. We could survey developers or assessors working under both more argument-centric certification processes and more prescriptive processes to determine whether there is an appreciable difference in practice. We might also survey assessors to find out what they know, what they would need to know, and what they are comfortable assessing. (See Section 6.3.)

A related question is whether it is possible for ordinary developers to *write* good safety arguments. There is ample evidence that bad safety arguments exist. While there are very few safety arguments in the public domain, some of these are known to contain errors in reasoning [58]. A publicly-available review into the issues surrounding the loss of one aircraft found the production of the aircraft's safety case to have been "a lamentable job from start to finish" [59]. Participants in this workshop have seen poor reasoning and writing in safety arguments that are not publicly available. The existence of bad safety arguments, together with a general lack of publicly available examples of excellent arguments, raises the question of whether it is possible for normal developers, working under normal conditions, to produce a good safety argument. This question is also worthy of serious study.

6.3 What Do Safety Assessors Know? What Must They Know?

Holloway and Johnson observe that it is difficult to design experiments such as the one proposed in Section 6.2 without an accurate understanding of the current state of practice [12]. They propose a survey to collect information about the current state of regulatory practice.

Research Question. What do conformance assessors know? What do they need to know?

Method. Survey assessors. Enquire about education and training, the skills assessors presently find most useful, knowledge about reasoning and argumentation, which artifacts assessors subject to the most thorough auditing, etc.

Threats. Holloway, Johnson, and the workshop participants identified several potential threats to feasibility and validity:

1. It might be difficult to recruit volunteer participants and a small response rate would make it difficult to generalize from the data.

2. It is difficult to develop unbiased survey questions.

3. The education, training, skills, and knowledge respondents perceive as most important might not be the most important.

6.4 Do Higher Levels Yield Lower Uncertainty?

Daniels and McDermid separately observed that if a standard with multiple integrity levels worked, we would expect that it would work better at higher integrity levels than at lower integrity levels [9,13]. They propose a range of study techniques that might determine whether this prediction holds.

Research Question. Does conformance to a higher integrity level reduce the likelihood that software will contain safety-relevant defects?

Methods. Daniels and McDermid identified several forms of study that might (partially) answer this question:

1. A historical study of either in-service failures or defects discovered while the software was in service.

2. Retrospective qualification of existing software to various integrity levels of a subject standard.

3. Expert review of the standard's assurance requirements.

4. An experiment in which subjects (possibly students, preferably practicing professionals) construct software in conformance with the subject standard at various integrity levels.

Threats. Daniels, McDermid, and the workshop participants identified several potential threats to feasibility and validity:

1. Manufacturers might be reluctant to release historical data on the in-service history of their software (if they have it).

2. Accident and incident reports might not have the detail needed for a good retrospective study.

3. A subject software-based system that had been qualified to an existing standard might have already achieved much or all of the benefit of qualification to the subject standard at lower levels.

4. A subject software-based system that had not been qualified might differ from modern safety-related software systems in ways that make the results inapplicable.

5. It is not known whether expert judgment is a valid way to determine what the effect of applying a standard will be.

6. The performance of student subjects might not be a valid predictor of the performance of qualified professionals.

7. It might be difficult to recruit a significant number of volunteer professionals.

8. It might not be possible to determine how many safety-related defects a specimen software system has (particularly without fielding it for many years).

6.5 What Likelihood of Correctness Does Conformance Produce?

Rushby, Littlewood, and Strigini propose to model the probability $p_{srv}\left(n\right)$ that software will survive n independent demands without failure as

$$p_{srv}\left(n\right) = p_{nf} + \left(1 - p_{nf}\right) \times \left(1 - p_{F|f}\right)^{n} \qquad (1)$$

where p_{nf} is the probability that the software is fault-free and $p_{F|f}$ is the probability that the software will fail on a given demand if faulty [14, 60]. At present, we do not know either figure but can make worst-case assumptions for $p_{F|f}$. The main question is then about p_{nf}.

Hypothesis. That it is possible to determine, on the basis of conformance to a standard, a value for p_{nf} for a class of systems that makes a plausible basis for making conservative yet acceptable claims about p_{srv} for those systems.

Methods. Assessing this hypothesis begins with establishing p_{nf} for a specimen class of systems. The hypothesis requires (a) a method of estimating p_{nf} on the basis of conformance, (b) that this method and Equation 1 be acceptable to certifiers/regulators, and (c) that the method produces values of p_{srv} high enough to be useful. Rushby, Littlewood, and Strigini propose a preliminary test of feasibility in the form of a combination of survey and retrospective study. The survey would ask certifiers what p_{nf} they might assess for each group of DO-178B objectives [14]. An estimate for overall p_{nf} would be calculated based on the results. Retrospective studies would examine whether systems in the specimen class have exhibited failures in operation.

Threats to validity. The authors and other participants discussed several potential threats to feasibility and validity of both the preliminary test and the method as a whole:

1. Certifiers might not accept the underlying mathematical model.

2. Assessors' opinions about the implications of a standard's assurance obligations might not be reliable.

3. Accident reports might not contain all of the details needed for retrospective assessment.

4. Some failures might go unreported.

5. Whether software is fault free might depend on (possibly unknown) factors other than conformance to a standard so that the p_{nf} estimated for some class of systems does not apply in another.

6. It might not always be appropriate to make 'conservative' assumptions: an incorrectly *low* estimate of reliability might cause system designers to target the wrong component for safety improvement.

6.6 Can We Measure Software Reliability?

Many standards for software in safety-critical applications (e.g., RTCA DO-178C [3], IEC 61508-3 [7], and CENELEC 50128 [52]) evaluate software indirectly through an assessment recipe: developers and independent assessors conduct a variety of reviews, analysis, and tests of both the final software and the development artifacts from which the software is derived, and assess the software on the basis of the collective results. Ashmore proposes assessing an alternative form of software standard in which software failure rates are measured directly [8]. If such measurement is practicable, a very simple standard based on direct measurement might inspire more confidence than those based on a more complicated, less direct assessment approach.

Hypothesis. That it is possible to directly measure software reliability in at least some systems of interest.

Method. Demonstration: measure the reliability of software in some systems of interest.

Threats to validity. The authors and other participants discussed several potential threats to feasibility and validity:

1. The test hardware might not faithfully replicate the target hardware.

2. The distribution of test cases might not accurately reflect the software's operational distribution.

3. The test oracle must be automated and assumed to be correct.

4. Lack of knowledge about internal state might preclude estimating the reliability of black box software.

5. The state space needed to model timing aspects might preclude estimating the reliability of real-time software.

6. If the test hardware and sequencing mechanism require the use of test stubs to replace I/O routines in the tested code, the results might not apply to the software as a whole.

7 Observations

During the presentations and the discussions that followed, workshop participants made a number of insightful observations. In this section, we recount some of the most interesting of these.

7.1 There is the Standard, and Then There is the Standard As Applied

It is not possible to read the text of even the most prescriptive standard and determine *exactly* what developers that comply to it do. There are at least two reasons for this: (1) the standards themselves are ambiguous, and (2) the standards might not be applied exactly as written.

No so-called ‘prescriptive’ standard is either perfectly unambiguous or perfectly prescriptive: there is always some flexibility in how developers interpret and satisfy its assurance requirements. This is often deliberate. For example, developers conforming to RTCA DO-178C might propose “alternative methods of compliance” in their Plan for Software Aspects of Certification [3]. While the standard does not make a direct statement of the purpose of allowing alternative means, it does note that “various [other] national and international standards for software are available” and that developers “may be obliged ... to comply with additional standards.”

Moreover, in domains where conformance with a standard is used to show satisfaction of an applicable law or regulation, regulators have several options once a standard is finalized. For example, they might (a) refuse to accept conformance with the standard as a means to show satisfaction of the relevant regulatory requirements, (b) accept conformance (possibly only when bits of it are interpreted in a specified way), or (c) accept conformance with selected assurance requirements as partially satisfying a regulatory requirement. For example, FAA Advisory Circular 20-115C [47] describes how RTCA DO-178C [3] can be used to show compliance with US airworthiness regulations.

Because standards can be interpreted, means of conformance bargained with regulators, and specific interpretations mandated by regulators, the text of a given standard is not a perfect description of development or certification practice using that standard. As a result, the question of whether or not a standard ‘works’ might have to be asked and answered in reference to several interpretations or subsets of each standard.

7.2 The One-Developer Assumption

Standards are typically written as though exactly one organization develops a system or its software, with the notable exception of commercial off-the-shelf software (COTS). For example, RTCA DO-178C is written in terms of the “applicant” and notes that “matters concerning the structure of the applicant’s organization [and] the commercial relationships between the applicant and its suppliers ... are beyond the scope of this document” [3]. But systems and software are sometimes (perhaps often) created by a network of integrators and suppliers (sometimes organized into many tiers) and the arrangements between these might affect the safety of the resulting product.

For example, an integrator might have representatives on suppliers’ safety boards. Such representatives might be able to raise system-level concerns where these might affect component-level design decisions and assess the impact of component-level design decisions on system safety. This arrangement might yield safer systems than alternatives such as documentation of the assumed scope of applicability and assumed safety requirements of a component used out of context [41]. But such oversight is an expense and some suppliers would prefer greater protection of their intellectual property.

The reality of products developed by more than one development organization prompts related research questions about how such arrangements should be structured, managed, and regulated. The existence of integrator-supplier relationships also means that any assessment of the efficacy of standards might need to account for the effect of such relationships.

7.3 Ambiguous Assurance Requirements

Assurance requirements in standards for software in safety-critical applications often require interpretation [61]. This interpretation raises several questions: (1) *Are there assurance requirements that cannot be conformed to as written?*, (2) *Can conformance be determined reliably?*, (3) *Is it possible to write assurance requirements that are less open to interpretation?*, and (4) *Are some assurance requirements more important than others?* Note that question (2) can also be asked in a more specific form, namely *Can conformance be determined reliably based solely on the conformance evidence demanded by the standard?*

Some assurance requirements of current standards would seem to be impossible for all applicants to conform to exactly as written. For example, RTCA DO-178C directs developers to conduct “review and analysis activities” to “determine the *correctness* and consistency of the Source Code, including … worst-case execution timing” (Objective 6.3.4.f, emphasis ours) [3]. Despite the text, applicants are not strictly expected to assess worst case execution timing by analysis of high-level source code (alone): the standard observes that “the compiler …, the linker …, and some hardware features may have an impact on the worst-case execution timing” and notes that “a combination of reviews, analyses, and tests may be developed to *establish* the worst-case execution time” (emphasis ours). However, the words ‘correctness’ and ‘establish’ suggest that applicants should determine the *actual* worst-case execution time rather an a sufficiently-accurate estimate or conservative upper bound. For complex modern processors, this is not always possible [62]. One workshop participant labeled such assurance requirements ‘pretentious’ and related that incredulous developers had asked him how they could possibly show that they had conformed. A survey such as that proposed in Section 6.3 or an analysis of the standards’ texts such as that proposed in Section 6.4 might help to identify such assurance requirements.

We might suppose that, if assessors cannot reliably determine whether or not a standard has been conformed with, that standard cannot be effective. Fusani and Lami noted this and proposed addressing it with “a quality requirement for standards” [10, 63]. Making conformance assessment more reliable might be desirable for many reasons, including making development and certification costs more predictable for developers. However, knowing whether or not conformance assessment is reliable might not tell us whether a standard works. Suppose that we take 'work' to mean that a system that conforms is very unlikely to be unsafe or incorrect (see Section 3.2). Variation in conformance judgment might mark the difference between false negatives (safe/correct but not conforming) and true positives (safe/correct and conforming). This might be true if, for example, causal factors such as developer expertise, diligence, and organizational safety culture are stronger causes of safety or correctness than conformance with the standard's assurance requirements.

If there is significant variance in conformance assessments, we might ask whether it is possible to reduce that variance, and, if so, how. Fusani and Lami have proposed some means to do this, as have others [10, 63–66]. However, two factors might make this difficult. First, precise specification of what must be done might harm innovation. (See Section 7.1.) Second, some flexibility in interpretation might be needed to gain the consensus of a diverse standardization committee.

Finally, it might be the case that not all assurance requirements are equally important. In part, this is a matter of differences between applications. For example, in an application where deadlines are generous or missing them has little safety impact, assurance requirements related to execution time might be less important than in applications where deadlines are tighter and the consequences of missing them more dire. But this might also be a matter of assurance requirements targeting a mix of operational risk and development risk. For example, RTCA DO-178C requires developers to review and/or analyze high-level software requirements to ensure that "requirements do not conflict with each other" (Objective 6.3.1.b) [3]. If we could determine whether software met its high-level requirements with perfect confidence, this assurance requirement would serve only to save developers from constructing a system only to find that it did not satisfy a requirement and required expensive rework. But because no current evidence of requirements satisfaction is perfect, this assurance requirement also addresses operational risk. The varying balance of operational risk and development risk as motivation for assurance requirements raises RQ17.

In any case, participants agreed that current standards left room for interpretation. This highlights the important distinction between assessing a standard as applied and assessing the same standard as written.

7.4 Can Software Behavior at the System Level Always Be Anticipated?

During the workshop, several participants raised the issue of growing software complexity. Decades ago, software-related accidents made clear the difficulty of accurately predicting the safety-related behavior of software and the wisdom of implementing safety features in easily-assessed hardware where practicable [1]. Avoiding software complexity might not always yield lower risk: complex software has enabled safety advances that would not be possible with simple hardware[15]. But the recipe approach used by most system-level safety standard requires human beings to understand the potential safety implications of software behavior in order to generate sufficient software safety requirements[16]. The growing complexity of software raises a question: *Is it always possible for human beings to anticipate all relevant potential effects of software behavior at the system level?*

No workshop participant sketched a study meant to address this question. However, given the trend towards ever more software in safety-related products [68], this question needs to be addressed.

7.5 The Problem of Data Confidentiality

Assessing what works and what doesn't in safety practice sometimes requires studying details about the practices of developers and regulators and the products they build and regulate. For example, the studies sketched in Section 6.3, Section 6.4, and Section 6.5 require such information. Despite the importance of answering these questions, this information might be difficult to obtain because the people who have it regard it as confidential.

Development organizations might be reluctant to share information with researchers for many reasons, including: (1) fear that the revealed information might prove embarrassing, (2) concern that details revealed could be used in a lawsuit, and (3) reluctance to give competitors information that would convey a competitive advantage. Disclosure agreements might go some way toward addressing these concerns, but it might be difficult to anonymize data sufficiently well to allay them. For example, a researcher might eliminate proper names from a paper, but if he or she is known to have worked with a specific company in the past, readers might suspect that the anonymized details are from that company. Aggregating data from many individuals has helped to preserve anonymity in other research areas (e.g., in health research), but in some safety-related application domains there are so few participating organizations that this measure might prove insufficient. No general solution to this problem was raised during the workshop.

[15] For example, there are far fewer controlled flight into terrain (CFIT) accidents since commercial aircraft were equipped with enhanced ground proximity warning systems (EGPWS) [67].

[16] It isn't strictly necessary for human beings to be able to *predict* the exact behavior of a system containing software. If developers anticipate the possibility of a behavior and determine that the behavior is undesirable, the system can be engineered to preclude it.

8 Related Work

The issues of whether standards for software in safety critical systems work and how to improve them have been a topic of discussion in the relevant literature for many years (e.g., [64–66, 69]). There have been studies into the efficacy of such standards (e.g., [70]). There are papers in the relevant literature raising research questions about the role of standards in achieving safety and sketching studies to answer those questions (e.g., [18]). There are even standards for standards (e.g., [71, 72]).

As far as we know, however, AESSCS 2014 was the first workshop exclusively dedicated to the question of how to rigorously assess whether such standards work. Many of the ideas brought to the workshop will have been introduced elsewhere, perhaps by people other than the workshop participants. The value of the workshop—and of this paper—is in bringing those ideas together to begin to assemble a coherent set of research questions and to sketch studies that might address those questions.

9 Conclusions

The *Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software* workshop, held in conjunction with the European Dependable Computing Conference (EDCC) in May 2014, attracted 23 participants and 7 position papers. Participants discussed what it means for such standards to work, identified research questions related to such standards, proposed studies and experiments to address these, and identified threats to the validity of those studies and experiments and challenges to their success. Participants broadly agreed that the scientific literature lacks the well-founded empirical results that should underpin excisions about how to regulate safety. Some participants expressed a desire to proceed with some of the work discussed at the workshop.

One conclusion that can be drawn from the position papers and the workshop discussion is that there are several different definitions of what it might mean for a software safety standard to 'work.' The difference in scope between standards that cover safety at a system level (e.g., ISO 26262 [25]) and that focus almost exclusively on software correctness (e.g., RTCA DO-178C [3]) is, perhaps, well known. But participants also discussed definitions focused on bounding uncertainty in safety or correctness claims, addressing the risk from the most likely or consequential mistakes, promoting developer competence, and giving certifiers a tool with which to improve safety.

Another conclusion is that there might be several different approaches to assessing standards. For example, one might measure the effect of conforming to the standard (e.g., on dangerous failure rates or safety-related defect density). One might assess a standard by parts, identifying what each assurance requirement is meant to demonstrate and how well the techniques used to satisfy that requirement accomplish that goal. One might instead assess conditions thought to be necessary for the standard to work, reasoning that if such conditions are absent then the standard needs improvement. One might even focus on how well a standard addresses the problems of the past.

Perhaps as a result of these realities, participants identified many distinct research questions at many different levels of abstraction.

Participants sketched and discussed several proposals for research to address those research questions. While participants generally agreed that the results would be useful, participants also agreed that most of the proposed research would be very challenging to conduct.

Research into the efficacy of standards for software in safety critical systems could be challenging for several reasons. Experiments involving human subjects face the difficulty of recruiting a sufficient number of representative subjects. Students are easier to recruit, but results obtained from students might not generalize to practicing professionals. Practicing professionals are, almost by definition, busy people with many demands on their time. Historical studies are limited by the quality and quantity of the available data. Accident and incident reports might not contain the necessary detail, and successful safety engineering leads to accident and incident rates so low as to be difficult to draw statistically-valid conclusions from. And the most detailed and useful information is likely to be the property of for-profit software development corporations that might be reluctant to share it. Participants deemed the studies most likely to directly inform policy decisions—those that directly correlate conformance and risk—so difficult as to be unlikely to be successful. Nevertheless, an empirical assessment need not be perfect to be useful. For example, experiments using student volunteer subjects, while not definitive, might serve to distinguish standards that require revision from standards that are plausibly effective and thus worthy of a more definitive (and expensive) evaluation.

Workshop participants generally agreed that, given the importance of the topic, it would be worthwhile to go forward with some of the proposed studies. The proposals by Holloway and Johnson to assess whether safety assessors have the knowledge and skill necessary to assess safety cases were deemed particularly relevant given the current interest in safety cases as a basis for making acceptance decisions.

References

1. Leveson, N. G.; and Turner, C. S.: An Investigation of the Therac-25 Accidents. *Computer*, vol. 26, no. 7, July 1993, pp. 18–41. URL `http://dx.doi.org/10.1109/MC.1993.274940`.

2. ATSB: In-flight Upset 154 km West of Learmonth, WA, 7 October 2008, VH-QPA, Airbus A330-303. Transport Safety Report AO-2008-070, Australian Transport Safety Bureau, December 2011. URL http://www.atsb.gov.au/publications/investigation_reports/2008/aair/ao-2008-070.aspx.

3. RTCA DO-178C: *Software Considerations in Airborne Systems and Equipment Certification*. RTCA, Inc., Washington, DC, USA, December 2011. URL http://www.rtca.org/store_product.asp?prodid=803.

4. ED-12C, E.: *Software Considerations in Airborne Systems and Equipment Certification*. The European Organisation for Civil Aviation Equipment (EUROCAE), January 2012. URL https://www.eurocae.net/eshop/catalog/product_info.php?products_id=214.

5. RTCA DO-178B: *Software Considerations in Airborne Systems and Equipment Certification*. RTCA, Inc., Washington, DC, USA, December 1992. URL http://www.rtca.org/store_product.asp?prodid=581.

6. ISO 26262-6:2011: *Road vehicles — Functional safety — Part 6: Product development at the software level*. International Organization for Standardization, 2011. URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51362.

7. IEC 61508-3: *Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 3: Software requirements*. International Electrotechnical Commission, 2nd ed., April 2010. URL http://webstore.iec.ch/Webstore/webstore.nsf/Artnum_PK/43984.

8. Ashmore, R.: The Utility and Practicality of Quantifying Software Reliability. *Proceedings of Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software (AESSCS)*, May 2014. URL http://arxiv.org/abs/1404.7528.

9. Daniels, D.: The Efficacy of DO-178B. *Proceedings of Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software (AESSCS)*, May 2014. URL http://arxiv.org/abs/1404.6804.

10. Fusani, M.; and Lami, G.: On the Efficacy of Safety-related Software Standards. *Proceedings of Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software (AESSCS)*, May 2014. URL http://arxiv.org/abs/1404.6805.

11. Habli, I.; and Rae, A.: Formalism of Requirements for Safety-Critical Software: Where Does the Benefit Come From? *Proceedings of Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software (AESSCS)*, May 2014. URL http://arxiv.org/abs/1404.6802.

12. Holloway, C. M.; and Johnson, C. W.: Towards Assessing Necessary Competence: A Position Statement. *Proceedings of Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software (AESSCS)*, May 2014. URL http://arxiv.org/abs/1404.6803.

13. McDermid, J.: Nothing is Certain but Doubt and Tests. *Proceedings of Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software (AESSCS)*, May 2014. URL http://arxiv.org/abs/1404.6801, keynote.

14. Rushby, J.; Littlewood, B.; and Strigini, L.: Evaluating the Assessment of Software Fault-Freeness. *Proceedings of Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software (AESSCS)*, May 2014. URL http://arxiv.org/abs/1404.6844.

15. Wiels, V.: A Formal Experiment. *Proceedings of Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software (AESSCS)*, May 2014. URL http://arxiv.org/abs/1404.7542.

16. Leveson, N.: Re: [sc] Safety Cases. Post to Safety Critical Mailing List, 9 May 2012. URL http://www.cs.york.ac.uk/hise/safety-critical-archive/2012/0244.html.

17. Vigen, T.: Spurious Correlations. Web site: http://www.tylervigen.com/. Last accessed 2015-04-06.

18. McDermid, J. A.; and Rae, A. J.: How Did Systems Get So Safe Without Adequate Analysis Methods? *From the 9th International Conference on System Safety and Cyber Security*, Manchester, UK, October 2014. URL http://tv.theiet.org/technology/computing/40710.cfm.

19. Ferrell, T.: RE: [sc] Safety Cases. Post to Safety Critical Mailing List, 9 May 2012. URL http://www.cs.york.ac.uk/hise/safety-critical-archive/2012/0247.html.

20. Hayhurst, K. J.; Veerhusen, D. S.; Chilenski, J. J.; and Rierson, L. K.: A Practical Tutorial on Modified Condition / Decision Coverage. Technical Memorandum TM-2001-210876, NASA, Hampton, VA, May 2001. URL http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20010057789_2001090482.pdf.

21. Knight, J. C.; and Myers, E. A.: An improved inspection technique. *Communications of the ACM*, vol. 36, no. 11, 1993, pp. 51–61. URL `http://dx.doi.org/10.1145/163359.163366`.

22. Various authors: Re: [SystemSafety] Qualifying SW as "Proven in Use" [Measuring Software]. Posts on Bielefeld SystemSafety Mailing List, 1 July 2014. URL `http://www.systemsafetylist.org/date.htm#msg323`, various posts on or around this date.

23. A-P-T Research, Inc.: Safety Case Workshop. Electronic document, 14–15 January 2014. URL `http://www.apt-research.com/news/2014-01-15_SafetyCaseWorkshop/T-13-00600%20Safety%20Case%20Workshop%20Findings.pdf`.

24. McDermid, J.: Safety Cases: Purpose, Process and Prospects. Slide presentation to the Safety Case Workshop, 14–15 January 2014. URL `http://www.apt-research.com/news/2014-01-15_SafetyCaseWorkshop/04.ppsx`.

25. ISO 26262-1:2011: *Road vehicles — Functional safety — Part 1: Vocabulary*. International Organization for Standardization, 2011. URL `http://www.iso.org/iso/catalogue_detail?csnumber=43464`.

26. Rae, A. J.: Acceptable Residual Risk: Principles, Philosophy and Practicalities. *Proceedings of the 2nd IET System Safety Conference*, London, UK, 2007. URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4399904`.

27. ARP4754A: *Guidelines for Development of Civil Aircraft and Systems*. SAE, December 2010. URL `http://standards.sae.org/arp4754a/`.

28. ISO 26262-3:2011: *Road vehicles — Functional safety — Part 3: Concept phase*. International Organization for Standardization, 2011. URL `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51358`.

29. Lutz, R. R.: Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems. *Proc. of the IEEE International Symposium on Requirements Engineering (RE '93)*, San Diego, CA, January 1993. URL `http://dx.doi.org/10.1109/ISRE.1993.324825`.

30. MacKenzie, D. A.: *Mechanizing Proof: Computing, Risk, and Trust*. Inside Technology Series, MIT Press, Cambridge, MA, USA, new ed., March 2004. URL `http://mitpress.mit.edu/books/mechanizing-proof`.

31. Performance-based Operations Aviation Rulemaking Committee/Commercial Aviation Safety Team Flight Deck Automation

Working Group: Operational Use of Flight Path Management Systems. Technical report, Federal Aviation Administration, September 2013. URL http://www.faa.gov/about/office_org/headquarters_offices/avs/offices/afs/afs400/parc/parc_reco/media/2013/130908_PARC_FltDAWG_Final_Report_Recommendations.pdf.

32. Harris, M.: Will Nissan Beat Google and Uber to Self-Driving Taxis? *IEEE Spectrum*, 26 February 2015. URL http://spectrum.ieee.org/cars-that-think/transportation/self-driving/will-nissan-beat-google-and-uber-to-self-driving-taxis.

33. National Highway Traffic Safety Administration: Preliminary Statement of Policy Concerning Automated Vehicles. Electronic Document, 2013. URL http://www.nhtsa.gov/staticfiles/rulemaking/pdf/Automated_Vehicles_Policy.pdf.

34. Checkoway, S.; McCoy, D.; Kantor, B.; Anderson, D.; Shacham, H.; Savage, S.; Koscher, K.; Czeskis, A.; Roesner, F.; and Kohno, T.: Comprehensive Experimental Analyses of Automotive Attack Surfaces. *Proceedings of the 20th USENIX Security Symposium*, August 2011. URL http://www.autosec.org/pubs/cars-usenixsec2011.pdf.

35. ISO 26262-2:2011: *Road vehicles — Functional safety — Part 2: Management of functional safety*. International Organization for Standardization, 2011. URL http://www.iso.org/iso/catalogue_detail?csnumber=51356.

36. ISO 26262-4:2011: *Road vehicles — Functional safety — Part 4: Product development at the system level*. International Organization for Standardization, 2011. URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51359.

37. ISO 26262-6:2011: *Road vehicles — Functional safety — Part 5: Product development at the hardware level*. International Organization for Standardization, 2011. URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51360.

38. ISO 26262-7:2011: *Road vehicles — Functional safety — Part 7: Production and operation*. International Organization for Standardization, 2011. URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51363.

39. ISO 26262-8:2011: *Road vehicles — Functional safety — Part 8: Supporting processes*. International Organization for Standardization, 2011. URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51364.

40. ISO 26262-9:2011: *Road vehicles — Functional safety — Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses*. International Organization for Standardization, 2011. URL `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=51365`.

41. ISO 26262-10:2012: *Road vehicles — Functional safety — Part 10: Guideline on ISO 26262*. International Organization for Standardization, 2012. URL `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54591`.

42. IEC 61508-1: *Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 1: General requirements*. International Electrotechnical Commission, 2nd ed., April 2010.

43. ISO/IEC 13335-1:2004: *Information Technology — Security Techniques – Management of Information and Communications Technology Security — Part 1: Concepts and Models for Information and Communications Technology*. ISO/IEC, 2004. URL `http://www.iso.org/iso/catalogue_detail.htm?csnumber=39066`, withdrawn.

44. IEC/TR 62443-3: *Industrial Communication Networks — Network and System Security*. International Electrotechnical Commission (IEC), 2009. URL `http://webstore.iec.ch/webstore/webstore.nsf/ArtNum_PK/43216?OpenDocument`.

45. Steele, P.; and Knight, J.: Analysis of Critical System Certification. *Proceedings of the 15th International Symposium on High-Assurance Systems Engineering (HASE)*, Miami, FL, USA, January 2014, pp. 129–136. URL `http://dx.doi.org/10.1109/HASE.2014.26`.

46. Reason, J.: Human Error: Models and Management. *British Medical Journal*, vol. 320, no. 7237, March 2000, pp. 768–770. URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1117770/?tool=pubmed`.

47. FAA: Airborne Software Assurance. Advisory Circular 20-115C, Federal Aviation Administration, Washington, DC, USA, July 2013. URL `http://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_20-115C.pdf`.

48. Rae, D.: Metro North Railroad. Disastercast podcast, episode 48, February 2015. URL `http://disastercast.co.uk/episode-48-metro-north-railroad/`.

49. Federal Aviation Administration: Airworthiness Standards: Transport Category Airplanes—Equipment, Systems, and

Installations. Code of Federal Regulations Title 14, Section 25.1309, U.S. Department of Transportation, November 2007. URL http://www.ecfr.gov/cgi-bin/text-idx?SID=50773a43831f3c69f9ad03d378e1d96a&node=se14.1.25_11309&rgn=div8.

50. FAA: System Design and Analysis. Advisory Circular 25.1309-1A, Federal Aviation Administration, June 1988. URL http://rgl.faa.gov/Regulatory_and_Guidance_Library/rgAdvisoryCircular.nsf/0/50BFE03B65AF9EA3862569D100733174?OpenDocument.

51. IEC 61508-3: *Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 3: Software requirements*. International Electrotechnical Commission, December 1998. URL http://webstore.iec.ch/webstore/webstore.nsf/ArtNum_PK/23448.

52. European Committee for Electrotechnical Standardization (CENELEC): Railway applications — Communications, signalling and processing systems — Software for railway control and protection systems. British Standard EN 50128:2001, British Standards Institution, March 2001. URL http://shop.bsigroup.com/ProductDetail/?pid=000000000030228797.

53. Graydon, P. J.; and Kelly, T. P.: Using Argumentation to Evaluate Software Assurance Standards. *Information and Software Technology*, vol. 55, no. 9, September 2013, pp. 1551–1562. URL http://www.mrtc.mdh.se/index.php?choice=publications&id=3268.

54. Common Criteria for Information Technology Security Evaluation, Part 1: *Introduction and general model, Version 3.1, Release 3, Final*. Common Criteria, July 2009. URL http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R3.pdf, cCMB-2009-07-001.

55. Holloway, C. M.: Making the Implicit Explicit: Towards And Assurance Case for DO-178C. *Proceedings of the International System Safety Conference (ISSC)*, Boston, MA, USA, August 2013. URL http://www.cs.virginia.edu/~cmh7p/issc2013-178c-paper-final.pdf.

56. Defence Standard 00-56: *Safety Management Requirements for Defence Systems, Issue 4, Part 1: Requirements*. (U.K.) Ministry of Defence, June 2007. URL http://www.dstan.mod.uk/closure_notice.html.

57. Kuhn, T. S.: *The Structure of Scientific Revolutions*. The University of Chicago Press, Chicago, IL, USA, 3rd ed., 1996.

58. Greenwell, W. S.; Knight, J. C.; Holloway, C. M.; and Pease, J. J.: A taxonomy of fallacies in system safety arguments. *Proceedings of the 2006 International System Safety Conference (ISSC)*, Albuquerque, NM, USA, July 2006. URL `http://www.cs.virginia.edu/~cmh7p/paper-issc06-fallacies-as-printed.pdf`.

59. Haddon-Cave, C.: *The Nimrod Review: An Independent Review Into The Broader Issues Surrounding The Loss Of The RAF Nimrod MR2 Aircraft XV230 In Afghanistan In 2006*. The Stationery Office, London, October 2009. URL `https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/229037/1025.pdf`.

60. Strigini, L.; and Povyakalo, A.: Software Fault-Freeness and Reliability Predictions. *Proceedings of the 32nd International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, F. Bitsch, J. Guiochet, and M. Kaâniche, eds., Springer Berlin Heidelberg, vol. 8153 of *Lecture Notes in Computer Science*, 2013, pp. 106–117.

61. Graydon, P.; Habli, I.; Hawkins, R.; Kelly, T.; and Knight, J.: Arguing Conformance. *IEEE Software*, vol. 29, no. 3, May–June 2012, pp. 50–57. URL `http://dx.doi.org/10.1109/MS.2012.26`.

62. Graydon, P.; and Bate, I.: Realistic Safety Cases for the Timing of Systems. *The Computer Journal*, vol. 57, no. 5, May 2014, pp. 759–774. URL `http://dx.doi.org/10.1093/comjnl/bxt027`.

63. Biscoglio, I.; Coco, A.; Fusani, M.; Gnesi, S.; and Trentanni, G.: An Approach to Ambiguity Analysis in Safety-Related Standards. *Proceedings of the 7th International Conference on the Quality of Information and Communications Technology (QUATIC)*, September 2010, pp. 461–466. URL `http://dx.doi.org/10.1109/QUATIC.2010.83`.

64. Fenton, N. E.; and Neil, M.: A Strategy for Improving Safety Related Software Engineering Standards. *Transactions on Software Engineering*, vol. 24, no. 11, 1998, pp. 1002–1013. URL `http://dx.doi.org/10.1109/32.730547`.

65. Knight, J.: Safety Standards — A New Approach. *Proceedings of the 22nd Safety-Critical Systems Symposium (SSS)*, Brighton, UK, February 2014. URL `http://scsc.org.uk/paper_126/protect_reg_01-Knight.pdf?pap=933`, keynote.

66. Pfleeger, S. L.; Fenton, N.; and Page, S.: Evaluating software engineering standards. *Computer*, vol. 27, September 1994, pp. 71–79. URL `http://dx.doi.org/10.1109/2.312041`.

67. Honeywell: Just How Effective is EGPWS? Electronic white paper, 2006. URL https://www51.honeywell.com/aero/common/documents/EGPWS-Effectiveness.pdf.

68. Charette, R. N.: This Car Runs on Code. *IEEE Spectrum*, 1 February 2009. URL http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code.

69. Squair, M. J.: Issues in the Application of Software Safety Standards. *Proceedings of the 10th Australian Workshop on Safety-Related Programmable Systems (SCS)*, T. Cant, ed., no. 55 in CRPIT, ACS, Sydney, Australia, 2005, pp. 13–26. URL http://crpit.com/abstracts/CRPITV55Squair.html.

70. Hayhurst, K. J.: Framework for Small-Scale Experiments in Software Engineering: Guidance and Control Software Project: Software Engineering Case Study. Technical Report NASA/TM-1998-207666, NASA Langley Research Center, Hampton, VA, USA, May 1998. URL http://ntrs.nasa.gov/search.jsp?R=19980197315.

71. ISO Guide 7: *Guidelines for Drafting of Standards Suitable for Use for Conformity Assessment*. International Organization for Standardization, second ed., 1994. URL http://www.iso.org/iso/catalogue_detail?csnumber=23361.

72. ISO/IEC 17007:2009: *Conformity Assessment — Guidance for Drafting Normative Documents Suitable for Use for Conformity Assessment*. International Organization for Standardization, 2009. URL http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=42635.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 01- 09 - 2015	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)* 05/2014

4. TITLE AND SUBTITLE Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Graydon, Patrick J.; Holloway, C. Michael	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER 999182.02.50.07.02
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER** L-20575
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA-TM-2015-218804

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 03
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We need well-founded means of determining whether software is fit for use in safety-critical applications. While software in industries such as aviation has an excellent safety record, the fact that software flaws have contributed to deaths illustrates the need for justifiably high confidence in software. It is often argued that software is fit for safety-critical use because it conforms to a standard for software in safety-critical systems. But little is known about whether such standards ‘work.’ Reliance upon a standard without knowing whether it works is an experiment; without collecting data to assess the standard, this experiment is unplanned. This paper reports on a workshop intended to explore how standards could practicably be assessed. Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software (AESSCS) was held on 13 May 2014 in conjunction with the European Dependable Computing Conference (EDCC). We summarize and elaborate on the workshop's discussion of the topic, including both the presented positions and the dialogue that ensued.

15. SUBJECT TERMS

Efficacy; Empirical assessment; Safety-critical; Software; Standards

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER ***(Include area code)***
U	U	U	UU	47	(757) 864-9658