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

Patrick J. Graydon; NASA Langley Research Center; Hampton, Virginia, USA

C. Michael Holloway; NASA Langley Research Center; Hampton, Virginia, USA

Keywords: safety-critical software, standards, empirical assessment, safety, efficacy

Abstract

Safe use of software in safety-critical applications requires well-founded means of determining whether software is fit for such use. While software in industries such as aviation has a good safety record, little is known about whether standards for software in safety-critical applications 'work' (or even what that means). It is often (implicitly) argued that software is fit for safety-critical use because it conforms to an appropriate standard. Without knowing whether a standard works, such reliance is an experiment; without carefully collecting assessment data, that experiment is unplanned. To help plan the experiment, we organized a workshop to develop practical ideas for assessing software safety standards. In this paper, we relate and elaborate on the workshop discussion, which revealed subtle but important study design considerations and practical barriers to collecting appropriate historical data and recruiting appropriate experimental subjects. We discuss assessing standards as written and as applied, several candidate definitions for what it means for a standard to 'work,' and key assessment strategies and study techniques and the pros and cons of each. Finally, we conclude with thoughts about the kinds of research that will be required and how academia, industry, and regulators might collaborate to overcome the noted barriers.

Introduction

Safe use of software in safety-critical applications requires well-founded means of determining whether software is fit for such use. While software in industries such as aviation has an excellent safety record, the fact that software has contributed to deaths and injuries illustrates the importance of having *justifiably* high confidence in safety-critical software. Despite the danger of relying on software having properties that cannot be guaranteed, developers are building ever-more-complex software for safety-critical applications, sometimes for good reasons. For example, enhanced ground proximity warning systems (EGPWS)—far too complex to be implemented in simple hardware—save lives by preventing controlled flight into terrain (CFIT) accidents (ref. 1). Today, it is often (implicitly) argued that software is fit for use in a safety-critical application because it conforms to a standard such as RTCA DO-178C (ref. 2), ISO 26262 (ref. 3), or IEC 61508 (ref. 4). But little is known about whether such standards 'work,' or even what it means for them to work. To base decisions to accept the risk associated with using software on conformance with such a standard, without knowing whether that standard works, is to perform an experiment in which developers build software and only later find out if the systems incorporating it are safe. Moreover, unless developers, operators, and regulators carefully collect well-chosen data and use this to assess the standard, this experiment is *unplanned* and thus unlikely to support definitive conclusions about whether the standard works.

We organized the *Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety-Critical Software* (AESSCS 2014) workshop to bring together academia, industry, and regulators to develop practical ideas for assessing standards for software in safety-critical applications (ref. 5). In this paper, we relate and elaborate on the workshop discussion (including ideas that we do not necessarily endorse). Assessing standards will require the cooperation of academia, industry, and regulators. Toward achieving this, we contribute discussion and analysis of (1) what to assess (e.g., the standard as written, the standard as applied, or a commonly-used combination of safety evidence), (2) what it might mean for a standard to 'work,' (3) strategies for assessing standards, (4) study methods, and (5) suggestions for moving forward. We conclude with thoughts about the kinds of research that will be required to assess standards and how academia, industry, and regulators might collaborate to overcome obstacles such as those related to data confidentiality and the representativeness of experimental subjects.

Salient Characteristics of Standards for Software in Safety-Critical Applications

Standards for software in safety-critical applications vary but exhibit commonalities. Typical standards comprise a number of *assurance requirements*, most of which specify activities that developers must perform or qualities that

development artifacts must have. For example, *Software Considerations in Airborne Systems and Equipment Certification* (DO-178C) requires "reviews and analyses of high-level [software] requirements" to "ensure that the … functional, performance, and safety-related requirements of the system are satisfied by the high-level requirements" (ref. 2). Such assurance requirements are called *objectives* in DO-178C and are known by other names in other standards. Most standards for software in safety-critical applications are sometimes called *prescriptive* or *process-based* because of their many assurance requirements and because these are expressed in terms of a model software process and lifecycle. However, developers are typically permitted to meet the spirit of each assurance requirement in any appropriate manner. For example, DO-178C explicitly permits the use of alternative methods of meeting its assurance requirements as agreed with certifiers (ref. 2).

Software standards are frequently used as part of a larger approach to system safety assessment. For example, the US FAA advises that conformance with DO-178C is an acceptable means of showing compliance with applicable airworthiness regulations for the software aspects of airborne systems (ref. 6). This raises a question: are standards fit to be used for those purposes? That is, do they work? (We will explore more operational statements of this question in later sections.) As important as the decisions riding on conformance are, there is little compelling evidence of the *overall* efficacy of these standards. It is essential to know both whether standards work and why, lest seemingly innocuous changes in practice disrupt the mechanisms that have been providing safety (ref. 7).

Most standards' assurance requirements are only indirectly related to the property of interest (e.g., system safety). For example, no assurance requirement of DO-178C specifies a direct measure of software contributions to system safety (or even of software correctness); instead, each relates to assessable actions or properties such as having reviewed the software requirements or achieved specified structural test coverage. Thus, standards could be seen as implicitly embodying a *recipe* for indirect software safety assessment. Many distinct recipes might conform to a given standard, but there are a limited number of canonical recipes. The recipe that seems most common includes software requirements defined at multiple levels of software abstraction, reviews and analyses of these to show traceability and refinement, and software testing at the unit, integration, software, and system levels.

Standards are deliberately flexible with regard to development and assessment approach so as not to stifle progress. Some standards require the construction of a *safety case*, which is typically defined as a "a structured argument, supported by a body of evidence that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given environment" (ref. 8). While some standards require specific evidence to appear in a safety case (e.g., ref. 3), others offer fewer constraints (e.g., ref. 9). Regardless of whether conformance evidence includes a safety case, a report on a safety case, or a DO-178C-style Software Accomplishment Summary (ref. 2), flexibility raises the possibility that the efficacy of a standard will depend on the recipe used. Thus, the efficacy of each recipe might be at least as interesting as the efficacy of each standard.

Related Work: Attempts to Assess Standards for Software in Safety-Critical Applications

The issues of whether standards for software in safety-critical systems work and how to improve them have been a topic of discussion for many years (e.g., refs. 10–13). There have been studies into the efficacy of such standards (e.g., ref. 14) and blue-ribbon inquiries into the problems of dependable software and certification (e.g., ref. 15). There are papers raising research questions about the role of standards in achieving safety and sketching studies to answer those questions (e.g., ref. 16). There are papers about the efficacy of techniques that standards require or recommend—e.g., conformance to coding standards such as MISRA C (e.g., ref. 17)—though definitive evidence of efficacy is rarely available for any such technique (ref. 15). There are even standards for standards (e.g., ref. 18). As far as we know, however, AESSCS 2014 was the first workshop dedicated to the question of how to rigorously assess whether such standards work. Many ideas brought to the workshop have been stated elsewhere, sometimes by people other than the workshop participants. The value of the workshop is in using those ideas to assemble a coherent set of research questions and to sketch studies that might address those questions (e.g., refs. 7, 19–25).

Much of what has been said against standards comes down to complaints about the costs of conformance assessment and of mandated development and verification activities (e.g., ref. 15), complaints that standards focus too much on process and too little on broader issues of what the software does (e.g., ref. 15), and complaints that the efficacy of required and recommended techniques is largely unknown (e.g., ref. 15). Some have criticized a similar standard, the Common Criteria for Information Technology Security Evaluation, for having seemingly little impact on security metrics such as the number of known vulnerabilities (ref. 15).

Much of what has been said in support of standards comes down to examples of problems that must be addressed or observations that industries using a given standard have a good safety record. For example, one might observe that inadequate configuration management would undermine software verification and validation results and conclude that a good standard should require configuration management. But a standard might address each known necessary condition and yet not work, either because its assurance requirements are insufficiently effective or because a key condition remains unknown and unaddressed. One researcher claims that current standards-based approaches have "proven in some industries (such as aviation) to have lowered accident rates dramatically" (ref. 26). But the argument from history is unconvincing. First, correlation is not causation (ref. 27): conformance and safe operation might occur together even if the latter is caused by a third factor such as developers' care and attention (ref. 16). Second, the correlation has not been measured and would be hard to measure accurately: many applications do not offer large enough exposure to assess the frequency of very rare events, multiple versions of a standards might have been used over the years, and anecdotes about lenient assessors suggest differences in how standards are interpreted and applied. It is also possible that a standard might work better for a system of the past than a new system featuring a more complex processor, a more integrated and networked platform, more complex software functions, or more autonomy. The community needs better evidence that standards are fit to support safety-related decisions.

What it Means for a Standard for Software in Safety-Critical Applications to 'Work'

Assessing whether a standard works requires identifying what it means to work. Ideally, each standard would define this for itself. But while some standards include statements about the standard's aims or contributions, these are rarely testable hypotheses about the standards' effects on safety or correctness. Some are vague, others are belied by the body of the standard, and yet others sidestep important issues. For example, ISO 26262 promises "guidance to help avoid [risks due to the failure of electrical or electronic components in automobiles] by providing appropriate requirements and processes" (ref. 3). But what does it mean to "help avoid" risk? The statement does not promise a result that will satisfy any of the well-known theories of residual risk acceptability (ref. 28). It is perhaps unfair to expect that it would: ISO 26262 is an industry consensus document meant to apply across a diverse supply chain. Nevertheless, to determine whether a standard works, researchers need a testable definition of "work." Participants in AESSCS 2014 identified (a) a crucial distinction between the goals of correctness and system safety, (b) several definitions of what it means for a standard to achieve one of those goals, and (c) crosscutting concerns such as cost effectiveness and applicability. 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. It remains to be seen which definition(s) will yield practical assessments that are sufficient grounds for making rational decisions about whether a given system is acceptably safe or for setting certification policy.

Safety Versus Correctness: Some standards for software in safety-critical applications are focused on correctness rather than safety: their assurance requirements are mainly about whether the software behaves as specified, not how that behavior affects system safety. For example, DO-178C (ref. 2) assumes software requirements derived from a system-level safety analysis such as might be performed in accordance with SAE ARP4754A (ref. 29). The standard's assurance requirements aim to show that the delivered software meets its requirements, but safety-related requirements are not singled out for special care. DO-178C specifies software development assurance levels in terms of the consequences of software failure but excludes all safety analyses from its scope. Other standards define and require safety analyses. For example, like DO-178C, Part 6 of ISO 26262 contains many assurance requirements related to verifying software correctness (ref. 3). But unlike DO-178C, ISO 26262 gives guidance for safety assessment of systems containing software: Part 3 defines a hazard and risk assessment process. This guidance is limited in scope to the safety impact of the failure of individual electrical and electronic 'items'. The standard's assurance requirements would apply to the design and implementation of an electronic stability control system *if* the manufacturer fits one. But the decision of whether to fit such a system might have greater impact on safety.

Correctness is not a sufficient condition for safety. Safety might depend more on decisions about *what* to build: there is some evidence to suggest requirements defects are a bigger problem than implementation defects (e.g., refs. 30 and 31). Automation, human factors, and security are all key considerations. The choices of what to automate and how to automate it are important concerns in aviation (ref. 32), and trends towards assisted and autonomous driving suggest that automation is a growing concern in the automotive domain (ref. 33). Airborne software might meet its requirements yet create challenges for aircraft maintainers, leading to a higher risk of maintenance error—and thus of an accident—than an alternative design. Designers might cut costs by implementing avionics functions

in a thematically unrelated unit with surplus processing power, causing flight crews to struggle to understand how the error messages they receive relate to the symptoms they experience. Building automobile radios with speed-sensitive volume controls increases passenger comfort but requires a data link between the radio and the relevant engine control unit(s) that might be a vector for deliberate attack on a vehicle's lights or brakes (ref. 34). While safety standards traditionally exclude security concerns, the potential for deliberate attack to compromise safety cannot be ignored; a complete approach to safety must weigh the impact of design choices that affect both. But—partly to avoid programmers having to be safety experts—most standards for software in safety-critical applications focus more on correctness than on safety. Accordingly, (a) some definitions of 'works' must be considered in both safety and correctness forms and (b) it might be useful to study the safety impact of combinations of standards.

Bounding Uncertainty in Safety or Correctness 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, 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" (ref. 2). 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). Instead, the standard is meant to increase the likelihood of spotting any mistake that would compromise system safety before systems are deployed. This definition of works is related to the *Filter Model* of safety-critical system certification, in which certification comprises a *filter* and a *decision procedure* (ref. 35). Each assurance requirement is a filter used to identify faults in a system. As in the *Swiss Cheese* accident model (ref. 36), each filter might catch defects that other filters in the same standard miss. Collectively, the filters allow certifiers to judge "whether faults remain that would subject the public to an unacceptable level of operational risk" (ref. 35).

The bounding uncertainty definition of what it means for a software safety standard to 'work' has the virtue of being the most relevant to decisions about whether to deploy, continue operating, or improve software in safety-critical applications. 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. But this definition captures what many speakers mean when they say that a standard 'works' and thus might help to define the meaning of an assessment against a different definition.

Addressing the Risk of Specific Mistakes: A standard for software in safety-critical applications might be said to work if its use makes developers less likely to make (or more likely to catch) a set of mistakes of interest. This set might be drawn from research or lessons learned from accidents and incidents. Ideally, the set would include the mistakes that give rise the greatest risk because they occur frequently or lead to the most severe accidents.

This definition differs from the previous one in that its focus is limited to specific contributions to uncertainty. To the degree that a system is like prior systems, addressing the ways in which those systems went wrong should make the system safer. This definition has the virtue of focusing attention on factors that are known to matter. However, addressing only such factors might not lead to safety where the application or development techniques are novel. For example, prohibiting most backward branches might have reduced the incidence of control-flow problems in assembly language programs, but such a rule is not applicable to software written in most high-level languages.

Promoting Developer Competence: A standard for software in safety-critical applications might be said to work because it promotes developer competence, either by teaching good practice or acting as a barrier to the entry of incompetent developers or organizations. While we would not recommend that developers learn to build critical systems *solely* by reading standards, a standard might (in part) have a pedagogical effect. Developers read the standard, come to understand its assurance requirements, and seek information about the recommended means of satisfying these. In so doing, they might gain knowledge and develop habits that would improve their systems even if conformance were (unexpectedly) never assessed. A standard might instead preclude the development of safety-critical systems by people who lack the requisite skills and experience or organizations that do not adopt best practices. That is, the standard—or perhaps its compliance enforcement regime—might pose a difficult-to-surmount barrier to people whose work might yield unacceptable risk without unduly impeding those who would produce acceptably safe systems with or without the standard. Several AESSCS participants expressed the related opinion that if you took the standards away from the best teams, they would carry on building acceptably safe software.

Giving Certifiers a Tool for Improving Safety: A standard for software in safety-critical applications might be said to work because it gives certifiers a tool for improving safety or blocking the deployment of unsafe systems. In this

view, a standard is simply the necessary foundation of a compliance assessment regime. Developers 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. This motivation might encourage developers to follow better practices than they otherwise would. 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, a standard's assurance requirements need not be unambiguous or perfectly bound confidence in safety, filter out unsafe systems, or teach best practice: if assessors can determine whether a given system is acceptably safe or whether an alternative would be safer, the leverage afforded by the certification regime could allow them to enforce or improve safety.

Cost-Effectiveness: The definitions of 'work' given above relate to standards' effects on safety or correctness. But standards also impact the cost of development and certification. Decisions about whether to use a standard, which standard to use, and what assurance requirements to include in a standard might turn on whether the standard is *cost-effective*. Decisions about whether a safety improvement is worth its cost are unavoidably political. Nevertheless, a well-planned scientific assessment of standards might produce data that could inform such determinations. For example, consider a study that compares the effect of two standards on confidence in software correctness. If the study design can answer the question *Which works better?* but not *By how much?*, the results would lack information needed to make a decision based on cost-effectiveness.

Applicability: Questions of whether a standard works need to be asked and answered with respect to a scope of systems of interest. A standard that is applicable to some systems might be inapplicable to others, and this is not merely a matter of domain. For example, DO-178C's assurance levels are defined in terms of software's effect on the aircraft and its occupants (ref. 2). But the destruction of a remotely piloted or autonomous aircraft might be far less important than its interference with an aircraft carrying crew or passengers. A standard providing assurance that the aircraft does not collide with people or vehicles carrying people might be more cost-effective for such aircraft.

The Standard's Text, its Application, or its Recipe(s): What Is the Target of Evaluation?

The power of a study to answer specific questions related to standards depends (in part) on the target of evaluation: does the study assess the standard as written, the standard as applied, or a recipe that can be used with the standard? Each choice of target has advantages and disadvantages; none is a perfect choice for all assessment purposes.

The Standard As Written: One could assess the text of a standard as written. For example, one could review the text for ambiguity in its assurance requirements or to see whether there are assurance requirements aimed at eliminating or catching a given kind of error. One could even couple what is known about the efficacy of safety and development techniques with analysis of the text to analyze where claims made on the basis of conformance might be weak. For example, one could identify a claim that standards conformance was used to support, extract the argument implicit in the standard's assurance requirements and internal commentary, and critically examine that argument for gaps and flaws, some of which correspond to ways in which the standard could be improved (ref. 37).

There are at least three advantages of examining the standard as written. First, the text of the standard is available to everyone who can afford it. Unlike a study of records of a system's history, there are no intellectual property or privacy concerns that would limit access to the standard's text. Second, it is easier to study a single, fixed thing. Unlike a study of a standard as applied, the study procedure need not account for variance in how the standard is interpreted. Third, many study designs are simpler than studies of the standard as applied. For example, review of a standard's text does not require extensive data collection and can be done by a single researcher working alone.

There are at least two disadvantages of examining the standard as written. First, the text might be non-specific, vague, or (mis-)interpreted to serve political purposes, with the effect that conformant practice might be both highly variable and different from what a reader of the text might envision. Second, relationship between an assurance requirement's text and its effect on safety or correctness might be known poorly or not at all. If it is not possible to predict practice from the standard's text, analysis of the text might not predict whether the standard works when applied in all intended or permitted contexts. We note that it might not be possible to eliminate variability in practice: many standards deliberately allow developers to use better techniques than those imagined by the writers where appropriate, and some standards are intended for a broad range of applications. For example, one safety standard applies to defense systems as diverse as soldiers' personal equipment and aircraft carriers (ref. 8.)

The Standard As Applied: One could assess a standard as applied. For example, one could review bug reports, incident reports, and accident reports to determine whether conformance with a standard at a higher integrity level corresponds to lower prevalence of dangerous faults (ref. 23). One could compare records for systems built to different standards to see if one standard is associated with a lower rate of a specific mistake of interest than another.

The obvious benefits of assessing a standard as applied are that the assessment results can account for variations in how the standard is interpreted and for any mechanism by which its text might affect safety or correctness. For example, if defect rates are studied across a representative sample of software development efforts that conforms to DO-178C (which might be difficult in practice), the results should generalize to other applications for which that standard is appropriate. If developers' office layout (e.g., offices vs. cubes) has a significant effect on the number and kind of defects introduced during coding, that effect might be visible in study data as a variance in defect rates that cannot be explained in terms of the independent variables identified in the study design.

The obvious disadvantages of assessing a standard as applied are difficulty, the possibility of noisy data, and the possibility that the studied population might differ from the population of interest. Standards are applied mainly by private commercial firms that might not consent to release relevant data, for instance out of concern that doing so might allow competitors to duplicate their success or press a legal claim. Variability in how standards are applied might lead to noise in measured values that makes it difficult to draw conclusions. And there is the possibility that, as software development and assessment methods evolve and applications change, the population documented in the records might differ from the population of interest in a way that makes study conclusions inapplicable.

The Standards' Recipes: One could instead assess the recipes used to conform to standards. For example, one could determine the distribution of dangerous defects discovered in software developed using review to show requirements correctness, source code developed in SPARK (Ada) using a correct-by-construction approach (ref. 38), an RTOS with fixed time partitioning, and testing that achieves Modified Condition/Decision Coverage (MC/DC, ref. 39).

The primary advantage of assessing the recipe rather than a standard is that the results generalize to some degree to any context in which the recipe can be used, including developments that conform to goal-based standards (e.g., ref. 9). The disadvantage is that, unlike assessment of the standard as applied, it will be unclear whether results reflect any effect from the standard or the certification process other than limits on the recipe employed. For example, suppose that the knowledge that third-party certifiers will audit their work prompts development organizations to use better practices than they would if conformance is assessed in-house. If that is true, researchers studying examples of the recipe in a third-party-certification context might find better outcomes than developers following the same recipe in an in-house-conformance-assessment context could reasonably expect to achieve.

Strategies for Assessing Standards for Software in Safety-Critical Applications

After identifying a hypothesis and a target of evaluation, researchers must choose an assessment strategy. In this section, we discuss four strategies. Again, none is a silver bullet: each has advantages and disadvantages.

Measuring the Effect: One might assess a standard using a metric such as accident or incident rate or number of defects (total and safety-related) discovered. For example, one might compare the number of safety-relevant defects found in software post deployment across software developed in conformance with different standards.

The advantages of this direct approach are simplicity and the elimination of the need to understand which factors impact overall effect and by how much. But there are at least three disadvantages. First, accidents and incidents might occur too infrequently to yield statistically significant results. (In transport aircraft, for example, developers aim to make certain accidents unlikely over the operating lifespan of all systems of that type.) Second, correlation is not causation: some unappreciated difference between the application domains in which two standards are used, rather than the difference between the standards, might be the actual cause of observed differences. Third, system developers and operators might be reluctant to disclose the relevant data for reasons discussed above.

Assessment by Parts: One might assess a standard or recipe by separately assessing the effectiveness of each of its assurance requirements or ingredients and then inferring an assessment of efficacy of the whole. For example, one might capture and then assess a standard's implicit argument as described above (ref. 37). Evaluating a recipe by parts would studies to assess the efficacy of each technique, e.g. the effect of formalizing requirements (ref. 21).

The advantage of this approach is that the efficacy of individual techniques might be shown by experiment, which is capable of demonstrating causality and justifying great confidence. But there are at least three disadvantages. First, it might be difficult to identify (or obtain agreement on) the standard's argument. SC-205—the committee charged with producing DO-178C (ref. 2)—produced a rationale document (ref. 40) with brief discussions of the reasons behind some specific objectives, but this rationale is not an overall argument. Second, few such experiments have already been done. For example, there are differing opinions about the efficacy of testing that achieves MC/DC and little empirical data in the public domain. Third, the value of the whole might not depend *solely* on the individual values of its parts (ref. 39). For example, the efficacy of a technique might depend in part on whether it is deployed in conjunction with specific other techniques. The needed studies might be *very* specific.

Testing Necessary Conditions: One might assess a standard by identifying necessary conditions or ways in which it might *not* work and assessing each. For example, one might suppose that unambiguous assurance requirements are a prerequisite for a standard to work and survey assessors to determine whether they interpret objectives identically.

Several AESSCS participants proposed testing necessary conditions. McDermid proposed finding out if standards work better at higher integrity levels (ref. 23). Fusani and Lami questioned whether conformance can be assessed practically (ref. 20). Holloway and Johnson proposed investigating whether safety assessors have the skills and education necessary to assess the safety arguments required by some standards (ref. 22). Ashmore, Rushby, Littlewood, and Strigini explore conditions necessary for assessing quantified software reliability (refs. 19 and 24). Wiels proposed an *abstraction view* model of certification that defines necessary conditions that might be evaluated in this way (ref. 25). In the model, certification is analogous to verifying software by verifying a formal abstraction of it. If conformance is to imply correctness, the standard's abstract view must be sound, tractable, and precise. That is, the abstraction must "defin[e] more behaviors than the actual behaviors of the program," conformance assessment must be practical, and nonconformities should correspond to ways in which the software is incorrect.

The advantage of this strategy is that it might be easier to assess a specific condition than to assess a standard's overall effect. For example, configuration consistency could be assessed through auditing. The disadvantage of this strategy is that it can reveal problems but not show efficacy: the unsatisfied necessary condition might be unknown.

Solving the Problems of the Past: One might assess a standard by determining how well it solves past problems. For example, suppose that several systems in an application domain overrun response time deadlines and that a new version of the relevant standard includes a new assurance requirements meant to improve confidence that deadlines will be met. One 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 strategy has the advantage of focusing on factors known to be important. (Some things that remain unknown might not matter in practice.) However, it has at least two disadvantages. First, it is difficult to determine whether a standard introduces new problems as side effects. Second, as with the other strategies, software might begin to fail in ways not seen in the past as new applications are tackled and new development techniques introduced.

Research Methods

Once researchers have identified a hypothesis, target of evaluation, and assessment strategy, they must choose an appropriate research method. Yet again, none is a silver bullet: each has advantages and disadvantages.

Historical Study: One might assess a standard using historical data. For example, researchers might use defect logs to measure the correctness of software developed in conformance to a standard. Researchers might count accidents and incidents as a measure of the system safety impact of software developed in conformance to a standard.

The advantage of historical study is that there is more data about software that has been used for years than about software that has never been fielded. Although some events might still be too rare to draw significant conclusions about, history offers far more exposure than can be obtained during pre-release testing. Historical study has at least two disadvantages. First, the historical record might be insufficiently accurate. For example, if one development organization records all identified defects while another records only defects discovered after unit testing, their defect counts might not be directly comparable. Second, much of the requisite data might not be publicly available.

Survey: One might assess a standard using a survey. For example, researchers could survey assessors to determine how broadly a given assurance requirement has been interpreted.

The advantages of survey research are that (1) it is not constrained to available data like historical study, (2) it is generally easier to plan and conduct than experimental research, and (3) it can reveal important observations from people who work with standards professionally. Surveys have at least three disadvantages. First, it can be difficult to obtain a sufficiently high response rate from a representative sample of the target population. The responses of a few assessors who decide to respond to a survey might not reflect the opinions of all assessors. Second, it is difficult to craft unbiased survey questions. The way a question is asked can easily skew responses. Third, surveys can only answer questions about what people know or think, which excludes many important questions. For example, opinions about the efficacy of achieving MC/DC might not reflect the actual efficacy of achieving MC/DC.

Experiments with Human Subjects: One might assess a standard using experiments featuring human subjects. For example, one could assess the efficacy of source code inspections (and how the inspection process affects this) by determining how many seeded defects subjects are able to locate (ref. 41).

The advantage of experiments with human subjects is that they can answer questions about human performance—and thus about the efficacy of many development and analysis techniques—more definitively than historical studies because they can be planned so as to control for confounding factors. Experiments have at least three disadvantages. First, it is difficult to recruit a large, representative sample of volunteer subjects. Easily recruited subjects such as students might not represent the population of interest, and practicing professionals are difficult to recruit. Second, the results might be difficult to generalize beyond the specimen problem or system used in the experiment. For example, skeptics might claim the results do not apply to their domain of interest because source code written for those applications has features not present in the code specimen used in the experiment. Third, the scope and scale of specimen artifacts and study process is typically limited by the need to conserve volunteers' time. For example, it would be difficult to recruit subjects willing to volunteer enough time to create a full-scale safety argument.

Towards Assessment of a Standard for Software in Safety-Critical Applications: Some Suggestions

The workshop discussion and subsequent thinking prompted us to make several suggestions for assessing standards.

Provide Authoritative Definitions of 'Works': The assessment of standards is complicated at the outset by the need for (third-party) researchers to identify research questions with answers that will support the decisions made about standards or based on conformance. If it is possible for standards committees—which ought to include some of the people making those decisions—to define what it means for their standards to work, they should. Where the definitions might vary with the circumstances of how standards are used, regulators who accept conformance for some purpose could provide their own definitions.

Identify Data that Is Useful, Data that Is or Could Be Available, and Acceptable Anonymization: Historical studies are difficult in part because it is hard to identify relevant, available data. The AESSCS workshop was a first step towards addressing this difficulty, but securing access to needed data will require collaboration between the researchers conducting the studies and the development organizations that have or must produce it. Researchers could characterize the data that would be useful so that developers know what might be of interest and can plan data collection. Industry could identify the data it has or could collect and the conditions under which it would be willing to release that data to researchers. Data aggregation and anonymization techniques might be useful ways to address industry's privacy and intellectual property concerns. It might help to have a trusted broker that could be given data to aggregate, anonymize, and distribute to researchers in accordance with disclosure agreements.

Identify an Available Pool of Suitable Human Subjects: One of the chief difficulties of experimental research is recruiting suitable volunteers. Since research in fields such as psychology involves experiments on human subjects, many universities organize "human subjects pools" and student experimenters typically volunteer as subjects in their peers' experiments. If industrial developers of software for safety-critical applications value the results that could be obtained through experiment, a similar arrangement might be possible. Just as professionals donate time to review technical manuscripts, development organizations might permit employees to donate time as study subjects. A central web site or mailing list might help match volunteers to studies, possibly with less bias than other methods.

Conclusions

The position papers submitted to *Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety-Critical Software* (AESSCS 2014) and the resulting discussion revealed both important research challenges and subtleties that affect research design. Practical assessment of standards is going to require collaboration between academia, industry, and regulators to define research hypotheses, identify available data, plan future data collection, and recruit suitable study participants.

Based on the position papers, the workshop discussion, and subsequent thinking, we have identified and discussed (a) several candidate definitions of what it means for a standard to work, (b) three potential targets of evaluation, (c) four assessment strategies, and (d) three kinds of research methods. One key observation is that there is no widely agreed upon definition of what it means for a standard to work. Another is that no single assessment strategy or research method is perfect: each has unique advantages and disadvantages. Building robust evidence that a given standard 'works' will require several types of evidence addressing a mix of targets of evaluation. The situation is analogous to software verification by testing: developers perform a mix of unit testing, software integration testing, software testing, and hardware-software integration testing because each addresses a weakness in the others.

'Further research is needed' is a cliché. Here, it is true. But performing that research will require first defining the right research questions and then finding ways to provide the resources needed. Finding a practical way to gather useful research data while addressing the reasons industry is reluctant to disclose such data is one key part of that. Finding a way to recruit appropriate experimental subjects is another. We hope that by identifying these challenges and proposing concrete research ideas, AESSCS 2014 has moved the community closer to practical evaluation of the standards for software in safety-critical applications that we all rely on.

Acknowledgment

We thank the authors and participants of AESSCS 2014 for their contributions, the EDCC 2014 organizers for their support, and Rob Ashmore, John Knight, John McDermid, and Andrew Rae for the discussions that led to this paper.

References

1. Honeywell. *Just How Effective is EGPWS?* Electronic white paper, 2006.
2. RTCA DO-178C. *Software considerations in airborne systems and equipment certification*. RTCA, Inc., 2011.
3. ISO 26262:2011. *Road vehicles — Functional safety*. International Organization for Standardization, ISO, 2011.
4. IEC 61508. *Functional safety of electrical/electronic/programmable electronic safety-related systems*. 2nd ed. International Electrotechnical Commission, 2010.
5. Graydon, P. J. and C. M. Holloway. *Planning the unplanned experiment: Assessing the efficacy of standards for safety critical software*. Technical Report NASA/TM-2015-**TODO**. 2015.
6. Federal Aviation Administration. Advisory Circular 20-115C. 2013.
7. Daniels, D. "The efficacy of DO-178B." *Proc. AESSCS*, May 2014.
8. Interim Defence Standard 00-56, Issue 5 (Amd 1). *Safety management requirements for defence systems*. Ministry of Defence, Glasgow, UK, 2014.
9. Interim Defence Standard 00-55, Issue 3. *Requirements for safety of programmable elements (PE) in defence systems*. Ministry of Defence, Glasgow, UK, 2014.
10. Fenton, N. E., and M. Neil. "A strategy for improving safety related software engineering standards." *Transactions on Software Engineering* 24, no. 11 (1998): 1002–1013.
11. Knight, J. "Safety standards — A new approach." *Proc. Safety-Critical Systems Symposium (SSS)*. Keynote. Brighton, UK, 2014.
12. Pfleeger, S., N. Fenton, and S. Page. "Evaluating software engineering standards." *Computer* 27 (1994), 71–79.
13. Squair, M. J. Issues in the application of software safety standards. *Proc. Australian Workshop on Safety-Related Programmable Systems (SCS)*. Sydney, Australia, 2005.
14. Hayhurst, K. J. *Framework for small-scale experiments in software engineering: Guidance and control software project: Software engineering case study.* Technical Memorandum TM-1998-207666. Hampton, VA, USA: NASA Langley Research Center, May 1998.
15. Jackson, D., M. Thomas, and L. I. Miller, eds. *Software for dependable systems: Sufficient evidence?* The National Academies Press, Washington, DC, USA, 2007.

16. McDermid, J. A. and A. J. Rae. "How did systems get so safe without adequate analysis methods?" *Proc. Int'l Conf. on System Safety and Cyber Security*. Manchester, UK, October 2014.
17. Boogerd, C. and L. Moonen. "Assessing the value of coding standards: An empirical study." *Proc. IEEE Int'l Conf. on Software Maintenance (ICSM)*. October 2008.
18. ISO/IEC 17007:2009. *Conformity Assessment — Guidance for Drafting Normative Documents Suitable for Use for Conformity Assessment*. International Organization for Standardization, 2009.
19. Ashmore, R. "The utility and practicality of quantifying software reliability." *Proc. AESSCS*. May 2014.
20. Fusani, M. and G. Lami. "On the efficacy of safety-related software standards." *Proc. AESSCS*. May 2014.
21. Habli, I. and A. Rae. "Formalism of requirement for safety-critical software: Where does the benefit come from?" *Proc. AESSCS*. May 2014.
22. Holloway, C. M. and C. W. Johnson. "Towards assessing necessary competence: A position statement." *Proc. AESSCS*. May 2014.
23. McDermid, J. "Nothing is certain but doubt and tests." *Proc. AESSCS*. Keynote. May 2014.
24. Rushby, J., B. Littlewood, and L. Strigini. "Evaluating the assessment of software fault-freeness." *Proc. AESSCS*. May 2014.
25. Wiels, V. "A formal experiment." *Proc. AESSCS.* May 2014.
26. Leveson, N. "Re: [sc] Safety Cases." Post to University of York Safety Critical Mailing List, 9 May 2012. http://www.cs.york.ac.uk/hise/safety-critical-archive/2012/0244.html
27. Vigen, T. Spurious Correlations. Web site: www.tylervigen.com. Last accessed 12 May 2015.
28. Rae, A. J. "Acceptable residual risk: Principles, philosophy and practicalities." *Proc. IET System Safety Conference*. London, UK, 2007.
29. SAE ARP4754A. *Guidelines for development of civil aircraft and systems*. SAE, December 2010.
30. Lutz, R. R. "Analyzing software requirements errors in safety-critical, embedded systems." *Proc. IEEE Int'l Symp. on Requirements Engineering (RE)*. San Diego, CA, USA, January 1993.
31. MacKenzie, D. *Mechanizing proof: Computing, risk, and trust*. MIT Press, Cambridge, MA, USA, 2004.
32. Performance-based operations Aviation Rulemaking Committee/Commercial Aviation Safety Team Flight Deck Automation Working Group. *Operational use of flight path management systems*. FAA, 2013.
33. NHTSA. *Preliminary statement of policy concerning automated vehicles*. Electronic document, 2013.
34. Checkoway, S., D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Kosher, A. Czeskis, F. Roesner, and T. Kohno. "Comprehensive experimental analyses of automotive attack surfaces." *Proc. USENIX Security Symp*. August, 2011.
35. Steele, P., and J. Knight. "Analysis of critical system certification." *Proc. Int'l Symp. on High-Assurance Systems Engineering (HASE)*. Miami, FL, USA, January 2014.
36. Reason, J. "Human error: Models and management." *British Medical Journal* 320, no. 7237 (2000): 768–770.
37. Graydon, P. J. and T. P. Kelly. "Using argumentation to evaluate software assurance standards." *Information and Software Technology* 55 no. 9 (2013): 1551–1562.
38. Barnes, J. *High integrity software: The SPARK approach to safety and security*. Addison-Wesley, 2003.
39. Hayhurst, K. J., D. S. Veerhusen, J. J. Chilenski, and L. K. Rierson. *A practical tutorial on modified condition / decision coverage*. Technical Memorandum TM-2001-210876. NASA Langley Research Center, May 2001.
40. RTCA DO-248C. *Supporting information for DO-178C and DO-278A*. RTCA, Inc., 2011.
41. Knight, J. and A. Myers. "An improved inspection technique." *Comm. of the ACM* 36, no. 11 (1993):51–61.

Biography

Patrick J. Graydon, Research Computer Scientist, NASA Langley Research Center, 100 NASA Road, Hampton VA 23681-2199, telephone – (757) 864-2003, facsimile – (757) 864-4234, e-mail – patrick.j.graydon@nasa.gov.

Patrick J. Graydon is a research computer scientist at NASA Langley Research Center. His professional interests are safety and security argumentation, dependable software engineering, and certification.

C. Michael Holloway, Senior Research Engineer, NASA Langley Research Center, 100 NASA Road, Hampton VA 23681-2199, telephone – (757) 864-1701, facsimile – (757) 864-4234, e-mail – c.m.holloway@nasa.gov.

C. Michael Holloway is a senior research computer engineer at NASA Langley Research Center. His primary professional interests concern epistemic issues influencing the level of confidence that may justifiably be placed in the safety of software-intensive systems.