Evaluating Software Safety Standards: A Systematic Review and Comparison

W. Eric Wong, Tej Gidvani, Alfonso Lopez, Ruizhi Gao
Department of Computer Science
University of Texas at Dallas
{ewong, gxr116020}@utdallas.edu

Matthew Horn
Department of Computer Science
Muhlenberg College

Abstract—Software safety standards are commonly used to guide the development of safety-critical software systems. However, given the existence of multiple competing standards, it is critical to select the most appropriate one for a given project. We have developed a set of 15 criteria to evaluate each standard in terms of its usage, strengths, and limitations. Five standards are studied, including a NASA Software Safety Standard, an FAA System Safety Handbook, MIL-STD-882D (US Department of Defense), DEF-STAN 00-56 (UK Ministry of Defense), and DO-178B (Commercial avionics). Results of our evaluation suggest that different standards score differently with respect to each evaluation criterion. No standard performs better than others on all the criteria. The lessons learned from software-related accidents in which the standards were involved provide further insights on the pros and cons of using each standard.

Keywords−software safety; system safety; safety standard; safety-critical software; hazards; mishap

1. Introduction

Areas crucial to life such as medicine, transportation, nuclear-energy, aeronautics, and communications all use software in one way or another. An intensive application of software to these domains also implies that the software has become even more safety-critical such that an error in the software or an error in its use could have devastating consequences, including significant financial loss, property damage, or even human casualties [23,24]. More challengingly, today's software systems are much larger and more complicated than ever before, and at the same time the pressure to produce safe and dependable software at a reduced cost keeps increasing. As a result, we urgently need good practices which help us not only achieve high quality but also reduce the development and maintenance cost. Without other solid guidance, many projects take an approach by appealing to authority and adopt certain safety standards to guide their development process. The rationale is that if one software system developed following a safety standard could meet all the expectations (functional and safety requirements, as well as budget constraint), a similar successful story could also be reproduced for another software system.

Indeed, to build software systems with high safety requirements, it is important to approach the process in a certain way to maintain efficiency and ensure with a high degree of confidence that the requirements are met effectively. However, given the existence of multiple competing safety standards, it is critical to select the one that is most appropriate for a given project. We have developed a set of 15 criteria (as detailed in Section 2) to evaluate each standard in terms of usage, strengths, and limitations. In this paper, five popular safety standards are selected including a NASA Software Safety Standard − NASA STD 8719.13B [18] (hereafter, simply refer to as NASA STD), an FAA (Federal Administration Aviation) System Safety Handbook [12] (refer to as FAA Handbook), two Military Standards: MIL-STD-882D [9] used by the U.S. Department of Defense and DEF-STAN 00-56 [17] used by the Ministry of Defense of the United Kingdom, as well as DO-178B [22] − a standard widely used for commercial avionics.

Results of our evaluation (see Section 3) suggest that different standards score differently with respect to each evaluation criterion and no standard is superior to others on all the criteria. As a result, each project should carefully select a standard that best matches its environments and meets the level of safety requirements it has to achieve.

The lessons learned (see Section 4) from software-related accidents in which the above standards were involved provide further insights on the pros and cons of using each standard. Our conclusion and future work appear in Section 5.

2. Fifteen evaluation criteria

We present a set of 15 criteria as described below to evaluate a software safety standard from different perspectives.

C1) Does the standard discuss splitting software into safety-critical components?

Software which resides within a larger system cannot be evaluated without looking at the system as a whole. In this criterion we are looking for preliminary hazard analysis of both the system and the software that contributes to the hazards found. The standard should require system level hazard analysis, prior to undertaking software analysis, to identify any subsystems that are safety-critical. Next, the standard should discuss the identification of software that either contributes to or avoids/mitigates a hazard. Then end result should be the identification and documentation of all software components involved in the overall system safety.

C2) Does the standard utilize integrity or risk levels?

Integrity and risk levels provide a means of categorizing a software component or a hazard based on the overall risk posed. Integrity levels are generally applied to the software components and describe the certainty with which the software must perform. Risk levels are applied to hazards and indicate the severity of a hazard and probability that it will occur. After identifying safety-critical software components and system level hazards, the standard should assign each component an integrity or risk level. Each level has appropriate design and verification activities, defined by the standard, to ensure the software attains the desired integrity. As the risk software can cause to system safety increases, so does the integrity level and the amount of effort put into assuring its safety.

C3) Does the standard discuss requirements and traceability?

Software safety analysis should produce new, or identify existing, software requirements. Any software requirements necessary to mitigate or avoid a hazard are designated software safety requirements. The standard should discuss both reactive and proactive requirements, responding to hazards as well as monitoring the system for signs of hazards to come. The software safety requirements must be either uniquely identified in the software requirements document, or be separated into a software safety requirements document. As many of the software safety requirements originate in system hazard analysis it is important that they are linked backwards to the specific hazards. The standard should require traceability in all directions, from the hazards to safety requirements to implementation and back. This practice is vital to ensure full satisfaction of the requirements, as well as to identify unused code not fulfilling a requirement. In addition, all tests cases created should also trace back to the requirements they are testing. Maintenance of documentation and reviews of software safety requirements should be an ongoing process.

C4) Does the standard require consistent testing and validation?

System and software development consists of multiple steps, from gathering requirements all the way to maintenance, where the output of one step is used as the input for another. Testing for software safety should take advantage of this fact and test the output of each step to catch any errors early in the development process. The standard should require testing at the unit, component, and system levels. The standard should require that testing, usually at the unit level, covers all software paths as well as expected/unexpected inputs. At the component level, all interfaces should be tested. Ultimately, system testing should verify that the system operates in a safe manner even in the event of any failures or faults. Test case creation should be guided by the previously defined safety requirements and integrity levels. Finally, the standard should require proper documentation of all test data, including but not limited to test cases, simulators, drivers, and results. Each test should trace back to a software safety requirement to ensure full coverage.

C5) Does the standard discuss complexity management?

The general consensus in the software-safety industry is that safety assurance activities increase as the complexity of a safety-critical system increases. Despite the acknowledgement that complexity increases the amount of work required to verify safety, the majority of standards in use today fail to discuss the issue. As mentioned by Squair in his paper [21] titled “Issues in the Application of Software Safety Standards,” it is only reasonable for a standard to require the simplest software possible to meet the system’s needs. Simplicity should be a governing philosophy, not just a footnote. Where complexity is unavoidable, the standard should discuss how to separate the software into some sort of modules. Each module will be easier to understand and verify, and then the modules can be combined to verify they interact correctly. Finally, at the lower level, the standard should require that the software’s source code be reasonably free of any unnecessarily complex syntax.

C6) Does the standard address quality assurance?

It is expected that the standard does not define quality assurance (QA) practices itself, but that it requires compliance with a separate quality assurance standard. Two of the most well known QA standards are the ISO 9000 and the Software Capability Maturity Model (SW-CMM). The standard should require compliance with a well developed QA standard. The standard should define any extra requirements as well as any exceptions to the QA standard chosen.

C7) Does the standard address configuration management?

The standard should appoint a single “configuration manager” to develop a configuration management process for the project. The configuration management should control all project development tools and environments, test tools and environments, source code, and data. The process should be practiced during all steps of the development lifecycle to ensure any changes made are uniform.

C8) Does the standard incorporate the safety benefit versus the cost benefit?

Safety assurance activities can significantly extend the development time, and thus cost, of a project. The decision of how much work is necessary often comes down to the cost of the additional safety activities versus necessary level of safety required. The application of a rigorous safety standard to a low-risk system should be avoided as the added cost, and subsequent safety, is unnecessary. The standard should provide guidance as to the minimum level of risk the system should pose for the application of the standard to be justified. Alternatively, the standard could define different levels of assurance required based on the severity of the risk posed by the system. This information would allow the standard to be tailored to different levels of risk, scaling the cost as necessary.

C9) Does the standard discuss procedures for future updates to the software?

The release of software into its operational phase does not mean the end of software safety analysis. The standard should require a maintenance plan be developed prior to release for dealing with any routine updates, as well as any uncaught failures during use. Routine maintenance tasks, for example loading new terrain data for a new environment, should undergo the same rigor of safety analysis as the initial design. The standard should account for the possibility of new personnel being responsible for maintenance and should make any relevant documentation available. Finally, the standard should require an evaluation process to approve any major updates and to conduct scheduled reviews of any minor updates.

C10) Is the standard in use and updated?

For a standard to stay relevant it needs to be updated in response to any industry feedback it receives. A lack of industry feedback can lead to an "orphaned standard" where a standard has extremely limited support and few developers have experience using it. To prevent this, the standard should be under the control of a governing organization and be in current use in the industry. The standard should be reviewed periodically to make any necessary updates or revisions.

C11) Does the standard provide a means for certification?

To ensure that all requirements put forth by the standard have been met, the standard should require certification, usually from the standard's governing organization. The certification process should examine both the finished product, as well as the documentation necessary to ensure all development requirements were met. It is important to remember that certification does not guarantee safety, only that all of the standard's requirements were met.

C12) Is the standard easy to use?

Ease of use is not a requirement for a standard, but a standard that is easy to use will be better understood. Leveson argues in her book, *Safeware: System Safety and Computers*, that the majority of accidents involving software come about due to a lack of understanding of the problem domain [16]. If an incomplete understanding of the problem domain is detrimental to the design of a system, it is only reasonable to assume that an incomplete understanding of a standard will be similarly detrimental to safety. The standard should be well organized and should clearly state its requirements. The standard should also provide additional information in appendices or an accompanying document.

Table 1. Scoring table

Score	Description
1	Standard does not, or only barely, mention the topic.
2	Standard mentions, but provides no details on, the topic. May provide a reference to other standards or documents, but provides no additional information of its own and does not specifically require that the recommendations of the referred documents be followed. For example, the standard NASA-STD-8713.13B is given a score of 2 for Criterion C2 "does the standard utilize integrity or risk levels" because it references a separate document but does not require its use. The same standard receives a score of 4 for criterion C6 "Does the standard address quality assurance" because it specifically requires compliance with a separate document to which it refers.
3	Standard mentions the topic and provides some details but does not discuss the topic in depth, nor does it provide specific examples or recommendations.
4	Standard discusses the topic in detail with thorough explanation but may be unspecific or general in its recommendations. May provide extensive discussion on most aspects of the topic but be vague on a few details.
5	Standard thoroughly and extensively discusses all aspects of the topic. A complete explanation is given, and the standard provides specific examples and/or recommendations for implementation.

C13) Lessons Learned

This part is not expected to be covered in the standard itself. The purpose is to list any additional criticisms, from either the governing organization or a third party, as well as any information about accidents in which the standard was involved. To be involved in an accident, the standard must have been used to develop the system that caused the hazard.

C14) Governing Organization

The criterion examines whether there is an organization that developed or is currently maintaining the standard.

C15) Main Industry/Use

The objective is to list the industry or specific systems which the standard was designed to address.

A rating of 1 to 5 is used with 1 as the lowest score implying the standard does not mention the topic, and 5 as the highest score for a standard that closely matches the criterion. This rating is applied to all criteria except for the last three (lessons learned, governing organization, and main industry/use). Refer to Table 1 for more details.

3. Evaluation of each standard with respect to each criterion

We first present the evaluation using Criteria C14 and C15, as listed in Table 2, to explain how each standard is most likely used by which industry sector(s) in practice and their corresponding governing organization. This will help clarify the following evaluation as why a standard has a very stringent requirement with respect to a given criterion, whereas another standard does not seem to care much.

Results of our evaluation on each the five standards against criteria C1 to C12 are listed below. The scores are given based on the description in Table 2.

Table 2. Evaluation with respect to criteria C14 and C15

Safety Standard	Governing Organization	Main Industry/Use
FAA System Safety Handbook	Federal Aviation Administration (FAA)	Standard is intended to provide recommendations for the design, implementation, and improvement of system safety programs, especially within the aviation industry.
DO-178B	Radio Technical Commission for Aeronautics (RTCA)	Standard is intended to provide guidelines for the international aviation community in the creation of software for airborne systems.
MIL-STD-882D	United States Department of Defense (DOD).	Standard is intended for all DOD departments and agencies.
NASA STD 8719.13B	National Aeronautics and Space Administration (NASA) Office of Safety and Mission Assurance (OSMA)	Standard is intended for all NASA programs, centers, and facilities.
DEF-STAN 00-56	Created for the Defense Materiel Standardization Committee by the Safety Standards Review Committee of the UK Ministry of Defense (MOD) and overseen by the Directorate of Standardization.	Standard is intended for all MOD projects and authorities.

C1) Does the standard discuss splitting software into safety-critical components?

Of the five standards, the FAA Handbook and the NASA STD provide the most thorough discussions of this topic, and both of these standards receive a score of 5 for this criterion. They require the use of a preliminary hazard analysis, in which the entire system as a whole must be evaluated, as well as the possible contribution of software to potential hazards. In both standards, the partitioning of software into safety-critical components is addressed and given proper emphasis in the context of the system.

DO-178B acknowledges the use of partitioning as a way to isolate functionally independent software components. However, unlike the FAA Handbook and the NASA STD, it does not require the use of a system-level hazard analysis prior to software analysis, nor does it address the benefits of such a practice. In general, DO-178B touches upon, but does not address in detail, the splitting of software into safety-critical components and thus receives a 3 for this criterion.

MIL-STD-882D and DEF-STAN 00-56 are the two weakest standards, of the five evaluated, in their discussion of this topic. In MIL-STD-882D, a system level hazard analysis is required, but subsystem and software hazard analysis is not mentioned. Similarly, DEF-STAN 00-56 does not discuss the importance of splitting a system into safety-critical subsystems, nor does it address how to do so. As a result of their general lack of information relating to the safety-criticality of software components and their partitioning, both of these standards receive a 1 for this criterion

C2) Does the standard utilize integrity or risk levels?

The FAA Handbook, DO-178B, and MIL-STD-882D all thoroughly discuss the use of integrity or risk levels, and as a result all three standards receive a score of 5 in this criterion. More precisely, the mishap risk categories in MIL-STD-882D and the risk levels in the FAA Handbook are based on the severity and the probability of a hazard's occurrence, whereas the software levels in DO-178B are only based on the severity of the outcome. Table 3 and Table 4 compare these software safety criticality levels between the standards. The FAA Handbook and DO-178B also provide recommended design and verification activities for each risk level. Although MIL-STD-882D does not provide these recommended activities, it does provide example acceptance levels and suggests that the project team work to determine appropriate activities for each mishap level depending on the project's needs.

Table 3. Comparison of levels of software safety criticality based on severity of outcome

MIL-STD-882D	DO-178B	FAA System Safety Handbook
I. Catastrophic	A. Catastrophic	Catastrophic
II. Critical	B. Hazardous	Hazardous
III. Marginal	C. Major	Major
IV. Negligible	D. Minor	Minor
N/A	E. No Effect	No Safety Effect

Table 4: Comparison of levels of software safety criticality based on probability of occurrence

MIL-STD-882D	FAA System Safety Handbook
A. Frequent	Probable
B. Probable	Remote
C. Occasional	Extremely Remote
D. Remote	Extremely Improbable
E. Improbable	N/A

Like the three standards previously mentioned, DEF-STAN 00-56 also determines risk based on an analysis of a hazard's severity and probability. However, DEF-STAN 00-56 does not mention specific risk or integrity levels, nor does it provide recommended or required activities based on the level of risk. The standard provides guidance as to the importance of risk estimation, but does not provide specific levels related to integrity or risk, and leaves the determination of what actions should be taken largely up to the contractor. Overall, the standard seems to acknowledge the importance of determining risk level, but provides less guidance than the FAA Handbook, DO-178B, and MIL-STD-882D regarding how to do so. As a result, DEF-STAN 00-56 receives a 3 for this criterion.

Finally, the NASA STD is the weakest of the five standards evaluated for this criterion. Unlike the other four, it does not utilize or extensively discuss integrity or risk levels. Risk levels are only briefly mentioned, but the use of them is not required and information on how to utilize them is not given directly. As a result, the NASA STD receives a 1 for this criterion.

C3) Does the standard discuss requirements and traceability?

All five standards receive a score of 5 for this criterion. They require fully traceable and extensive documentation throughout all phases of system development. The FAA Handbook requires a safety action record protocol, which must track and measure hazards and the measures taken to mitigate them, and requires a chronological history of all actions taken with regard to a specific requirement. MIL-STD-882D requires a tracking system for all hazards, any actions taken in response, and residual risk. DO-178B stresses traceability between system requirements and software requirements, between high- and low-level requirements, and between low-level requirements and source code. NASA STD also requires that hazard analyses be used to determine and thoroughly document all requirements in order to mitigate each hazard identified, and that traceability must be ensured in all directions for all requirements. DEF-STAN 00-56 requires that a hazard log must be created and maintained, which should include information on the determination and management of all potential hazards and risks of the system, and traceability between safety requirements and their source must be provided for each safety requirement.

C4) Does the standard require consistent testing and validation?

DO-178B and NASA STD are most thorough in addressing testing and validation. Both standards require extensive and regular testing and validation throughout the system development process, including full and thorough documentation and traceability of all tests. They both receive a 5 for this criterion. The FAA Handbook also requires constant testing and analysis throughout the design and development process. However, it does not discuss the importance of having each test trace back to a specific software requirement to ensure full coverage. As a result, it receives a score of 4. Although MIL-STD-882D also requires consistent testing and validation throughout the development process, it does not require or recommend the use of system/subsystem testing or unit testing. We give it a 3. Finally, DEF-STAN 00-56 is the weakest because it only recommends, but does not specifically require, consistent testing and validation. The standard recommends the use of testing and some analysis-based validation; however, detailed guidance on how to achieve this is not provided. As a result, it receives a 2 for its fulfillment of this criterion.

C5) Does the standard discuss complexity management?

All five standards are relatively weak in their discussions of complexity management. The FAA Handbook briefly discusses the measurement of complexity, but simplicity is not listed as an important concern, and the Handbook does not discuss the importance of separating the software into simpler modules for ease of understanding and modification. Similarly, both MIL-STD-882D and NASA STD do not address complexity management, nor do they give simplicity an important role in the design of software requirements or systems. DEF-STAN 00-56 acknowledges that more complex systems generally come with higher risk, and therefore require more effort in achieving safety. However, the importance of purposefully managing this complexity is not expressly delineated and further guidelines are not given regarding this topic. DO-178B provides the strongest discussion of complexity management of the five standards evaluated, although it is not as extensive as it could be. It discusses partitioning and modularization as techniques for isolating faults and decreasing software verification effort, but it does not require their use. It suggests that design standards should include complexity restrictions such as a maximum level of nested calls; however, simplicity does not seem to be an important design philosophy.

C6) Does the standard address quality assurance?

DO-178B and NASA STD are the strongest in their discussion of quality assurance. While DO-178B extensively discusses quality assurance in all of its aspects, it falls slightly short by not requiring adherence to a more extensive quality assurance standard. On the other hand, NASA STD does require compliance with a separate quality assurance standard, but does not go into as much detail as DO-178B on quality assurance activities specifically within the standard itself. However, both standards cover the topic of quality assurance relatively strongly and thus both receive a score f 4. DEF-STAN 00-56, MIL-STD-882D, and the FAA Handbook, however, are all weak in the quality assurance category. The FAA Handbook and DEF-STAN 00-56 both briefly mention quality assurance but do not go into detail, nor do they provide any details on specific quality assurance requirements or how to implement them. Neither standard requires compliance with a separate quality assurance standard or references quality assurance requirements or recommendations from another document. As a result, both the FAA Handbook and DEF-STAN 00-56 receive a 2 for their fulfillment of this criterion. MIL-STD-882D is the weakest. The standard does not mention or address quality assurance, or provide reference to another quality assurance standard or document. It receives a 1 in this category.

C7) Does the standard address configuration management?

NASA STD and MIL-STD-882D are the strongest in their discussion of configuration management. Both require that software configuration management be addressed by a single person, and provide extensive requirements for software configuration management. They receive a 5 in this category. DO-178B also thoroughly discusses software configuration management and mandates that the various aspects of configuration management be addressed; however, it does not appoint a configuration manager and leaves the details of the methodology and process of configuration management up to the contractor. Thus, it receives a 4 in this category. DEF-STAN 00-56 and the FAA Handbook are the weakest. Neither of these standards go into detail on the software configuration management process, nor do they provide guidance on how to achieve any aspects of configuration management. Therefore, these two receive a score of 2.

C8) Does the standard incorporate the safety benefit versus the cost benefit?

MIL-STD-882D and DO-178B are the strongest in their discussion of safety benefit versus cost. While MIL-STD-882D addresses the safety benefit/cost tradeoff directly, DO-178B does not specifically discuss the relationship between the two factors but instead provides extensive instructions on how to tailor the standard's requirements and activities to different safety criticality levels. Both standards, however, clearly support the idea of tailoring a safety program to a project's particular needs in both cost and safety criticality. As a result, they receive a 5 for their fulfillment of this category. The FAA Handbook is also strong in this category, although it lacks some thoroughness found in the previous two standards. It directly addresses the safety benefit versus cost tradeoff, and explicitly acknowledges the importance of properly managing this balance. However, it does not provide as much guidance as MIL-STD-882D and DO-178B regarding how to achieve this balance, since it does not give recommendations for safety assurance activities for different risk or criticality levels. It receives a 3. Finally, neither NASA STD nor DEF-STAN 00-56 directly addresses the safety benefit versus cost tradeoff, nor do they provide guidance as to how to balance these two important aspects of a software project. Each standard briefly mentions that a software project should tailor its activities to its own size and needs, but further discussion of why or how to do so is not provided. As a result, they receive a 2 in this category.

C9) Does the standard discuss procedures for future updates to the software?

NASA STD is the strongest in its discussion of future updates to the software. It extensively discusses the process of software maintenance and updates, and provides extensive requirements and recommendations regarding this topic. It receives a 5 in this category. DEF-STAN 00-56, the FAA Handbook, and MIL-STD-882D are also strong in their discussion of this criterion. All three standards explicitly discuss the importance of properly managing future updates to the software, but do not provide as thorough guidance as NASA STD for how to achieve this goal. Thus, these three are given a 4 for this criterion. Finally, DO-178B is the weakest. It does not discuss procedures for future updates to the software, or require any specific methodology for maintenance and upkeep of the system after it enters its operational phase. Although the standard does require that documentation on the use and development of the system be kept up to date, its discussion of maintenance and updates to the system is relatively weak and thus receives a 2 for this criterion.

C10) Is the standard in use and updated?

DEF-STAN 00-56 and NASA STD 8719.13B are the most frequently, and recently, updated standards of the five reviewed. Both are also in wide use. As a result, both receive a 5 for this criterion. DO-178B is also widely in use, and is perhaps the most popular software safety standard in the civil aviation industry, as well as being heavily used in many other safety-critical software industries. It has also been updated several times, but its updates are less consistent than those of DEF-STAN 00-56 and NASA STD. It receives a 4 in this category. The FAA Handbook and MIL-STD-882D receive a 3, as they are the less frequently updated.

C11) Does the standard provide a means for certification?

DO-178B and NASA STD both receive a score of 5, because both standards extensively address certification and provide detailed requirements regarding the certification of software. While the NASA STD provides a certification office, the NASA OSMA, DO-178B does not specify a particular certification authority. However, both standards provide detailed guidance in this category. DEF-STAN 00-56 is slightly less strong in this category than DO-178B and NASA STD, as it does not require independent certification, nor does it provide a certification authority. However, it does provide extensive guidance on how to document and assess whether a project follows the standard's requirements and recommendations. It receives a score of 4. On the other hand, MIL-STD-882D and the FAA Handbook are both weak in their discussion of software certification. The FAA Handbook does not require, or even recommend, software certification. Similarly, MIL-STD-882D does not require certification or discuss any aspects of the certification process. However, MIL-STD-882D does list some agencies through which software certification can be obtained (although the use of MIL-STD-882D is not required in those cases), and the FAA Handbook requires the use of a system safety program plan which can be examined to ascertain that the Handbook's recommendations were followed. Due to their shortcomings, both MIL-STD-882D and the FAA Handbook receive a score of 2 for this criterion.

C12) Is the standard easy to use?

All five of the standards evaluated receive high scores in this category. DO-178B, MIL-STD-882D, NASA STD, and DEF-STAN 00-56 are well organized and provide detailed glossaries or appendices with definitions of all terms used in the respective standards. They all receive a score of 5. The FAA Handbook is also clearly written and well organized; however, it is the longest of the five standards and contains some repetition that may make it longer than necessary. As a result, it is slightly more cumbersome to use than the other four standards and receives a 4 in this category.

4. Lessons Learned

Following criterion C13, here we list some lessons learned from using these five safety standards. A common accident theme among aviation systems is the unintended effects of updates to the system. According to the FAA, this occurs when "an initiative, change, new process, or other activity intended to improve something actually produces, in addition to improvement, an undesirable outcome." [13] While the FAA Handbook does provide guidelines for maintenance and updates to the system, specifically in the system safety plan, this suggests that more attention should be paid to the importance of thorough integrity and risk analysis on any updates and/or changes to the system, as well as the management of complexity in an updated system, and the Handbook should reflect this focus.

Similarly, NASA has had accidents tied to software complexity management problems. Software bugs were implicated in the crash of the Mars Polar Lander in 2000, a system jointly developed by NASA and Lockheed Martin Aeronautics [4]. A software error caused a sensor signal to incorrectly report that the probe had touched down prematurely, causing the descent engines to shut down prior to landing. It was speculated that ineffective engineering practices for managing complexity largely contributed to the error [4]. It is possible that the lack of discussion of complexity management in NASA's Software Safety Standard (NASA-STD-8719.13B) [18] caused the engineers developing the system to overlook these issues.

Furthermore, the FAA, in analyzing recent accidents, has determined that a "lack of system isolation/ segregation" is a common cause of flight accidents [13]. Since many aviation systems use the FAA's System Safety Handbook [12] as a guide, these accidents would indicate that updating the Handbook to reflect these lessons might help future projects to avoid similar problems. Although the Handbook does discuss the importance of subsystem analysis and splitting software into safety critical components, it might be beneficial to ensure that thorough segregation of subsystems and modules is emphasized.

Another common criticism of software safety standards involves cost issues [25]. For DO-178B [22], criticisms have been made that developing to the standard can be very expensive for many projects. Although it is difficult to collect exact data on the full cost of developing to DO-178B for most projects, it has been noted that the standard should add only 5% (for level D) to 55% (for level A) to development costs. However, sources estimate that industry average costs are anywhere from 20%−50% more than the aforementioned recommended costs [15], to 75%−150% more than the cost of non-safety-critical applications [14]. Another source estimates that developing to Level A of the standard can increase the cost to a factor of five over that of non-safety-critical projects [2].

In addition, criticisms have been made that the DO-178B standard may have contributed to unnecessary delays in the delivery of aircraft such as the Boeing 787 [19]. On the other hand, Lockheed Martin Aeronautics reported that they were able to develop their C-130J aircraft to DO-178B Level A for half the cost of non-safety-critical code [1,2]. Additionally, their testing process cost was less than a fifth of normal industry costs [2,3].

The above examples suggest that there is a tradeoff between the safety benefits of the strict and rigorous guidelines provided in the standard, and the costs of implementing them. However, it is possible to develop to the highest level of the standard for a relatively low expense, provided that the correct development practices are followed.

Another common problem with software safety standards is a lack of sufficient supporting guidance. In some areas, several standards provide strict requirements but lack extensive recommendations on how to achieve these goals. For example, MIL-STD-882D [9] only defines what is required, rather than how to implement the requirements. As a result, the government and industry recognized the need for creation of supporting recommendations on how to utilize the guidelines set forth in the standard [20]. Because this was unavailable, industry often supplemented with DO-178B [22] but there was still confusion regarding further information. There was also no consensus or DoD policy requirement on when to require or utilize the standard.

As a result, in September 2004, the memo "Defense Acquisition System Safety" was released, which specifically mandated the use of MIL-STD-882D to manage risk [26], and the Defense Acquisition University created a course, "System Safety in Systems Engineering," which was the first formal guidance on how to effectively use and implement MIL-STD-882D [20]. The standard was additionally mandated in the November 2006 Defense Acquisition Guidebook [5] and the December 2008 DoD 5000.02 "Operation of the Defense Acquisition System" [10]. These updates and additional documents provided guidance on issues that were confusing or sparsely illustrated in MIL-STD-882D, specifically the importance of integrating the safety tasks into the entire systems engineering process.

Like MIL-STD-882D [9], previous issues of DEF-STAN 00-56 [17] have been criticized for a lack of sufficient guidance on certain requirements and recommendations found in the standard. Interim Issue 3 of the standard, published in December 2004, incorporated some aspects of the old standards DEF-STAN 00-54 [6], 00-55 [7], and 00-58 [8]. Issue 3 was a major departure from the previous issue in that it was a goal-based standard. Experience and feedback from Ministry of Defence stakeholders and users in industry had shown that the rigorous requirements of earlier issues were needlessly strict for contractors [11]. The standard had been criticized for not allowing contractors the flexibility to tailor their approach for each individual project to best achieve the safety requirements. Furthermore, experience showed that examples from the standard that were intended to provide guidance for how to achieve certain techniques were often copied directly, rather than altered to fit the project. The new goal-based approach for the standard sets out general requirements, but does not mandate a specific method for how they are to be met. Contractors must propose and justify their chosen method of compliance. This puts a higher burden of proof on contractors, but also allows them more flexibility in tailoring their approach to safety to fit the needs of their specific project. This can also permit contractors to use other relevant standards and utilize this in their argument for the safety of their system.

5. Conclusion and Future Work

We present a systematic evaluation of five commonly used safety standards using 15 proposed criteria to show the strength and weakness of each standard. We also discuss the potential enhancements based on the comments from practitioners who actually use these standards to produce real-life safe software at work. Some software-related accidents are also reviewed. Although few of them have been directly attributed to the five standards examined here, the criticisms listed indicate that the software safety industry would benefit from addressing these lessons in order to better the software safety landscape as a whole. Furthermore, additional lessons could perhaps be found by keeping records of all of the projects which use these standards, and any failures, cost overruns, or accidents that they may face. A current lack of any centralized repository for the lessons learned from the use of these standards makes it more difficult to fully explore the benefits that may be gained by examining any problems to which the standards may have contributed. One of our ongoing efforts is to create the aforementioned repository to systematically integrate the sporadic information with a careful screening and present an overall picture of how software safety standards have been applied to produce safe software in practice, the limitation of using these standards, and how to adjust software processes, methods, and tools in response to a specific level of safety requirements so that the final products can be delivered in a more cost-effective way.

Acknowledgment

This paper is based on the work by the students who participated in the REU (Research for Undergraduate Students) program at the University of Texas at Dallas from 2009 to 2013. The program is funded by the National Science Foundation (NSF CNS-1156977). More details of our REU program can be found at http://paris.utdallas.edu/reu.

References

1. C. Abts, B. W. Boehm, and E. B. Clark, "COCOTS: A COTS Software Integration Lifecycle Cost Model- Model Overview and Preliminary Data Collection Findings," *USC Center for Software Engineering*, 2000
2. P. Amey, "Correctness by construction: better can also be cheaper," *CrossTalk: Journal of Defense Software Engineering*, pp. 24-28, March 2002
3. P. Amey and A. Hilton, "Practical Experience of Safety- and Safety-Critical Technologies," *Ada User Journal*, vol 25, no. 2, pp. 98-105, June 2004
4. G. Clark, "Fatal Error: Buggy Software May Have Crashed Marks Polar Lander," *TechMediaNetwork*, March, 2000
5. Defense Acquisition University, "Defense Acquisition Guidebook," February 2010 (http://www.ndia.org/Advocacy/LegislativeandFederalIssuesUpdate/Documents/March2010/Defense_Acqauisition_Guidebook_3-10.pdf)
6. Defence Standard 00-54, "Requirements for Safety Related Electronic Hardware in Defence Equipment," Issue 2, December 1996.
7. Defence Standard 00-55, "Requirements for Safety Related Software in Defence Equipment," Issue 2, Aug 1997.
8. Defence Standard 00-58, "HAZOP Studies on Systems Containing Programmable Electronics," May 2000.
9. Department of Defense, United States, "Standard Practice for System Safety" (MIL-STD-882D) (http://www.system-safety.org/Documents/MIL-STD-882D.pdf)
10. Department of Defense, United States, "Operation of the Defense Acquisition System" (DoD Instruction 5000.2), April 2002.
11. Director General Safety & Engineering, DStan, Ministry of Defence, United Kingdom, Standards in Defence News, Issue 205, July 2007
12. Federal Aviation Administration (FAA), United States, "System Safety *Handbook*" (http://www.faa.gov/regulations_policies/handbooks_manuals/aviation/risk_management/ss_handbook/)
13. Federal Aviation Administration (FAA), United States, "Lessons Learned From Transport Airplane Accidents," Release 9, 2013 (http://accidents-ll.faa.gov/)
14. HighRely Inc. "DO-178B and DO-254: Big Bang or Evolution?"2005
15. HighRely Inc, "DO-178B Costs versus Benefits," 2005
16. N. G. Leveson, "Safeware, System Safety and Computers," *Addison Wesley*, 1995
17. Ministry of Defence, United Kingdom, "Safety Management Requirements for Defence Systems" (DEF-STAN 00-56)
18. National Aeronautics and Space Administration (NASA), United States, "Software Safety Standard" (NASA-STD-8719.13B),(http://www.system-safety.org/Documents/NASA-STD-8719.13B.pdf)
19. B. Rigby and T. Hepher, "Brake Software Latest Threat to Boeing 787," Reuters, July 2008
20. R. E. Smith and S. G. Forbes, "Overview of Draft MIL-STD-882D w/Change 1," in *Proceedings of 12th NDIA Annual Systems Engineering Conference*, San Diego, October 2009
21. M. J. Squair "Issues in the application of software safety standards," in *Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software,* pp. 13-26, Darlinghurst, Australia, August 2005
22. Radio Technical Commission for Aeronautic, Inc., "*Software Considerations in Airborne Systems and Equipment Certification*" (DO-178B), 1992

23. W. E. Wong, V. Debroy, and A. Restrepo, "The Role of Software in Recent Catastrophic Accidents," *IEEE Transactions on Reliability*, Volume 59, Issue 3, pp. 469-473, September 2010
24. W. E. Wong, V. Debroy, A. Surampudi, H. Kim, and M. F. Siok, "Recent Catastrophic Accidents: Investigating How Software Was Responsible," in *Proceedings of the 4th IEEE International Conference on Secure Software Integration and Reliability Improvement* (SSIRI), pp. 14-22, Singapore, June 2010
25. W. E. Wong, A. Demel, V. Debroy and M. Siok, "Safe Software: Does it Cost More to Develop?" in *Proceedings of the 5th IEEE International Conference on Secure Software Integration and Reliability Improvement* (SSIRI), pp. 198-207, Jeju Island, Korea, June 2011
26. M. Wynne, "Memorandum for Defense Acquisition System Safety*," Acting Undersecretary of Defense*, September 2004, (https://acc.dau.mil/adl/en-US/18586/file/740/Defense%20Acquisition%20System% 20Safety.pdf)