ACQUISITION Research
Sponsored Report Series

**Requirements Framework for the
Software Systems Safety Review Panel (SSSTRP)**

30 September 2009

by

Dr. Luqi, Professor

Dr. Valdis Berzins, Professor

MAJ Joey Rivera

Graduate School of Operational & Information Sciences

Naval Postgraduate School

Abstract

This paper describes the research and progress made during FY09 at the Naval Postgraduate School on a Software Systems Safety Review Panel (SSSTRP) Requirements Framework. Accomplishments made in FY09 include the discovery of the primary causes for the high level of vendor failure rates during the SSSTRP process. Research showed that the lack of structure associated with the vendor-provided Technical Review Package (TRP) led to inconsistent documentation and standards in the SSSTRP process of evaluating the vendor's software safety risk. The development of a domain-specific Requirements Framework designed to work with the SSSTRP process will both help the vendor fully understand the measurable requirements for the TRP, and the SSSTRP members to understand the measurable standard by which the TRP is evaluated. This process should result in a reduction of SSSTRP failures.

This paper further discusses the application of the NASA Software Safety Standard to Naval Weapons Systems development processes. This development is dependent on commercial off-the-shelf (COTS) software in order to meet deadline and cost requirements; however, this dependency poses a problem, as commercial programs are not commonly designed to a high standard for safety-critical applications. The NASA *Software Safety Standard* is one of the most robust software safety assessment standards that can be identified and, thus, provides an ideal basis for assessment of COTS software components for Naval requirements. This report identifies the portions of the NASA *Software Safety Standard* that are relevant to the assessment of COTS software and proposes a guideline of how these standards can be applied to the Naval weapons systems development. This discussion includes both an analysis of the standard itself and justification of the need for safety-critical applications within the Naval Weapons Systems development. It also includes a brief discussion of the program, and identification and application of the appropriate portions of the standard to Naval weapons systems development (including the identification of checklists and other features

that must be integrated into the system). This report can be used to identify specific ways in which the NASA *Software Safety Standard* can be applied to Naval requirements, as well as to identify potential gaps in the standard that could be addressed by the Navy in an extension of this standard.

Acknowledgments

THIS PAGE INTENTIONALLY LEFT BLANK

About the Authors

Dr. Luqi is a Professor of Computer Science at NPS. Her research on many aspects of software reuse and computer-aided software development has produced hundreds of research papers in refereed journals, conference proceedings and book chapters. She has served as a PI or co-PI for many research projects funded by the DoD and DoN. She has received the Presidential Young Investigator Award from NSF and the Technical Achievement Award from IEEE.

Valdis Berzins is a Professor of Computer Science at the Naval Postgraduate School. His research interests include software engineering, software architecture, computer-aided design, and theoretical foundations of software maintenance. His work includes papers on software testing, software merging, specification languages, and engineering databases. He received BS, MS, EE, and PhD degrees from MIT and has been on the faculty at the University of Texas and the University of Minnesota. He has developed several specification languages, software tools for computer-aided software design, and a fundamental theory of software merging.

MAJ Joey Rivera is a Reserve Officer assigned to US Pacific Command J63 as an Information Assurance Officer. In his civilian profession, he is President of Rivera Consulting Group, which works with customers in the DoD. He is an expert in Information Technology business process reengineering and has helped agencies in both the public and private sectors streamline their operational costs. He has over 20 years of IT experience—ranging from Programmer to Program Manager. He has a Master's Degree in Computer Resources and Information Management from Webster University and is currently pursuing a PhD in Software Engineering at the Naval Postgraduate School.

THIS PAGE INTENTIONALLY LEFT BLANK

ACQUISITION Research
Sponsored Report Series

**Requirements Framework for the
Software Systems Safety Review Panel (SSSTRP)**

30 September 2009

by

Luqi,

Valdis Berzins, and

Joey Rivera

Graduate School of Operational & Information Sciences

Naval Postgraduate School

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

Introduction

The use of commercial software in safety-critical systems within some contexts (such as the Naval weapons system development program) is increasingly common, as it has been shown to be highly cost-effective and may speed system development time (NASA, 2004a, p. 269). However, in some cases safety issues are not even considered; instead, off-the-shelf software is used by default—such as with operating systems, low-level real-time operational code (for example, BIOS software), and seemingly non-complex systems such as word processing or e-mail. However, the use of these systems in the business environment does yield some degree of difficulty in that their use is not strongly controlled and does pose a risk to safety-critical systems.

One of the most complex issues in using commercial off-the-shelf (COTS) software for safety-critical applications is that, in most cases, this software is designed with commercial goals in mind. This means that the software is not ideal in terms of functionality, but instead meets the majority of the needs of its users in terms of both functionality and safety, with improvements added as an incremental process. This software development methodology reduces the cost and time required for development and allows commercial firms to release products in a timely manner. In many cases, the potential for failure is not necessarily a problem—commercial and other enterprises can often sustain a brief service interruption or endure difficulties caused by software that is not perfectly functional in a given area. However, for safety-critical software or security-critical software, this approach to commercial software development can be highly problematic and drastically reduce the utility of the software program. In many cases, safety-critical and security-critical software applications are designed from scratch in order to meet the enhanced safety requirements and security framework requirements that allow the systems to operate at higher levels of safety and security.

In some cases, however, this approach is neither necessary nor desirable. For example, ongoing development of the Naval Weapons Systems programs is increasingly reliant on COTS software (as well as hardware) components, which are integrated into air and sea weapons commands as well as into command and control centers across the Navy (Friedman, 2006, p. 100). This integration of COTS software components provides a much faster and, in many ways, more robust means of integrating and upgrading systems; however, it does pose some security risks. Foremost among these risks is the lack of a clear standard governing assessment of the safety and security of COTS software components.

There are some standards that have been developed for analysis of safety-critical software that could be applied. Foremost among these standards is the NASA *Software Safety Standard*, which was developed for and is applied to all NASA software development projects (including internal software development, as well as COTS software components). This *Standard* addresses the more stringent quality and safety requirements needed for software that will be deployed in situations in which human lives are at stake. It is highly robust and has been proven in a number of safety-critical solutions, and thus was selected as the most appropriate choice for this situation. This report discusses the needs of the Naval Weapons Systems development program in brief, analyzes the NASA *Software Safety Standard*, and determines how this *Standard* can most effectively be applied to the development situation at hand. It then identifies potential gaps in the *Standard* and provides recommendations for filling those gaps. The goal of this discussion is to provide a clear guideline for application of the NASA *Software Safety Standard* to COTS-based development in the Naval Weapons System program.

The Naval Weapons System Program

Naval weapons systems are based in network-centric communications and technologies (Friedman, 2006, p. vii). Surveillance, communication and monitoring networks are used to channel information to the people operating the system, providing real-time feedback and awareness of situational aspects outside the reach of his or her own senses. According to Friedman (2006), active development is ongoing in a wide range of systems—including surveillance and communication, combat direction systems, radar, electro-optical sensors, shipboard guns and gun systems, strategic strike systems, antisubmarine and antiaircraft systems, electronic warfare, and mines and mine countermeasures. These systems each rely on an integrated system of software that handles communication, sensory and targeting capabilities, and other tasks. Because these systems are used in safety-critical situations, and their failure could mean a highly disastrous outcome for the operator of the system (as well as others that are depending on him/her), the need for safety-critical engineering design standards is clear. Within these systems, the software should be as robust and well designed and engineered as the hardware on which it is based.

The Naval Open Architecture and Use of COTS Software

Naval weapons system design is based in the Naval Open Architecture. The strategic goals of the Naval Open Architecture include "Encourage competition and collaboration […] build modular designs and disclose data to permit evolutionary designs, [...] build interoperable joint warfighting applications and ensure secure information exchange […] identify or develop reusable application software […] [and] ensure lifecycle affordability" (Department of the Navy, 2009, p. 2). Avoidance of COTS obsolescence is a major component of the goal of lifecycle affordability . One of the major components of the Naval Open Architecture strategy is lowering development cost and time by integrating COTS software wherever possible, and then using custom software to develop a modular system that can be rapidly redeveloped or updated if required (Department of the Navy, 2009). This system has

been in place since 2004, and a large number of operational weapons systems have been developed either wholly or partially using COTS software components (Department of the Navy, 2008). Some of these systems include SONAR systems, onboard ship management and communication systems, and surface ship directional systems, among many others (Friedman, 2006, pp. 107,119). This program has proved to be highly effective in terms of both cost savings and efficiency. For example, one SONAR cabinet system developed using primarily COTS technology was found to be less than a quarter of the cost of the MilSpec custom-designed alternative (p. 667).

The use of COTS hardware and software has become extremely common in Naval weapons system design, as discussed above. However, there is still no single safety standard for integration of COTS software components, and safety of software is evaluated on a case-by-case basis. This lack is particularly problematic because software quality testing is not as straightforward as hardware safety evaluation; its complexity can hamper attempts to determine quality and reliability. By instituting a single standard of software safety, decision-makers would provide an increased level of safety and control of COTS component integration.

The NASA *Software Safety Standard*

One highly viable candidate standard for integration into Naval weapons systems development is the NASA *Software Safety Standard*. The NASA *Software Safety Standard* was developed for NASA by the Office of Safety and Mission Assurance, and is used to assess software risk in all programs used within NASA's systems (NASA, 2004b, p. 1). Specifically, it is applied in the following situations:

> Safety-critical systems that include software must be evaluated for software's contribution to the safety of the system during the concept phase, and prior to the start, or in the early phases, of the acquisition or planning for the given software. Unless the evaluation proves that the software is not involved in the system safety, this Standard is to be followed. (NASA, 2004b, p. 1)

Safety requirements that are addressed by this *Standard* include process-oriented and technical requirements (p. 1). Both kinds of safety requirements must be met. Technical requirements are not specified by the *Standard*, but are instead identified by the manager of the software development process during the requirements and design phase of the project (p. 1). The *Standard* specifies only the process-oriented requirements of the system. As the document states, use of the *Standard* for process-oriented aspects of software safety does not preclude the requirement for the development and verification of the system to address technical-oriented software safety issues. However, the process standards that are included are designed to encompass the determination of what level of software safety is required.

The scope of the NASA *Software Safety Standard* includes:

- Identification of the need for software safety and requirements generation,

- Consideration of software safety within the system design,

- Discussion of software safety in project planning, management, and control activities,

- Software safety throughout the lifecycle from requirements generation

to operation,

- Assurance that all COTS and contracted software undergo software security evaluation and determination of its "safety contribution and limitations"

- Inclusion of software safety verification in software verification processes,

- Software certification requirements, and

- Change and reconfiguration management during operational use (NASA, 2004b, p. 2).

Although much of this scope can have some incidental application to COTS software acquisition (for example, many integration processes that involve COTS software do involve a degree of project management, even if this is not a formal software development effort), there are also provisions that apply directly to the acquisition and use of COTS software components. It is these areas that will be most interesting within this discussion.

In order to enact the guidelines of the *Software Safety Standard*, NASA has also produced a *Software Safety Guidebook* (NASA, 2004a). This *Guidebook* provides operational guidelines for enacting the software security standards encapsulated in the *Standard* document. The *Guidebook* document includes technical details and information intended to guide the development of operational safety practices (NASA, 2004a, p. 12). The document includes not only information for programmers, but also information for program and project managers intended to ensure that these personnel understand the process and requirements of software safety. As such, these two documents are incontrovertibly connected, and should be used together in an operational software safety setting to ensure that both the technical demands of software safety and the need for an organizational integration of software safety standards are met.

Determination of Safety-critical Systems

The NASA *Standard* offers a number of criteria on which software can be evaluated for safety criticality. The overarching criterion is inclusion within a safety-critical system—in which case, all software is included, as all is presumed to be safety-critical unless it can be shown to not be safety-critical (NASA, 2004b, p. 14). Further criteria include: causing, contributing to, controlling, or mitigating a hazard; controlling safety-critical functions or processing safety-critical commands or data; either detects and reports or corrects a system in a hazardous state or mitigates damage from a hazardous state; or resides in the same system as a hazardous state (p. 15). Additionally, if the system processes or analyzes data used in a safety-critical situation or verifies or validates other safety-critical systems, the system should be considered to be safety-critical (p. 15). The evaluation of safety-critical status extends not only to software, but also to data required to make safety-critical decisions (p. 15).

Risk-mitigation Processes

A variety of processes have been identified for risk mitigation. One of these processes is the isolation of safety-critical and non safety-critical software through a process such as partitioning. This is done in order to prevent failure of non-safety-critical software from negatively impacting the operation of safety-critical software (NASA, 2004b, p. 15). The use of an evaluation process that identifies the safety-critical nature of a given component during the conceptual phase of the project, prior to acquisition, is also highlighted as a means of ensuring that the appropriate software acquisition goals will be used (p. 15). The *Standard* does note that this evaluation can be performed by the supplier rather than the purchaser, but that if this is the case, it must be noted within the project plan, and the system should still be evaluated for the need for safety-critical software prior to the process (p. 15). The *Standard* also notes that the use of mitigation techniques (like manual operator overrides) should not affect the absolute determination of the appropriate level of software safety within the system (p. 16). This will prevent failures due to ineffective

mitigation techniques or integration of software that is essentially unsuitable due to reliance on an erroneously inflated perceived value of a given mitigation technique.

The Process of Evaluation and Planning

The NASA *Software Safety Standard* presents a specific approach to identifying system hazards, which must be applied to each individual project or acquisition (NASA, 2004b, p. 22). The first stage in this analysis is the use of Preliminary Hazard Analyses (PHAs), which "identify potential system hazards and may identify which proposed subsystems contribute to, or are needed to control, those hazards" (p. 16). The PHA can then be used to determine where—within the system as a whole—safety-critical design may be required. The *NASA Standard* requires that software safety analysis should take place along side and be integrated into system safety analyses from the conceptual stage onwards through the system lifecycle (p. 16). The *Standard* also requires a program to record the identified software safety requirements together with information about software hazards and how they can be controlled in an appropriate document: a system safety plan, software management plan, software or system assurance plan, or standalone software safety plan (p. 16).

The *Standard* also identifies software safety planning as one of the mission-critical activities in safety-critical software development and acquisition (NASA, 2004b, p. 21). Personnel involved in the process of software safety planning include software assurance engineers, project managers, and others involved in the development process (p. 21). The *Standard* recommends the establishment of a Software Safety Plan, which should outline the software safety process for the facility or project as a whole, "including organizational structure, interfaces, and the required criteria for analysis, reporting, evaluation and data retention to provide a safe product (p. 21)." It should include analysis details and scheduling for the project's safety planning discussion; however, the *Standard* allows for the use of both standalone documents and documents integrated into the overall project management plan. NASA recommends that software safety planning should

encompass not only the initial acquisition, but the entire software lifecycle—through the acquisition stage to implementation, use and maintenance (NASA, 2004b, p. 24). There are a variety of documents associated with the *Standard* for safety assurance; the most relevant of these documents for the purposes of COTS software include the Software Safety Plan, the Software Configuration Management Plan, the Software Requirements Specification, the Verification and Validation Plan, Safety Analyses and Reports, Test Documentation, and user documentation and procedures (pp. 24-25).

The *Software Safety Standard* and COTS Software

The majority of the remainder of the *Software Safety Standard* is dedicated to discussion of the software development process—which is largely irrelevant in this case, as software development is not part of the process of integrating COTS software. However, there is also an explicit discussion of the requirements for integration of COTS software into an existing or new software-based system. NASA's position on COTS software is addressed in Section 5.12 of the *Standard*, which addresses off-the-shelf (OTS) software (including commercial and government off-the-shelf software components). This discussion of off-the-shelf software is specific to both new software acquired for the project and software that has been reused from previous projects in the past (2004b, p. 28). The *Standard* states,

> It is important to evaluate the differences between how the OTS or reused software will be used within the new system, and how it was used in previous systems. Differences in configuration of the software or operational constraints may affect the operation of the OTS/reused software, sometimes in unexpected ways. (NASA, 2004b, p. 28)

The guidelines for handling off-the-shelf software include evaluation of all OTS software for the "potential to impact safety-critical functions within the current system" (p. 28). This includes a safety analysis process that evaluates not only the software's ability to meet the level of safety required for the current project, but the impact of any additional functionality or the potential inclusion of the software in

future projects (p. 28). This also includes evaluation of not only the code in and of itself, but also its ability to interface with other pieces of code, hardware, and the system as a whole (p. 28). Specifically noted is the interaction between COTS software and other software in the planned system, including other OTS software and custom-developed modules (p. 29). The *Standard* specifically recommends the use of black-box testing in order to ensure that COTS software is equivalent in safety standards to in-house developed software (p. 29). (Black-box testing, or data-driven testing, is testing of the functionality of the code without considering its structure, with the intention of finding non-conformances (Myers, Badgett & Thomas, 2004, p. 9).) NASA also notes the isolation of safety-critical and non-safety-critical components within the system as being particularly important for the integration of COTS software into safety-critical systems (NASA, 2004b, p. 35). As stated previously, while the needed isolation can be designed into the system from the start in an in-house development process, COTS software is often not designed as safety-critical. Thus, this may not be a natural feature of the COTS software package or component. As such, PMs should pay particular attention to the safety criticality of a system when choosing COTS components.

Implementation of the *Software Safety Standard*

While the *Software Safety Standard* documentation does provide a perfunctory example of how to evaluate software safety risk in an operational setting, this document does not provide specific implementation details, but is rather more of a rough guideline. In order to fill this gap, NASA has also instituted a *Software Safety Guidebook* which addresses the implementation details of the *Standard* in such a way that it can serve as a template for implementation in another organization. Although many of the concerns throughout the *Guidebook* are relevant to the discussion of COTS software, the *Guidebook* also provides a specific discussion of the use of COTS software and software acquisition (Chapter 12 of the *Guidebook*). The highlights of this discussion are addressed below.

Initial Acquisition and Implementation of COTS Software

The *Guidebook* identifies a number of concerns with the use of COTS software in terms of implementation in a safety-critical system (NASA, 2004a, pp. 270-271). These include: inadequate or inaccurate documentation that does not provide sufficient information for integration; no access to source code (which can impede appropriate safety analysis); no information about the software development or testing processes used; the potential that the OTS developers either do not fully grasp interactions within the system or don't communicate them to the user; incomplete information regarding software bugs (including deliberately misleading statements as well as bugs that simply were not detected and corrected during the development process); no software analysis; and missing or extra functionality. The last point is particularly problematic, as it either requires the use of glueware (which can increase the inherent risk within a system due to its own bugs and defects) or, in the case of extra functionality, may pose a threat on another level. The addition of extra functionality that will not be used within the system is a risk; for instance: 1) this additional functionality may be exploited at some point in the future, or 2) instability within untested extra functionality may be problematic for the system itself (NASA, 2004a, p. 271). Because of these potential issues, the use of COTS

software within safety-critical designs requires extra vigilance in terms of safety assurance and risk management. Extra functionality can also complicate testing procedures.

The *Guidebook* offers a specific checklist for safety-critical off-the-shelf applications (Appendix A), that can be referred to for specific application. (This inclusion is a truncated version of the checklist that contains only the main points of each item. Specific technical details are included in the *Guidebook*'s checklist, and PMs should refer to NASA's text when putting this checklist into use). The checklist offered has specific safety-critical features highlighted, such as the ability to recover previous software configurations and the need for a safety-impact assessment (p. 360). The construction of this checklist is such that , by following it during the assessment of a given COTS software plan, a PM could eliminate a large number of potentially inappropriate choices for implementation—some of which may already be addressed in implementation testing. For example, basic hardware and software compatibility is addressed within this checklist. Although it is not likely that a basic incompatibility between hardware and software would pass unnoticed through user testing explicitly calling attention to this issue will reduce the risk that a more subtle mismatch may go unnoticed. In this case, it is plausible that no one would otherwise think to validate the OTS driver software package for a COTS hardware product, and as driver failure has a strong potential to affect system safety, this compatibility is likely to be highly important.

The *Software Safety Guidebook* offers a number of specific recommendations regarding the acquisition process for COTS software. First, it specifies that all COTS software residing on the same system as a safety-critical application needs to be carefully examined and, if necessary, partitioned off from the resources used by the safety-critical application—regardless of whether or not the COTS software itself will be used within a safety-critical context (p. 272). The *Guidebook* further specifies that, if possible, non safety-critical software should not be included on a safety-critical platform *at all*, in order to manage the potential risk of interference at the

highest level (p. 272). However, this rule obviously does not apply for COTS software that will be used in the implementation of safety-critical systems. In order to deal with this case, the NASA *Guidebook* offers a number of suggestions for analyzing the potential candidates for inclusion and determining which software product is most likely to be appropriate for the application required. The first suggestion is that the IEEE Standard for Software Safety Plans (IEEE 1228) should be integrated into the analysis of the COTS software package. Vendor and package-specific recommendations include discussion of the stability and accessibility of the software package and the vendor as a whole (NASA, 2004a, p. 272). For example, the checklist addresses the niche that will be filled within the system, the responsiveness of the vendor, and the user base of the software—as well as a wide variety of other technical and operational requirements for the effective use of the software (p. 272). By using these criteria to assess and evaluate the software *prior to* engaging in a formal evaluation of the product, a PM is likely to save a great deal of time and effort in terms of determining overall viability of the software package.

In addition to evaluation of the COTS software itself, there is also the problem of integration of COTS components into the system. The NASA *Guidebook* (2004a) also addresses this issue, discussing details of implementation and testing of glueware, firewalls, and of the composite COTS-glueware system (p. 277). One particular problem noted in NASA's text is the issue of extra functionality or dormant code within the COTS code base. It states, "The more dormant code there is in the OTS software, the more likely it is to 'trigger' accidentally" (p. 277). The *Guidebook* offers a number of technical implementation details regarding this stage of the documentation, but the main point of the discussion is that, in addition to testing the COTS software in isolation, PMs must also evaluate the system as a whole.

NASA's text identifies a number of specific tests that should be performed both for COTS software in isolation and for the system as a whole; as the *Guidebook* notes, if they are performed for the software package, they should be performed again following integration of the COTS software into the system (p. 280). These

tests include software fault tree (including faults and dormant code in the COTS software); timing, sizing, and throughput tests; interdependence and independence analyses; design constraint, code interface, and code data analyses; and interrupt and test coverage analyses (pp. 280-281). By using these tests both throughout the system and specifically focused on the COTS software, PMs can help to prevent the potential for negative interactions between COTS components and the remainder of the system.

Safety throughout the Lifecycle

It is important to note, however, that this checklist and the notes above are appropriate only for initial implementation. In order to provide guidance throughout the software lifecycle, the *Software Safety Guidebook* also addresses issues of software operations and maintenance. COTS software is subject to rapid changes and upgrades from the supplier in order to fix bugs, add or remove functionality, adapt to changes in the underlying hardware, or deal with changes in other components (such as operating system patches) that change the operational environment of the COTS software (p. 200). In some cases, the analysis of these changes can be even more complex than the initial system configuration, and may involve a more thorough examination of the software. Some of the components that may have changed during a software update include the API, interface to glueware, functional details (additional functionality or removed functionality), interfaces to hardware or software already in place, required upgrades (such as memory increases), the way in which the upgrade will be performed, the potential to test the upgrade prior to implementation, and a number of other differences (p. 200). NASA's suggestion to COTS upgrades states, "The first and best choice regarding COTS upgrades is to ignore them. If you do not change that software, nothing in your system needs to be changed" (p. 202). However, the *Guidebook* does recommend acquiring the software if no upgrades are going to be planned by the vendor, in order to guard against the potential of COTS software obsolescence (p. 202). This may become relevant if future changes to the operating environment require additional changes to the COTS component to keep it safe. The *Guidebook* also recommends

strict configuration management control in order to allow for reproduction of previous configurations. It states that this management should extend not only to COTS software packages, but to all supporting compilers, libraries, source code bodies, kernels, and other software structures (p. 202). This will enable reconstruction of older systems if they turn out to be needed for a rollback to a stable older version, as well as ensuring that the overall implementation of the system remains consistent and as initially planned.

THIS PAGE INTENTIONALLY LEFT BLANK

Application of the NASA *Software Safety Standard* to the Naval Weapons System Program

The above discussion regarding the application of the NASA *Software Safety Standard* to the Naval weapons systems development program does not uncover any significant roadblocks for integration. It is likely that the program will require some degree of modification in order to be consistent with existing documentation and logging requirements, and that other requirements will be implemented on a project basis. However, the basic elements of the NASA *Standard* are sufficient for providing an inital systematic approach to risk analysis for use by the Naval development system. The table below addresses specific requirements of the *Software Safety Standard* (derived in relation to COTS software components) and discusses how they may be implemented within an existing system. The most important element of this *Standard* for the use of COTS software is the OTS software safety checklist (Appendix A). On this checklist, the first three items are essential for projects with potentially life-threatening implications, while the remainder of the checklist refers primarily to areas of the software that are still critical, but will not directly affect the outcomes of life-threatening hazards (p. 360). However, the recommendation of this report is that PMs should consider all items on this checklist, and should garner expert opinion on the ramifications of the software's risk-assessment profile.

Document	Brief Description
Software Safety Standard	Provides the general terms of NASA's standard and briefly identifies process-related implementation details
Software Safety Guidebook	Provides technical and process guidance on software safety assurance
Checklist for Off-the-shelf (OTS) Items	Provides a clear risk assessment checklist for OTS software (including COTS, GOTS, MOTS, etc.) that outlines required software safety decisions.
IEEE 1228	Standard that designates the content of Software Safety Plans, including specific requirements for previously developed (or reused) software and COTS and other acquired software

Table 1.   Index to Supporting Documents

There are no specific implementation details within this *Software Safety Standard* that must be changed or modified. However, the Standard may be extended as needed to support additional consideration of specific Naval safety requirements.

Future Research

One of the main issues with the current implementation of the NASA *Software Safety Standard* is that it provides little clear guidance on methods for identification of technical requirements for software safety or identification of safety-critical systems. In most cases, these choices may be clear (as there are specific situational guidelines that identify the areas in which software should be incontrovertibly safety-critical). However, in other cases, there may be a more subtle approach to determine the safety-critical features of the software system. In this case, there is no specific method specified by the standard to identify safety-critical features. Instead, the document recommends the use of skilled project management personnel and safety engineering personnel to provide insight into how and when given software applications are likely to require safety-critical design. By combining expert oversight into the design of the eventual system, with the use of the clear cut requirements laid out by the *Standard,* a PM will best meet the requirement for software safety in critical systems.

A second problematic issue is organizational support for the software safety process. The *Standard* notes that internal organizational support for software safety—including adequate assignment of resources for requirements determination, evaluation, testing and other needs—is essential to ensure software safety in safety-critical systems (NASA, 2004b, p. 20). This includes not only organizational support for the project itself, but organizational support and authority given to project management and technical leaders to ensure that the software safety demands of the given systems are taken into account (p. 20). However, obtaining such support may be problematic in a setting that does not assign this level of authority to the project managers and others involved in the safety determination. As this may be the case with some Naval software development processes, this should be addressed before applying the Standard to specific Navy safety assessments, since the *Standard* is highly dependent on individual authority and management of safety issues and responses.

THIS PAGE INTENTIONALLY LEFT BLANK

Recommendations and Conclusion

The overall approach of the NASA *Software Safety Standard* to COTS software component integration is much stricter and more conservative than the current Navy approach to this integration. In the *Standard*, COTS software integration is noted as being risky and requiring intensive scrutiny and increased oversight, as well as justifying potential concessions—such as gaining access to the source code and maintaining the source code in a fixed state (not implementing upgrades or bug fixes if the issue is not deemed to be a problem within the system). The *NASA Guidebook*, intended to guide implementation of the system, is considerably more conservative—stating that in most cases it is actually best to not use COTS software, and that if it must be used, it requires considerable oversight into safety, security, and configuration management. However, the actual technical treatment of the use of COTS software within the *Software Safety Standard* is very strong. It includes specific technical details for risk management and control over software quality and configuration that can be used to ensure that, if a COTS component or package is integrated into a safety-critical system, it can be effectively managed without negative consequences for the remainder of the system.

However, although the NASA approach to COTS integration is commonly more conservative than the Naval approach, there are a few recommendations that PMs should consider in order to implement a full safety-critical risk-prevention system. The first of these is that serious consideration should be given to the concept that non safety-critical software should not be installed on safety-critical hardware systems or integrated into safety-critical software systems *at all*. Although the use of partitioning or firewalling methods can help prevent negative interactions between the software, it cannot prevent negative software-hardware interactions from affecting the functionality of the system as a whole. For example, if a commercial software package crashed and forced a reboot of a safety-critical system, it would be difficult to shield the safety-critical portions of the system from this reboot. Thus, the complete isolation of non safety-critical software packages

from safety-critical situations is the best way to ensure that these negative interactions do not happen. The third recommendation is that not only COTS software components, but operating systems, component packages, compilers, libraries, SDKs and languages should be considered to be COTS software, and treated as such for the purposes of analysis, integration, and version and configuration control. This classification will prevent a number of potential mishaps from occurring that could negatively affect the software package. Finally, the use of software version control and configuration control, in addition to gaining access to the software component code base if at all possible, poses a significant potential for dramatically improving the ability to maintain, control, and ensure the safety of the COTS software integrated into safety-critical systems. This control should be considered a basic operational standard, as it will provide a clear-cut way to control changes within the COTS codebase and make it possible for system developers to correct defects if necessary.

List of References

Department of the Navy (DoN). (2008). *Naval open architecture.* Retrieved August 31, 2009, from https://acc.dau.mil/CommunityBrowser.aspx?id=18016&lang=en-US

Department of the Navy (DoN). (2009, January 15). Naval open architecture strategy.. *Naval open architecture.* Retrieved August 31, 2009, from https://acc.dau.mil/CommunityBrowser.aspx?id=129676&lang=en-US

Friedman, N. (2006). *The Naval Institute guide to world naval weapon system*s (5th ed). Washington, DC: Naval Institute Press.

Myers, G., Badgett, T., Thomas, T. & Sandler, C. (2004). *The art of software testing.* NY: John Wiley and Sons.

National Aeronautics and Space Administration (NASA). (2004a). *NASA software safety guidebook.* Technical Standard. Washington, DC: Author.

National Aeronautics and Space Administration (NASA). (2004b). *Software safety standard.* Technical standard. Washington, DC: Author.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A. Forms and Checklists

Checklist for Off-the-shelf Software

This checklist is included in the *NASA Software Safety Guidebook*. It is intended to support in all OTS acquisition processes as a means of risk assessment and risk management, and should be applied with this aspect of the process in mind. The first three items are mandatory for applications in which there are potentially life-threatening hazards (NASA 271). (The NASA *Guidebook* has a more detailed version of this checklist that includes specific technical details; this more-detailed version should be referred to for operational use).

No.	Items To Be Considered	Does it Apply? (Yes/no)	Planned Action
1*	Have the vendor's facilities and processes been audited?		
2*	Are the verification and validation activities for the OTS appropriate?		
3*	Can the project maintain the OTS independent of vendor support?		
4	Does software contain interfaces, firewalls, wrappers, etc?		
5	Does the software provide diagnostics?		
6	Any key products influencing choices?		
7	Has the software vendor been used before?		
8	Is this the initial version?		
9	Have competitors been researched?		
10	Is the source code available?		
11	Are industry standard interfaces available?		
12	Has the product research been thorough?		
13	Is the validation for the OTS software driver package available?		
14	Are there features that will not be used?		
15	Have tools for automatic code generation been independently validated?		

16	Can previous configurations be recovered?		
17	Will a processor require a recompile?		
18	Has a safety impact assessment been performed?		
19	Will the OTS tools affect safety?		
20	Is the OTS being used for the proper application?		
21	Is there compatibility between OTS hardware and software?		
22	Does the vendor have ISO certification?		
23	Does the vendor receive quality products from its suppliers?		

2003 - 2009 Sponsored Research Topics

Acquisition Management

- Acquiring Combat Capability via Public-Private Partnerships (PPPs)
- BCA: Contractor vs. Organic Growth
- Defense Industry Consolidation
- EU-US Defense Industrial Relationships
- Knowledge Value Added (KVA) + Real Options (RO) Applied to Shipyard Planning Processes
- Managing the Services Supply Chain
- MOSA Contracting Implications
- Portfolio Optimization via KVA + RO
- Private Military Sector
- Software Requirements for OA
- Spiral Development
- Strategy for Defense Acquisition Research
- The Software, Hardware Asset Reuse Enterprise (SHARE) repository

Contract Management

- Commodity Sourcing Strategies
- Contracting Government Procurement Functions
- Contractors in 21st-century Combat Zone
- Joint Contingency Contracting
- Model for Optimizing Contingency Contracting, Planning and Execution
- Navy Contract Writing Guide
- Past Performance in Source Selection
- Strategic Contingency Contracting
- Transforming DoD Contract Closeout
- USAF Energy Savings Performance Contracts
- USAF IT Commodity Council
- USMC Contingency Contracting

Financial Management

- Acquisitions via Leasing: MPS case
- Budget Scoring
- Budgeting for Capabilities-based Planning
- Capital Budgeting for the DoD
- Energy Saving Contracts/DoD Mobile Assets
- Financing DoD Budget via PPPs
- Lessons from Private Sector Capital Budgeting for DoD Acquisition Budgeting Reform
- PPPs and Government Financing
- ROI of Information Warfare Systems
- Special Termination Liability in MDAPs
- Strategic Sourcing
- Transaction Cost Economics (TCE) to Improve Cost Estimates

Human Resources

- Indefinite Reenlistment
- Individual Augmentation
- Learning Management Systems
- Moral Conduct Waivers and First-tem Attrition
- Retention
- The Navy's Selective Reenlistment Bonus (SRB) Management System
- Tuition Assistance

Logistics Management

- Analysis of LAV Depot Maintenance
- Army LOG MOD
- ASDS Product Support Analysis
- Cold-chain Logistics
- Contractors Supporting Military Operations
- Diffusion/Variability on Vendor Performance Evaluation
- Evolutionary Acquisition
- Lean Six Sigma to Reduce Costs and Improve Readiness

- Naval Aviation Maintenance and Process Improvement (2)
- Optimizing CIWS Lifecycle Support (LCS)
- Outsourcing the Pearl Harbor MK-48 Intermediate Maintenance Activity
- Pallet Management System
- PBL (4)
- Privatization-NOSL/NAWCI
- RFID (6)
- Risk Analysis for Performance-based Logistics
- R-TOC AEGIS Microwave Power Tubes
- Sense-and-Respond Logistics Network
- Strategic Sourcing

Program Management

- Building Collaborative Capacity
- Business Process Reengineering (BPR) for LCS Mission Module Acquisition
- Collaborative IT Tools Leveraging Competence
- Contractor vs. Organic Support
- Knowledge, Responsibilities and Decision Rights in MDAPs
- KVA Applied to AEGIS and SSDS
- Managing the Service Supply Chain
- Measuring Uncertainty in Earned Value
- Organizational Modeling and Simulation
- Public-Private Partnership
- Terminating Your Own Program
- Utilizing Collaborative and Three-dimensional Imaging Technology

A complete listing and electronic copies of published research are available on our website: www.acquisitionresearch.org

THIS PAGE INTENTIONALLY LEFT BLANK