Source of Acquisition
NASA Johnson Space Center

NASA'S SOFTWARE SAFETY STANDARD

Christopher M. Ramsay

NASA, Johnson Space Center, Space Shuttle Safety Division, 2101 NASA Parkway, Houston, TX, USA
Email: Christopher.m.ramsay@nasa.gov

ABSTRACT

NASA (National Aeronautics and Space Administration) relies more and more on software to control, monitor, and verify its safety critical systems, facilities and operations. Since the 1960's there has hardly been a spacecraft (manned or unmanned) launched that did not have a computer on board that provided vital command and control services.

Despite this growing dependence on software control and monitoring, there has been no consistent application of software safety practices and methodology to NASA's projects with safety critical software.

Led by the NASA Headquarters Office of Safety and Mission Assurance, the NASA Software Safety Standard (STD-1819.13B) has recently undergone a significant update in an attempt to provide that consistency.

This paper will discuss the key features of the new NASA Software Safety Standard. It will start with a brief history of the use and development of software in safety critical applications at NASA. It will then give a brief overview of the NASA Software Working Group and the approach it took to revise the software engineering process across the Agency.

1. A HISTORY OF SOFTWARE SAFETY AT NASA

Despite a growing dependence on software control and monitoring, until recently there has been little to no consistent application of standardized, Agency-wide, software safety practices and methodology to projects with safety critical software. NASA was formed in October of 1958 by legislation that, among other things, converted laboratories and facilities belonging to the previously exiting NACA (National Advisory Council on Aeronautics) over to NASA. These laboratories had a long history of conducting their own research and engineering practices, with little or no supervision from a centralized "headquarters" authority.

The Apollo Program used its onboard software and computers as the primary method for performing the critical rendezvous in lunar orbit, which was absolutely necessary to get the crew back to earth. (This served as a driver for the development of microprocessors that would fit within the small Apollo spacecraft.) However, the ground software, as well as the crew onboard served to back up the onboard computations. In fact, procedures existed to completely return to earth without the onboard computer. The rapid development cycle of Apollo also meant that much of the management for Apollo was at the "center" level. This did not allow for any real standardization across the Agency on software creation. In fact, it tended to centralize the expertise for producing such software within a select group of engineers and contractors. This led to a view of the software as a system within itself, rather than a part of the much larger Apollo system.

As the Shuttle Program came into existence, the role of software greatly increased, along with its complexity. Given the extreme aerodynamic environments in which the Orbiter is required to fly (from Mach 25 at 400,00 ft. to a glided touchdown), it became the first true "fly-by-wire" vehicle **designed** (NOTE: Other "fly-by-wire" vehicles were actually **built and flown** before the Orbiter). The crew would make their inputs into the software that would actually fly the vehicle. In the ascent phase, the software would be in direct control of the vehicle. Because of this, it was realized that the software and computers would be directly responsible for the safety of the vehicle.

The computer system for the shuttle was built with a large amount of hardware redundancy. Not only were there multiple computers to perform the same task, there were multiple data paths to provide commanding to the components. This concept of redundancy was also applied, in a more limited sense, to the software, with the development of a Primary Avionics Software System (PASS) and a Backup Flight System (BFS). The BFS was independently programmed by a totally separate vendor using different requirements running on the same platform. The concept was that this would protect for a "generic software problem" in the primary flight software.

Both of the Shuttle software systems were designed and built under very tight requirements management, design reviews, configuration control, and testing processes. By the late 1970's similar processes had been used quite successfully for the development of complex hardware systems. Their rigorous application to software was rather revolutionary for that timeframe. The management of the Shuttle Program saw no reason why they could not be used for the development of complex software systems as well. This "process-driven"

approach to the management of software worked quite well for the shuttle. In fact the IBM flight software organization was one of the first to achieve a CMM Level 5 rating because of their excellent processes. However, as with Apollo, there was no standardization of these processes from the headquarters organization for use across the Agency.

Meanwhile, academia and private industry started stepping forward with procedures and standards for safety critical systems and software. For example Dr. Nancy Leveson's book *Safeware: System Safety and Computers*, published in 1995, was one of the first to recognize that software was a part of overall **system** safety. Dr. Leveson's approach was to analyze the system as a whole and determine what hazards were either caused or mitigated by software. Specific software requirements would then be developed to mitigate or control these hazards. This would lead to a **traceability** between the system hazard, and the software requirement[s] that would control it. This approach put software as part of a "top-down" system safety analysis, thereby ensuring that it would not be considered in isolation from the other parts of the system which it controlled and/or monitored.

The first NASA Software Safety Standard, (STD-8719.13a) was published in 1997. This was the first NASA effort at a standardized methodology throughout the Agency to look at software as a contributor to safety at a "system" level. Unfortunately, this product was widely ignored by the software development organizations across the Agency. This was due to its complexity and poor organization. It also focused on concepts rather than definite procedural requirements organized around a software project lifecycle.

2. THE SOFTWARE WORKING GROUP

In 2002, the Chief Engineer's Office at NASA Headquarters chartered the Software Working Group (SWG). With designated representatives located at each center, its purpose "is to develop and oversee the formulation and implementation of an Agency wide plan to work toward continuous, sustained software engineering process and product improvements in NASA" [1]. Among its functions are to: "Establish guidelines and requirements for the development of software in NASA.", and to "Recommend, draft as requested, review, and promote software lifecycle management, engineering, and assurance...standards...to the Agency." [1]. This charter made the NASA Chief Engineer, Chief Information Officer, and the Associate Administrator of Safety and Mission Assurance "responsible for jointly promoting software policies, [and] standards...in their respective areas of responsibility." [1] This Charter provided the motivation and guidance for the complete revision of the way that NASA developed and managed its software programs, which included the NASA Software Safety Standard. It also, for the first time in NASA's history, put the development of software strategy and procedures to the Headquarters level for greater visibility and oversight.

In order to meet this challenge, the NASA Chief Engineer's Office implemented the *NASA Software Engineering Initiative Implementation Plan.* This plan created 4 "strategies" in order to meet the objectives of the SWG [2]. These strategies address the major aspects of software development and management at NASA. "Strategy 2" has as its goals to "improve safety, reliability, and quality of software through the establishment and integration of sound software engineering principles and standards."[2] Given this focus on "safety, reliability, and quality", it became natural to see this strategy as "software assurance", and to place it under the control of the NASA Safety organization.

In order to meet the challenges of Strategy 2, the NASA Office of Safety and Mission Assurance established the Software Assurance Working Group (SAWG). The SAWG is a sub-group of the SWG that focuses on the challenges of the many disciplines related to software assurance. These disciplines are best represented by the "umbrella" concept shown in Fig. 1.

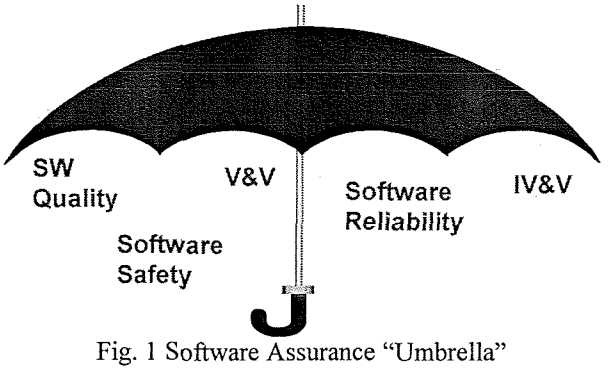

Fig. 1 Software Assurance "Umbrella"

This figure shows the major sub-disciplines that are part of software assurance and thus fall under strategy 2. As can be seen, Software Safety falls under this umbrella of assurance, and thus strategy 2 had the responsibility for revising the software safety standard, as well as the standards for assurance, and engineering. To cover this umbrella of differing disciplines, the SAWG established a series of documents. These documents could be arranged in a "tree" fashion. The higher documents on the tree would be the ones that invoke the requirements or procedures of the lower level documents.

The portion of this tree that pertains to the software engineering and assurance processes is shown in Fig. 2. The documents with the "♦"symbol (NOTE: NPD =

NASA Program Directive, NPR = NASA Program Requirement) are actually under the control of the NASA Chief Engineer's Office. However, they directly call out the "8700" series of documents that contain the standards and procedures for "Safety and Mission Success". This shows the relationship established between the engineering processes and safety and mission assurance at the NASA Headquarters level.

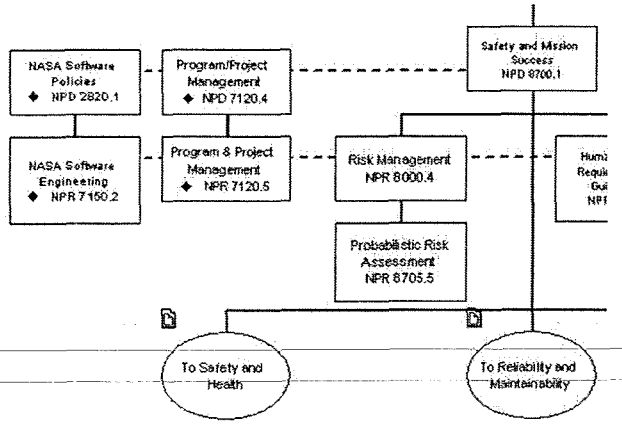

Fig. 2 NASA Safety and Mission Assurance Requirements Tree (Partial) [3]

The structure of this tree would also seem to require that the NPD's and NPR's be developed before the standards. This was not the approach taken by Strategy 2. Instead, there was a parallel effort to develop the standards and the higher level documents that invoked them. In fact, the standards were completed before the higher level documents.

3. DEVELOPMENT OF STD-8719.13B

In order to develop a new Software Safety Standard, (STD-8719.13B) the leader of Strategy 2 pulled together an Agency-Wide team of software safety experts. Originally, there were representatives from each NASA center on the team with a team leader. This team established the objectives for the update, along with an outline for the standard. These objectives were based on what were seen as the major weaknesses of the existing standard. In summary these were:

- More clearly define the requirements (i.e. the "shalls") for the definition, creation, testing, documentation, procurement, and operational use of safety critical software within NASA.
- Define personnel ***functions*** that are necessary to manage a safety critical software project.
- Bring the standard more in line with the current philosophy and thinking of the software safety community in the rest of government, academia, and private industry.
- Document the good practices that are currently being performed, or *should have been performed*, by safety-critical software projects across the Agency.

It was then decided that in order to write the actual text of the standard the process would work much better if the team was reduced to a "core" team of 4 members that would actually put the standard together. The other representatives would then be brought back for the later stages of development of the standard to get comments and "buy-in" from the software practitioners at their respective centers.

During the development and approval, it was made clear to the centers that this standard would be a "living document" and could be expected to go through significant revisions as the Agency learned more about the development and use of software in safety critical systems. Also, great care was taken to make sure that the document would not conflict with any "local" (i.e. center-specific) standards or procedures. This was done by focusing on *what* to do rather than *how* to do it. These "how-to" questions would be deferred to an accompanying software safety "guidebook".

4. MAJOR FEATURES OF THE UPDATE

The following sections of this paper will briefly discuss the major features of the new standard. In summary these are:

- It was completely reorganized into a "requirements based" document.
- It clearly states what constitutes "safety critical software".
- Provides a clear set of organizational and personnel *responsibilities* and *functions* that must be performed for safety-critical software.
- It spells out the linkage between software safety and software assurance.
- It contains requirements for "off-the-shelf" software that is used in safety-critical applications.

In addition to the above, the new standard was accompanied by a "tri-fold" brochure that contained easy reference material for managers and practitioners to implement the requirements of this standard.

5. SOFTWARE SAFETY PROCESS OVERVIEW

As mentioned above, a major goal of the new standard was produce a "requirements based" document. The emphasis is on stating, by use of "numbered shall statements", what *must* be done for safety critical software projects. This made the standard a process-oriented document as compared to the previous version of the standard which had a more philosophical

approach. Some introductory paragraphs included for explanations and guidance at the start of each major section.

At a high level, the software safety process required by the standard breaks down as follows:

First of all, there is an analysis and determination of software's contribution to the safety of the *system*. The emphasis at this point is on the entire system, not just the software. This then leads to the creation of software safety requirements. These are created from system Hazard Reports and other generic software and system safety sources. The Standard further defines them as "new, or ... existing, software requirements necessary to mitigate or resolve any hazards where software is a potential cause or contributor, or enable software to be used as a hazard control."[4] At this point, the project needs to provide a traceability mechanism "between software safety requirements and system hazards, as well as trace the flow down of software safety requirements to design, implementation, and test." [5]

Analysis is required throughout all of the stages of the software project's life cycle: Requirements, design, implementation (coding), and test. A critical part of this analysis activity is to provide a closed-loop feedback between this analysis activity and the overall *system* design. This is to ensure that any new system-level hazards that may involved software are properly mitigated.

As the software nears its operational phase, "there shall be an official certification process established, documented, and conducted prior to the release of any safety-critical software for its intended operational use. [and] Center Safety and Mission Assurance software safety personnel shall participate in the certification process." [6] All of these activities required by the Standard shall be thoroughly documented. Additionally, these activities "shall continue to be applicable after the safety-critical software has been released for operations. The software safety requirements to specify, develop, analyze, and test safety-critical software, shall apply to all changes made to the software or routine operational updates (e.g., mission specific database updates)." [7]

Further, the standard makes it clear that all of the requirements of the standard will be implemented or formally waived. It does provide a description of the waiver process that can be followed if a project cannot meet one or more of the requirements of the standard. During the creation process for the standard, there was a significant amount of discussion about allowing the specific requirements of the standard to be "tailored" by a project. The previous version of the standard had allowed such tailoring of requirements. This led to the perception that the standard lacked authority to be enforced. Because of this, it was decided to address the topic of tailoring at the very start of the standard. This is what the standard says regarding tailoring:

> "While the requirements of this Standard cannot be tailored, the specific activities and depth of analyses needed to meet the requirements can, and should, be tailored to the *software safety risk*. That is, while the requirements must be met, the implementation and approach to meeting these requirements may and should vary to reflect the *system* to which they are applied. Substantial differences may exist when the same software safety requirements are applied to dissimilar projects." (emphasis supplied by author) [8]

This means, for example, that small, low budget projects (e.g. a payload on the Space Station) do not need the same number of people to perform the tasks as a much larger and more complex project. They would not have the resources to hire a large staff of software safety experts to perform the required analyses and documentation. The standard makes clear that the *function* for the software safety personnel rather than the title or position. To help project managers understand this concept, an appendix was added to the standard showing how a smaller project can meet the requirements without "breaking the bank". As mentioned earlier in this paper, the goal of the standard was to focus on *what* to do rather than *how* to do it.

6 SAFETY-CRITICAL SOFTWARE DETERMINATION

In order to ensure the new standard had the proper application across the agency, there needed to be a clear, comprehensive, and coordinated definition of what constitutes safety critical software. To meet this need, an entire section of the new standard is devoted to "Safety Critical Software Determination". To start with, the standard states, "Until proven otherwise... all software within a safety critical system shall be assumed to be safety critical."[9] It then goes on to provide what has been referred to within Strategy 2 as "software safety litmus test":

"Software shall be classified as safety-critical if it meets at least one of the following criteria:

> a. Resides in a safety-critical system (as determined by a hazard analysis) AND at least one of the following apply:
>
> 1) Causes or contributes to a hazard.
>
> 2) Provides control or mitigation for hazards.
>
> 3) Controls safety-critical functions.

4) Processes safety-critical commands or data
5) Detects and reports, or takes corrective action, if the system reaches a specific hazardous state.
6) Mitigates damage if a hazard occurs.
7) Resides on the same system (processor) as safety-critical software (see note 4-2 below).

b. Processes data or analyzes trends that lead directly to safety decisions (e.g., determining when to turn power off to a wind tunnel to prevent system destruction).

c. Provides full or partial verification or validation of safety-critical systems, including hardware or software subsystems."[10]

In order to make sure this definition got the widest possible visibility, Strategy 2 made this decision to include it in the Software Assurance Standard, which is the controlling document for all software classifications.

7 CONCLUSION

The volume of software produced and used at NASA is enormous. Thus, the implementation of such a comprehensive set of procedures and requirements will not be an easy task. As of the writing of this paper, NASA has just completed a "gap analysis" of its software projects to see how much in (or out of) compliance they are with these requirements. Also, NASA has decided that most of the larger projects that were ongoing as of the publication of the Standard will be "grand fathered" out of having to comply with it. This includes the Space Shuttle and the International Space Station. As of this date, it is assumed that the new Crew Exploration Vehicle (CEV) will be in compliance.

The tasks and requirements of developing and maintaining safety-critical software are potentially expensive and time consuming. Many project managers have complained that this is too much for them to deal with for their particular projects. However, one does not have to research very hard to find examples of recent incidents where software has played a role in high-profile mission failures and hazardous incidents (e.g. the Mars Orbiter and Mars Polar Lander failures). In response to those who would question and complain about the effort and expense to develop this software I would like to close with a quote from Dr. Werhner von Braun in testimony to Congress after the Apollo I fire, "We are not in the business of making shoes".

RERENCES:

1. "NASA Software Working Group Charter, Responsible Office: AE/Office of the Chief Engineer (CE)", April 3, 2002. As copied from the SWG website: http://software.nasa.gov.

2. "NASA Software Engineering Initiative Implementation Plan Office: AE/Office of the Chief Engineer", January 11, 2002. URL: http://swg.nasa.gov/docs/NSII_Plan_Signed-1-11-02-wo-bdgt-info_1.doc

3. NASA Safety and Mission Assurance Requirements Tree URL: http://www.hq.nasa.gov/office/codeq/doctree/qdoc.htm

4. National Aeronautics and Space Administration, Software Safety Standard, (NASA-STD-8719.13B w/Change 1), NASA Technical Standard, July 8, 2004. URL: http://standards.nasa.gov/ (NOTE: Site requires a long-on ID), Section 4.2.2

5. Ibid., Section 5.7.1

6. Ibid., Sections 5.14.1 through 5.14.2

7. Ibid., Sections 7.1 through 7.2

8. Ibid., Section 1.1

9. Ibid., Section 4.1.1.1

10. Ibid., Section 4.1.1.2