EXISTING AND EMERGING STANDARDS FOR SOFTWARE SAFETY

Cynthia L. Wright and Anthony J. Zawilski
The MITRE Corporation, McLean, Virginia

ABSTRACT
This paper describes several standards that deal with aspects of software safety. Four descriptive attributes common to such standards are defined and specific standards are contrasted using these attributes.

1.0 INTRODUCTION
The use of computers to perform functions and control processes in safety-critical systems continues to grow within many industries, such as aerospace, chemical, medical, defense, nuclear power, and transportation. Whether it is a computer in an air traffic control terminal, a microprocessor in a medical treatment device, a banking computer handling electronic funds transfers, or a nuclear reactor control system, computers are being used in systems whose operation could result in injury, loss of life, create environmental harm, or cause large financial loss. Standards that are used in the life cycle development of the safety-critical software for these systems should be ones that increase the probability of developing systems that operate at acceptable levels of risk.

Currently there is no consensus across industry sectors for developing safety-critical software. Each industry or application sector has developed or is developing standards to meet, from their perspective, unique needs. In some cases, it is argued that the standards are premature, that we do not have the technology or expertise in place for developing safety-critical software that can meet acceptable levels of risk. However, this argument does not prevent us from developing these systems, with or without appropriate technology.

Standards offer collective perspectives on procedures, practices, and techniques that have been used effectively in software development. They leverage the many "lessons-learned" from previous efforts, and provide a baseline from which to measure software development progress. This paper identifies and describes several existing and draft standards that deal with the management, analysis, design, and development of safety-critical software. It describes these standards according to their emphasis on the development process or the final product, how they classify safety-critical software, their industry domain, and their definitions of key terminology. It is hoped that this analysis will identify standards areas that need more work, and highlight consistencies or conflicts among existing and emerging standards.

2.0 IDENTIFICATION AND OVERVIEW OF STANDARDS
This paper identifies software safety standards, and standards addressing systems safety, quality assurance, and acquisition where software safety is an integral part. A major United States standard, Department of Defense DOD-STD-2167A, is not included. Although it identifies a requirement to perform safety analysis for requirements, design, and operating procedures, it does not otherwise address software safety. The standards identified in this paper do not comprise an exhaustive list, however, they provide a representation of the "state of the standard" in software safety.

This standard has been available for over eight years and is undergoing revision. ANSI-7.7-4.3.2 is industry
specific, and lists concrete safety requirements with a near even balance between software and hardware considerations. The standard itself is only 5 pages long, and a valuable 3 page forward sets the context. Its software requirements checklist and overview of verification and validation are both valuable.


This is one of a family of four Quality Assurance (QA) program standards prepared by the CSA. The standards are intended to address different application domains and custom or commercial software. It does not address safety as a separate entity; it concentrates on describing the procedures, plans and administrative responsibilities of a software QA program. It can be used as a skeleton for a software safety process, but it would require the user to define and implement the safety specific aspects of such a process.


Like its companion, this standard is not specific to software safety. Primarily it describes mechanisms that a user can implement to test for quality (and software safety) in an "after the fact" mode. It points out the need for tracking such information as what previously developed software may be embedded in a ready-made product.

European Space Agency (ESA) PSS-01-40, Issue 2, "System Safety Requirements for ESA Space Systems and Associated Equipment"

The ESA standard defines safety program and technical system safety requirements. It requires contractors to implement a formally controlled and documented safety program, including a system safety program plan and safety data files. Hazardous consequences are categorized as catastrophic, critical, marginal, and negligible. Identified hazards can be mitigated by elimination, design, and control. The contractor is required to establish a tracking and acceptance system for hazard categories of catastrophic through marginal.


This draft standard deals with the design, development, and implementation of software for industrial safety-related applications. Part I addresses software safety and the software life cycle, while Part II identifies methods for the design and assessment of safety-related software. Major elements of this standard include the definition of a software safety process, identification of five software safety integrity levels, description of software safety activities for the life cycle phases, and identification of methods for the design and assessment of safety-related software. The software safety process in WG9 assumes that a system hazard analysis has been performed, and uses the hazard analysis documentation to identify the software safety functions. The activities needed to implement the safety functions to achieve an appropriate integrity level are identified in the Software Safety Plan.


WG10 is a generic standard that can be used for design and assessment across application domains. It identifies a safety life cycle model that can be used for safety-related systems, regardless of the technology that is implemented. The standard introduces the concept of safety integrity levels, i.e., the likelihood of a safety-related system achieving the required functions under the stated conditions within a stated period of time. An acceptable level of risk for each specific application is used to select the appropriate safety integrity level.


This document is in development, and has not yet reached draft stage. The charter of the P1228 Working Group calls for the creation of a standard for software safety plans. It recognizes the need for development of safety critical software with mechanisms distinct from and more specialized than the traditional quality assurance practices. The P1228 document defines and describes the mandatory elements of a Software Safety Plan, and addresses the entire lifecycle from conception through retirement including integration with other software development activities. Both management and practice issues are discussed. P1228 also includes an Appendix with descriptions of tools and techniques. However, no implementation methods are specifically recommended. A more complete description of P1228 is found in [10].

Ministry of Defence (MOD) Interim Defence Standard 00-55, "Requirements for the Procurement of Safety Critical Software in Defence Equipment"

The Defence standard addresses procedures and technical requirements for the acquisition and
development of safety critical software for use in defence equipment. The components of the standard include safety management procedures (planning and controlling), software engineering practices, including methods and tools, and stages of the project life cycle. It requires an Independent Software Safety Assessor (ISSA) to independently assess whether the Safety Plan has been implemented correctly and certify that the requirements of the standard have been met. Required practices include: hazard analysis, safety integrity analysis, specification, design, techniques, and reviews, static code analysis, and verification and validation testing. The standard requires that safety critical software be specified using formal mathematical techniques and identifies as unacceptable the practices of using floating point arithmetic, object code patching, and assembly level programming languages.

MOD Interim Defence Standard 00-56, "Requirements for the Analysis of Safety Critical Hazards". The Defence standard establishes mandatory techniques and procedures for hazard analysis to be applied to defence equipment. It categorizes accidents in terms of severity and identifies risk classification of hazards in terms of probability. Mandatory major hazard analysis activities are described for specific risk classes. Severity categories (catastrophic, critical, marginal, negligible) are used with the probability classes (frequent, probable, occasional, remote, improbable, incredible) to identify risk classes (A, B1, B2, C). The standard introduces the concept of safety critical features, i.e., the functional and nonfunctional properties of the system that remove Class A risks and limit Class B1 and B2 risks to acceptable levels.

Draft NATO Standardization Agreement 4404, "Safety Design Requirements and Guidelines for Munition Related Safety Critical Computing Systems," March 1989. This NATO effort is an example of a standard targeted to a single application domain and as a result focuses down at the tool and technique level. It includes elements such as: a Safety Kernel (an independent piece of software that monitors the system for unsafe state), and a Safe State (to which the system can switch when a safety fault is detected). The standard lists techniques such as positive completion returns, unique internal flags, single entry point code, and watchdog timers.


By the time this paper goes to press, 882B may have been superseded by 882C, now a working draft for Software System Safety. The Appendices in 882B are notable for a summary of the Software Hazard Analysis and the relation of safety to the system lifecycle. 882B contains a section of carefully constructed definitions of the major safety terms, safety-related personnel requirements, and lists of safety program features. Program tasks for safety are clearly defined. The draft 882C incorporates a series of tasks specifically for software systems safety.

United States DoD, "System Safety Standard for Space and Missile Systems", MIL-STD-1574, 15 August, 1979. This standard is a fairly complete look at systems safety with an emphasis on military systems. Safety plans, management and organization are described from the systems and integration level, but software safety is specifically addressed in two short sections - 5.2.6 (Software safety analysis) and 5.2.7 (Integrated software safety analysis). The emphasis is on command and control functions running from arithmetic calculations to the human interface with the analysis geared towards finding "errors." This contrasts with a proactive software safety effort that might originate with the software design. The software analysis report acts as input to the systems verification and validation task.

United States Air Force (USAF), "The Nuclear Surety Design Certification Program for Nuclear Weapon System Software And Firmware", AF Regulation 122-9, August, 1987. This regulation describes the safety certification processes for nuclear weapon software and hardware. It mandates a fully independent verification and validation (IV&V) activity as well as a separate Nuclear Safety Cross Check Analysis. Safety goals are stated as are the areas of concern (secrecy, unauthorized or inappropriate weapon use). Categories of software and software discrepancies are discussed and the roles and responsibilities of the chain of command are described. Attachments include a sample software discrepancy report, a nuclear safety objectives list, a generic nuclear safety certification plan and a generic IV&V plan.

USAF, "Nuclear Surety Safety Design Criteria for Nuclear Weapon Systems", AF Regulation 122-10, January, 1982. This is a companion to AF REG 122-9, but it approaches safety from a systems level. Software safety is only touched upon with a few paragraphs describing memory usage and "processor deviation" (e.g., branching into
critical code). The bulk of this document is concerned with issues such as how weapons-handling forklifts should be designed. It is valuable as context for AF REG 122-10.

3.0 ATTRIBUTE IDENTIFICATION

Four common attributes shared by software safety standards have been identified as a basis for discussion. These attributes are: an emphasis on either the software product or the development process, how safety-critical software is classified in the standard, the industry domain of the standard, and key definitions.

Process/Product: In developing safety-critical software, attention is often on the final software product. The concept of "product" also includes life cycle phase products such as specifications, and design documents. It is equally important to evaluate the process by which we build safety-critical software. The relative emphasis placed on how/why software is being built vs what is built defines the process/product attribute.

Classification of Safety-critical Software: Some standards provide a classification scheme for safety-critical software. The classifications are done with factors such as hazard, severity, and risk. These factors are used to identify software integrity levels for safety-critical software across a system. Techniques and appropriate level of resources are identified to meet the requirements of the specific integrity level.

Industry Domain: Standards are written for specific industry domains such as defense, medicine, transportation, manufacturing, space, and nuclear.

Key Definitions: Definitions of key terminology used across industry boundaries are important for implementation of a rigorous safety-critical software development process. Comparison of the definitions used in the current standards can highlight consistencies and identify areas that require additional research.

4.0 ATTRIBUTE DESCRIPTIONS OF THE STANDARDS

4.1 Process/Product

Although no standard discussed in this paper is purely a process or product oriented standard, each one does emphasize either the process or the product. This attribute can act as a guide for software engineers seeking specific types of information or ideas when developing safety-critical software. Some product oriented standards provide a checklist of features that should be included in a product or practices that should be avoided during the creation of a product. Process oriented standards examine one or more phases of the life cycle, and provide significant guidance on how to manage and structure the safety information flow through these phases. [1] The following orientation lists are based on emphasis, not total content, and are therefore open to debate.

Product Oriented: ANS-7-4.3.2-1982, ESA PSS-01-40, IEC SC65A WG9, MOD-00-56, STANAG 4404, MIL-STD-1574, AF REG 122-9, AF REG 122-10


As an example of the cross-over between orientations, consider ANS-7-4.3.2-1989. Although it is listed as a product oriented (nuclear power generating systems) standard, it does include a section on software development. However, this section is abbreviated and does not address specifics of a development process. Conversely, although AF REG 122-9 is targeted for nuclear weapons safety and requires procedures specific to that industry domain, it specifies the procedures and information flow for safety, not the techniques.

4.2 Classification of Safety-critical Software

This attribute refers to the type of scheme used in a standard to classify or categorize the criticality of software. For safety-critical software, special emphasis on procedures and practices is needed during the development process. The greater the criticality, the greater the need is to devote resources to these procedures and practices. A standard that has an explicit classification scheme can assist the software engineer in prioritizing tasks and allocation resources. Different standards have different philosophies of classification, and some (e.g., ANS-7-4.3.2-1982, CSA-Q396.1.1-89, CSA-Q396.1.2-89, and IEEE P1228) have no classification schemes because it is outside the scope of the standard. MIL-STD-1574. has no classification schemes, but it does require contractors to have a scheme for determining hazard level categories.

Of the standards examined for this paper, ESA PSS-01-40, MOD-00-56 (and its companion MOD-00-55 which references 00-56) and MIL-STD-882B all rely on a similar classification schemes for hazard and risk. They use the same five classes of frequency of occurrence (frequent, probable, occasional, remote, and improbable),
with MOD 00-56 adding a sixth class called incredible. These standards also use the same four classes of hazard criticality (catastrophic, critical, marginal and negligible). The risk classification is derived by pairing a frequency of occurrence with a hazard criticality. (This rating scheme would satisfy the requirement in MIL-STD-1574 for hazard level categories.)

IEC SC65A WG9 and IEC SC65A WGI0 do not directly classify the risk or the software. Instead they postulate five levels of system "integrity". An application may be composed of any mix of high or low integrity hardware and software as long as the final system integrity reaches the required integrity level. These standards do not, however, address how to rate or classify software to match it to an integrity level. Such gaps in software safety standards are not unusual. Since the formal basis of safety-critical software is still being researched, mandating specific techniques or tools for a generic standard is premature. The exception is for standards that are written for a well understood, specific application.

STANAG 4404 is a standard that addresses a specific application, and it is the only standard discussed here that defines a software classification scheme based solely on the software's function. The four classes (and two subclasses) create categories that fit all software from the most safety-critical (e.g., firmware that controls a nuclear trigger) to non-critical items (e.g., personnel records). The midrange classifications are used for items such as man-in-the-loop processes or safety information display software.

A totally different approach is taken in AF REG 122-9 and 122-10. There are three categories of software (I, IIa, IIb), but they are merely labels for software previously designated as critical or non-critical. The categories have different documentation and certification requirements. What sets these regulations apart from the others is a classification scheme for software discrepancies. Five classes of discrepancies (critical, urgent, degraded, noncritical and minor) are defined and the requirements for their resolution are outlined. This concern with flaws in the final software is part of the product orientation of AF REG 122-9 mentioned in Section 4.1.

4.3 Industry Domain

The standards identified in this paper fall into 4 industry domains, and a non-specific category:

Defense: MOD-00-55, MOD-00-56, STANAG 4404, MIL-STD-882B, AF REG 122-9, AF REG 122-10
Nuclear: ANS-7-4.3.2-1982
Space Systems: ESA PSS-01-40, MIL-STD-1574A
Industrial Control: IEC SC65A WG9

Note that some standards cross domain boundaries such as AF REG 122-9: a defense standard that deals with nuclear weapons, but has an appendix which is a generic nuclear certification plan. It might fit in either the defense or nuclear industry domains. The industry domain attribute should not be taken as a limiting factor when looking at standards. While a domain specific standard must include details for that domain, it may also be a resource for general software safety process information. In fact, it may prove that the process advocated by a standard in one domain leads to an improved software safety analysis in another.

4.4 Key Definitions

Using terminology that is precise and that conveys the same meaning, regardless of the industry domain, offers a universal capability that promotes technological understanding. By using technical terms that have the same meaning, research and software engineering advances in one industry can be communicated more easily to another. The tables in Section 4.4.1 compare key terminology used in existing standards. In many cases, the standards do not have definitions for the three key terms selected. This is because several of the standards reviewed here are not standalone software safety standards. They are either quality assurance that rely on cited references for safety terminology or domain specific standards that use terminology as it is generally understood by the specific community. When such standards are applied in part or in whole, the meanings assigned to key safety words become quite important. In this paper, the key terms of hazard, risk, and fault have been selected for comparison.

As can be seen in Table 4-1, hazard is defined in the general terms of a condition. The IEC WG10 standard limits the definition to physical situations. Each of these definitions imply that a potential for an accident exists.

For each of the standards, risk is described in Table 4-2 in terms of both probability and severity. While the standards use different words (probability/likelihood and severity/consequence), the definitions convey the idea
that risk must include both of these concepts. In addition, the standards describe risk as a measure or expression, i.e., an attribute or characteristic that can be measured.

Table 4-3 shows the different approaches taken to define the term fault. The IEC standards, WG9 and WG10, are consistent with each other in relating the definition of fault as a causal factor to error, but the nature of the error is not characterized in the definitions. The MOD standards provide a different perspective by relating fault to failure. In the case of MOD 00-55, the fault is limited to software, since this standard is specific to software safety.

4.4.1 TABLES OF DEFINITIONS

<table>
<thead>
<tr>
<th>Standard</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEC WG10</td>
<td>A physical situation with a potential for human injury.</td>
</tr>
<tr>
<td>MIL-STD-882B</td>
<td>A condition that is prerequisite to a mishap.</td>
</tr>
<tr>
<td>MIL-STD-1574</td>
<td>An existing or potential condition that can result in an accident</td>
</tr>
<tr>
<td>MOD 00-55</td>
<td>A condition that can lead to an accident.</td>
</tr>
<tr>
<td>MOD 00-56</td>
<td>A condition that can lead to an accident.</td>
</tr>
<tr>
<td>PSS-01-40</td>
<td>A source of potential threat to safety (danger).</td>
</tr>
</tbody>
</table>

Table 4-1
Definitions of Hazard

<table>
<thead>
<tr>
<th>Standard</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESA PSS-01-40</td>
<td>A measure of the magnitude of the threat to safety.</td>
</tr>
<tr>
<td>IEC WG10</td>
<td>The combination of the frequency, or probability, and the consequence of a specified hazardous event.</td>
</tr>
<tr>
<td>IEC WG9</td>
<td>The combination of the frequency, or probability, and the consequences of a specified hazardous event. The concept of risk always has two elements; the frequency, or probability, with which a hazard occurs and the consequences of the hazardous event.</td>
</tr>
<tr>
<td>IEEE P1228</td>
<td>A measure that combines both the likelihood that a system hazard will cause an accident and the severity of that accident.</td>
</tr>
<tr>
<td>MIL-STD-1574</td>
<td>Measure of vulnerability to loss, damage or injury caused by a dangerous element or factor</td>
</tr>
<tr>
<td>MIL-STD-882B</td>
<td>An expression of the possibility of a mishap in terms of hazard severity and hazard probability.</td>
</tr>
<tr>
<td>MOD 00-55</td>
<td>A measure of the severity and likelihood of an accident.</td>
</tr>
<tr>
<td>MOD 00-56</td>
<td>A measure of the severity and likelihood of an accident.</td>
</tr>
</tbody>
</table>

Table 4-2
Definitions of Risk

Other key terms not shown in this paper but that are important for software safety standards are: accident, critical function, and safety-critical software. In the literature for software safety, the term accident is often used synonymously with mishap. [5] Defining "critical function" is usually done in the context of an application, or as it relates to the terms hazard, accident and risk. Of more interest is the multiple definitions for the term safety-critical software. The two basic meanings are: 1.) Software that, through its improper operation, can lead to a hazardous state and 2.) Software intended to mitigate the severity of an accident. These different meanings reflect a proactive and a reactive view of safety. A third meaning is merely the combination of the first two.
The cause of an error is a fault (e.g., hardware defect, software defect) which resides, temporarily or permanently in the system.

A fault is a defect that gives rise to an error.

A software defect that can lead to a failure. It is the result of an error in development.

An imperfection or deficiency in the system which may, under some operational conditions, contribute to a failure.

### Table 4-3

**Definitions of Fault**

5.0 CONCLUSION

The importance of software safety continues to grow within the discipline of software engineering. Standards can be used to advance our understanding and improve the development process of safety-critical software. Comparison of the existing and emerging standards highlights commonalities and offers an overall perspective that can be used to guide the direction of future software safety standards efforts.

REFERENCES


