C

ertification is the “procedure by which a third-party gives written assurance that a product, process, or service conforms to specified requirements.”¹ These certifications should assure conformance with the applicable requirements for a given purpose, within specific operations scenarios.

Product certification focuses on specific, limited aspects and functions. For example, Y2K certification evaluates how effectively software will perform before, during, and after the date conversion, avoiding Y2K-related problems. In the aviation and nuclear domains, certification normally focuses on safety, or safety certifications, in which the requirements are based on safety objectives of the system. Following the above definition of certification, software certification should ensure that programs put on the market conform to a certain level of quality. ISO defines it as “the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs.”² Software-safety certification corresponds specifically to safety objectives.

Software Safety Certification: A Multi-domain Problem

Patricia Rodríguez-Dapena, European Space Agency
Because software is not unsafe itself, and only contributes indirectly to accidents, software safety problems must be evaluated within the context of system safety. This article focuses on software-safety certification, which will be considered as part of the overall system-safety certification.

In many application domains, safety certification is mandatory before a system is put in operation. Currently, this certification is the responsibility of national certification authorities, who cover all certification aspects for systems belonging to specific application domains. It is also based on well-known domain-specific technical and operational requirements, historical information about the reliability of the different subsystems, and historical accident data.

Decisions about the use of safety-critical systems often rely on an assessment of the risk involved. These risks can be derived from historical information about the reliability of the individual components and the models that define the connection between these components, or historical accident data about similar systems. Many new systems are composed of subsystems belonging to different application domains, and some are built to be used and operated internationally. The failure of these safety-critical systems can have catastrophic consequences to human lives. The definition of the safety objectives for these systems and the process and methods to ensure their safety are completely new, not proven before and being defined while the systems are being built.

**SYSTEMS THAT CROSS BORDERS**

The Global Navigation Satellite System uses satellites for navigation and automatic surveillance of air, maritime, and land systems. Safety certification of such a system is a complicated and largely uncharted area—no historical databases exist as a basis for the certification of the complex technical and interdomain safety requirements. Further, no certification authorities have yet been appointed due to the need for international and interdisciplinary legal recognition—an intensely political matter.

Similarly, remote telemedicine systems use satellite communication links for remote consultation, surgery, and other medical procedures. How do we perform safety certification on interdomain systems when we cannot easily define and verify a set of requirements? The different domains are still working in isolation, under different regulations and standards. Certification becomes even more difficult when one subsystem already exists and possible new regulations and standards do not apply to it.

To further complicate the situation, the proportion of subsystems containing software is increasing rapidly, with critical functions increasingly implemented using software. For example, computer-based devices govern many aircraft command control functions, and cars may have 15 or more microcomputers for monitoring, control, and navigation. Software failures have potentially severe effects. When software is constructed, it is usually specially constructed to one system, so no historical information exists about its reliability. This means defining safety constraints for the software requirements, design, and implementation, and using verification analysis methods. Such techniques are immature, but several standardization bodies and working groups have begun analyzing and addressing the problems of international, interdomain, safety-critical software-based systems, often as international collaborations (see the boxed text, “International Software Safety Initiatives,” on p. XX).

**LEGAL CERTIFICATION ISSUES**

Both producers and consumers increasingly recognize the need for certification. For producers, it implies more quality in the products and sometimes a reduction in their liability for defects. An example is limited liability when the products are used for other purposes. For consumers and system operators, certification schemes legally recognized by national governments secure protection for their investments, efforts, and safety operations.

In some countries, any person, association, or company can set up a certification business—credibility lies in commercial success rather than any legal mandate. In general, national governments recognize and accredit certification agencies for each application domain, particularly in domains where safety is essential, to provide more structured, effective supervision.
Certification is mandatory before a system is put in operation in some application domains. For example, every country certifies new aircraft to ensure compliance with airworthiness requirements before allowing them to fly. Up to now, we have been content with multiple domain-specific certification schemes, standards, and organizations through nationally recognized certification authorities. For example, the Federal Aviation Authority regulates both technical and legal certification of US aircraft and related products and parts; individual European Community member states have their own regulatory authorities for similar matters.

A legal problem arises when we build new systems from subsystems that come from different application domains, because no certification scheme exists yet for these interdomain systems. To define, agree upon, and apply a certification scheme for the overall system, we need agreements between the various certification authorities and application domains. In addition, if these new systems will be operated and used internationally, international agreements are also needed. Liability issues, international recognition of the certificate, certification authorities, and many other issues have not yet been defined for these new systems.

In addition, we need internationally recognized accreditation agencies, such as testing laboratories, to bolster the authority of safety demonstrations. Software poses a particular problem to existing certification schemes and accredited certification agencies—few accredited certification agencies have expertise in software safety issues, which are not generally well defined or understood. This problem is internationally recognized among those who have attempted to perform in-depth evaluations for new multidomain software-intensive systems.

ESA is currently working in various international committees intended to define and set the certification schemes for new complex software-intensive systems, such as GNSS. The European Committee for Standardisation (CEN) has created the Information Society Standardisation System (ISSS) with a working groups entitled, “Accredited Testing of Software in Safety-Critical Systems.” Its major goal is to ensure that the approval, certification, and evaluation process for safety-critical systems, in which a software failure could be a safety hazard, is handled within the EU to support the single European market. I hope more initiatives like this one will start in order to solve other legal issues that still remain open, such as liability, international certification authority, and international recognition.

**Technical Certification Issues**

The technical aspects of software safety certification present an even more difficult challenge. We must ensure that the product complies with the applicable technical requirements by following a process agreed upon by the producer and the certification organization. It must also be based on a set of applicable technical requirements which, for the system safety certifications, are based on safety objectives.

**Iterative Certification Process**

For existing systems, safety certification follows a known process, agreed on by the parties in advance, where the certification requirements are set up and progressively updated based on technology evolution and lessons learned. In the absence of a certification scheme or historical information for the new interdomain and software-intensive systems, the first certification of such systems will be based on validation of their new operation and performance requirements, and on the verification that no mistakes are made in the safety analysis of the system and in the construction of both the system and subsystems.

The process by which the certification authority and the applicant interact is often iterative, running while the system is being developed. Without this iteration—if the applicant intends to request certification at the end of system development—there is a high risk of incurring extra costs to achieve certification. Developing a system based on stringent requirements, standards, and plans might not be enough for system certification.

The iterative approach certainly offers a more cost-effective and secure way to obtain certification, but it is not always possible for every part of a system. Because COTS products are often not specifically developed to work as part of a safety-critical system, evidence rarely exists that they are robust and will not
introduce extra hazards into the system. Their safety analysis is performed after they are developed.

Figure 1 shows a sample iterative reference process for safety certification focusing on the software product development process. The process starts by defining the system safety requirements as a basis for the software safety certification, culminating in the safety demonstration. This safety demonstration can be defined in many different ways, depending on the method selected to document that the system is safe.

Certifiers and/or developers can tailor this reference process to their system's applicable certification requirements and methods. There are several methods for individual domain-specific safety demonstrations; the safety case approach is an example. However, no validated method yet exists for multidomain, international, software-intensive systems. Any method that system developers use will serve for pilot trials, and future certifications will validate them and aid in improving the safety assessments of these systems. In any case, the system safety demonstration will reuse information from the different safety analyses produced at each phase of the software development life cycle.

Because software can contribute to system hazards, eliminating or controlling hazardous software behavior will reduce or prevent accidents and, in turn, reduce risk. Software faults might be caused by errors at any software life-cycle stage. They could be introduced in the software requirements phase. For example, aviation software written for use in the northern hemisphere often creates problems when used in the southern hemisphere, or at altitudes below sea level. Faults could also originate from a later phase; the failure of the Ariane 5 first test flight was the result of a software variable that was not checked under software exception handling control—its overflow caused the catastrophic event. To prevent the failure, developers could have implemented a control mechanism for hazardous software behavior.

This suggests that safety verifications must be done in parallel to software development. Developers can derive design and implementation constraints from these safety verifications to apply to later software implementation phases. These constraints may vary depending on the criticality class of the software product and its nature, such as COTS versus custom software.

The sample process in Figure 1 should be adapted on a case-by-case basis. Developers can define different life-cycle approaches, such as the spiral model, JAD, and the iterative life-cycle model (as typified by object-oriented development); not all steps can be performed in every case. For example, the process in Figure 1 cannot be applied to software COTS products. Their certification process is...
INTERNATIONAL SOFTWARE SAFETY INITIATIVES

FAA-SSAC (http://shemesh.larc.nasa.gov/ssac/). FAA began the Streamlining Software Aspects of Certification (SSAC) program to address concerns about time and expense associated with software aspects of certification. The primary objectives are to

- analyze the current software approval process for certification and identify target areas for improvement,
- determine if the desired safety benefit justifies the expense burden, and
- if necessary, establish streamlined processes for software aspects of certification that are faster and less expensive than the current processes.

ISO/TC1/SC7 (http://saturne.info.uqam.ca/Labo_Recherche/Lrgl/sc7/index_e_frameset.html). This committee is responsible for developing ISO standards for software engineering. A liaison has been established with ISO/TC 176 and IEC/TC 56. Two of its related working groups are WG 9 Software Integrity and WG 6 Software Evaluation and Metrics.

IEC Technical Committee 56 (http://www.iec.ch/home.e.htm). This committee works on standards for availability, reliability, maintainability, and maintenance support in any technological areas considered appropriate, including those not normally dealt with by IEC Technical Committees. Some of its software-related working groups include

- TC 56/WG 10: Software Aspects, which develops several international standards related to dependability assessment of software (IEC 61704, IEC 61713, IEC 61714, IEC 61720);
- SC 65A/WG 9: Safe Software, which is responsible for the publication of IEC 61508, functional safety of electrical, electronic, and programmable electronic safety-related systems Part 3; and

Squale—Security, Safety, and Quality Evaluation for Dependable Systems (http://albion.ncl.ac.uk/squale/index.html). The objective of this Esprit project is to analyze the existing standards and practices in the safety and security area and to define a combined harmonized approach to gain confidence in the correctness and effectiveness of systems with safety and security requirements. The project will define “dependability criteria” describing this harmonized approach and apply these criteria to a demonstrator system.

Esprit Project 22187: Serene—Safety and Risk Evaluation Using Bayesian Nets (http://www.cordis.lu/esprit/src/22187.htm). The project’s objectives are to develop a method for constructing software safety arguments using BBNs, to adapt an existing BBN tool to support the method, and to evaluate the application of the method and tool through practical trials. The results are the provision of manual detailing procedures for identifying and structuring evidence so that a system meets the safety requirements of IEC 1508, the provision of a tool using BBN technology to automate the implementation of the method, and a quantified comparison of the performance of the proposed method and tool compared to conventional methods.

Information Society Standardization System (http://www.cenorm.be/issss). Created by the European Committee for Standardization, ISSS offers a workshop, “Accredited Testing of Software in Safety-Critical Systems.” This group’s goal is to ensure that the approval, certification, and evaluation process for safety-critical systems in which a software failure could be a safety hazard is handled within the EU to support the single market.

RTCA/EUROCAE SC-190/WG-52 “Application Guidelines for RTCA DO-178B (Software)” (http://forum.pr.erau.edu/sc190/). Special Committee 190 has formed a Joint Activity with EUROCAE Working Group 52 to develop a series of position papers that will provide consistent clarification of ambiguities and problems encountered in the application of RTCA DO-178B.

IEEE Safety Study Group instituted by the IEEE Software Engineering Standardization Committee, this group was created in November 1997 and is chaired by Victoria Stavridou. To become a member, contact her at victoria@cs1.sri.com.

The charter of the SSG is to:

- Examine the existing standards collection to discover places where safety requirements should be incorporated during the next revision cycle;
- Study the compatibility of existing IEEE standards with IEC standards;
- Track the development of IEC 61508 and recommend adoption, modification, or rejection;
- Recommend further actions; and
- Report back to SESC in 6-monthly intervals.

normally applied at the end of the software development process. It consists of checking the fulfillment of the derived system safety requirements and constraints for the software product, as defined in the first system development phases. Accepting the product as part of the system—or as part of the software development or system testing environment—depends on the remaining level of risk and added cost required to fulfill the necessary safety requirements and constraints.

The current international standards for software development do not clearly define this parallel safety process. ISO 12207 mentions safety only as requirements to be implemented in the software, but safety should be defined as a complete and visible parallel process. Other standards do identify...
safety explicitly, but either cover certification-only issues, such as Do178B for the certification of software in airborne systems, or belong to specific domains, such as the British Defense Standards.

The certification process shown in Figure 1 has specific aspects that should be analyzed in more detail. The relationship between system and software deserves special attention, since it is a subject of discussion among different organizations and standardization groups.

**System and software safety relationship**

The safety certification process starts at the system level: analyzing the system’s potential hazards and evaluating their probability and the severity of their consequences. Any project entails risks; risk management is gaining importance as a discipline and as an integral part of project management. Risk data forms the basis for multi-attribute decision making.

Risk management includes these two factors:

- systematically identifying and evaluating all risk causes and consequences prior to defining and implementing a decision to accept, monitor, or take action.
- systematically defining, implementing, controlling, and verifying actions for eliminating risk or reducing it to an acceptable level.

When the risk is too high, developers must either eliminate or control hazards. They may need to define protection devices as part of the system design. For example, they can add hardware interlocks or software safe states. An alternative is to define protection procedures for system usage—like defining what an operator should do when the red emergency alarm light is on—to control hazards in the system. Some of these system hazard controls can be in the software requirements. They could be defined as software safety requirements or as functional, performance, or any other kind of requirements including design and implementation constraints.

In many cases, developers perform a system-level hazard analysis at the system design stage to ensure they meet safety requirements. By applying the analyses to the final design, it should demonstrate that, for example, no single failure or double failure point can cause catastrophic consequences. If needed, developers can revise the system design to have completely redundant subsystems, a physically independent checking and monitoring subsystem, or some software implementation constraints, such as diversification.

Software requirements and constraints derived from system safety requirements can be applied to any software development phase. Examples include design constraints regarding software error handling, coding constraints related to the application of specific coding rules, and stress tests to demonstrate system safety. Methods for translating system safety requirements to software requirements are immature, and the issue is even more complicated when the systems are multidisciplinary.

A few international initiatives are trying to bring together the system and software safety fields and to define standard requirements and methods for software safety (for more details see the boxed text, “International Software Safety Initiatives,” on p. XX). These methods, requirements, and constraints are linked with the system safety criticality levels that apply to software products and are also assigned from the system safety analysis. Part of IEC 61508, a generic system safety standard from the International Electrotechnical Commission’s Technical Committee 56, is dedicated to software safety. Although IEC 61508 provides a general framework for the safety of software systems many technical details remain unresolved for multinational systems to operate effectively all around the world.

**Requirements engineering**

Requirements engineering is the initial step of the software life cycle. It consists of acquiring the various requirements for the software, including functional needs of potential users and nonfunctional concerns about quality, performance, costs, and so on; precisely specifying these requirements; and evaluating if they are correct and feasible. This step is critical in the software life cycle, because a variety of errors can be introduced at this stage that may negatively influence the subsequent development process and the quality of the resulting software product. Requirements engineering must be carried out with great care and precision since it is critical that the specification be correct later during the system certification. In most domains, developers specify system requirements textually, with no way to formally verify their correctness and completeness.

Recently proposed formal methods seek to help engineers capture and specify software require-
ments more precisely. These methods express requirements in a formal language, often based on mathematical logic equipped with suitable structuring mechanisms for organizing large, complex descriptions. Developers can then manipulate such specifications formally. For example, they can check desired properties and detect errors using automated theorem proving techniques or obtain executable prototypes using transformation techniques. Most current requirements engineering methods are limited, however, since they are highly restricted in scope and provide very little support for capturing nonfunctional requirements and information about the environment in which the software will run. More work is required in this area to ensure that certification is based on a complete and consistent set of requirements.

Design and implementation constraints
Safety- or mission-critical subsystems have long been designed with fault-tolerant mechanisms, redundancies, and other techniques covering inherited system-level safety requirements. Software fault tolerance is still a subject of research. For example, diversification, exception handling, or error recovery mechanisms require further validation of their usability and relationship with the safety integrity levels. This is particularly on the new software-intensive systems. Some existing standards require the use of specific design and coding constraints and, if carefully applied, can be the basis for accumulating historical data on the use of these methods in safety-critical systems. Some recommendations based on lessons from the Ariane 5 test flight failure were considered in defining the new set of European Space standards.

More effort is required in these areas to gain confidence about possible design and code constraints. Some international initiatives could share experiences and combine projects from different application domains. For example, how is N-version programming related to the different software integrity levels?

Verification methods
Because we cannot rely on historical data for safety verification of the software, developers should use other methods. The different analysis methods may vary depending on the software's criticality class, life-cycle stage, and nature. For example, COTS products require different analyses to either reject their use for a particular system or define boundaries that ensure their safety.

The safety analyses performed at each software development phase vary among standards and the different application domains. For example, formal arguments to demonstrate that the object code satisfies the formal specification in Def. Stan 00-554 might not be required by standards used in other application domains, or might be difficult to apply. As another example, DO178B specifies a difficult and expensive set of requirements to ensure that software code does not have run-time errors for all test methods proposed.

Software verification approaches fall into two main groups: dynamic and static. Existing standards use both to verify the software as part of the safety analysis shown in Figure 1. However, these verification techniques are not advanced enough in relation to the safety integrity levels needed for software. Formal verification techniques still have major drawbacks. First, they are not entirely practical. For example, showing consistency between the requirements and the code does not ensure confidence in safety since most safety problems stem from flaws in the requirements. Another drawback is feasibility, since the few formal verifications applied to real programs require massive effort for relatively small software, few properties can be verified using formal methods, and many formal methods are not readable by system engineers and application experts who should perform these analyses. System developers should use non-formal verification techniques to complement dynamic and formal verification methods for analyzing software-intensive safety-critical systems.

Only practical demonstrations can validate the usability of some of the verification methods. These demonstrations can address several issues:
- The efficiency of using sometimes-expensive, 100 percent statement test coverage versus complementary methods when reaching 80 percent statement coverage
- The application of fault tree analysis to software design and code
- Which metrics should be used to analyze the code, and their limits in relation to the integrity levels
How verification-oriented software development will improve final product quality

Some International Initiatives

All of the above technical aspects are being analyzed by several international standardization groups either individually or jointly. Both the Federal Aviation Authority and Radio Technical Commission for Aeronautics are working to improve the DO-178B standard for software considerations in airborne systems and equipment certification. An FAA survey described some of the problems encountered when applying the RTCA standard DO-178B. The following list is extracted from that FAA survey:

- DO-178B has inadequate and ambiguous guidance for requirement definition and analysis.
- DO-178B has inadequate and ambiguous guidance for partitioning (e.g. The growing importance of partitioning is recognized and questions like "what types of techniques are acceptable and what are the criteria to accept a partitioning strategy?" are still to be answered).
- DO-178B has inadequate and ambiguous guidance for verification activities (e.g. no clear direction on regression analysis, different interpretations of the applicability of coverage analysis techniques to different stages of verification).
- DO-178B has inadequate and ambiguous guidance for COTS software.
- DO-178B inadequately addresses the effect of software on the safety of the overall system.

RTCA has created several working groups to provide consistent clarification of ambiguities and problems encountered in the application of RTCA DO-178B. Working groups have been created to tackle DO178B details, such as the verification process and the process of incorporating COTS tools in critical application.

The IEEE software safety group is examining existing IEEE standards to identify areas where safety requirements should be incorporated during the standards’ next revision cycle. The group is also studying the compatibility of existing IEEE standards with IEC standards and is tracking the development of IEC 61508 with the ultimate goal of recommending its adoption, modification, or rejection.

Several separate European initiatives address dependability and other issues of safety-critical software-intensive systems. Find more information about these and other international initiatives in the boxed text, “International Software Safety Initiatives,” on p. XX.

The Future: Building Safe Systems

Many international initiatives have begun to analyze and improve software safety and certification, addressing both the legal and technical aspects. Major changes will be needed regarding the existing standards and methods to be used when building software that is embedded in multidomain systems. An international certification authority will emerge to certify the safety of the new systems. Identifying the parallel safety process for the software-specific technical aspects will affect the techniques and methods for development and testing, as well as software management planning, cost, organization, and development interfaces.

Despite the deficiencies discussed here, the state of international, interdomain safety certification is slowly improving. New systems that contain software-based safety-critical functions and are composed of varied subsystems have already been implemented without a mature certification scheme in place. The current state of the art does not provide answers to all the fundamental issues. We cannot wait for all the answers that will free us from accidents and losses before building and using these systems. “The benefits of continued development seem to outweigh the perceived risks.”

Acknowledgments

I thank several of my colleagues at ESA for their support and valuable comments on an earlier draft of this article, as well as my colleagues at the university.

References

3. SCOPE: Esprit 2151 (Esprit Second Framework Programme), Software Certification on Program in Europe, DGIII, European Commission, Brussels.

About the Author

Patricia Rodríguez-Dapena has been a software engineer in the software engineering and standards section of the European Space Agency since 1992. Her main interest is software engineering and verification for software in safety critical systems; she is also interested in software engineering standardization and in software process modeling, implementation, and software process assessment models. She has been employed in the software industry since 1987 and has worked as a programmer, analyst, and researcher.

Rodríguez-Dapena received an MS in computer science from the Polytechnical University of Madrid (UPM), where she is currently preparing her PhD.

Readers may contact Rodríguez-Dapena at ESA/ESTEC, Keplerian 1, 2200AG Noordwijk, The Netherlands; e-mail parodrig@estec.esa.nl.