About Safety Critical Software

HOT TOPICS

Artisan – Aonix merger

atego

Atego launches as the foundation for an integrated tool chain strategy

Atego Newsletter

Sign up to receive the Atego Newsletter with information about new technologies, customer experiences, partner highlights, and product information.
>> Subscribe Now!

DETAILS

Atego Management
International Distributors
Career Opportunities

Need to Dialogue with Aonix?

- General Contact Info Form
- Consulting Services Info Form
- Education Services Info Form

Subscribe to our EMail List

Introduction

Software now pervades almost every aspect of daily life. Transport systems depend on software for control of vehicles and their infrastructure. Financial institutions rely on software for accounting and the transfer of money. Industrial software controls equipment and manufacturing processes. Hospitals depend on software for managing patient records and for control of life-support systems. The use of software has grown dramatically over the last decade with the availability of low-cost, high-performance hardware. It is clear that the safety of much human life and property depends directly or indirectly upon the correctness and deterministic properties of software.

Software can provide users with considerable operational flexibility. However, this flexibility brings with it a greatly increased chance of error. There is now an increasing awareness that strict control is needed in order to reduce the risks of errors in what has come to be called safety critical software—that is, software systems whose failure may lead to loss of life or severe injury.

As a result, there is a growing concern in all major industrial nations regarding the legal and ethical obligations of companies and their officers to ensure that systems do not violate safety regulations.

Many industries are in the process of setting, or have already set, specific standards for the development, testing, and certification of safety critical software. As these standards emerge, the focus is on the use of best practice. In some areas, standards mandate specific techniques for the development of safety critical systems. In all cases, a reasoned justification for the techniques actually used is required, together with evidence to show that the life cycle development processes are being followed.

Example of Failure

A passenger airplane is circling in a prearranged location off the coast of Florida. The landing is delayed because of bad weather conditions. As the plane is banking into a turn, a sudden updraft causes the plane to roll much faster than the software control system expects. The software "assumes" a glitch, and the computers are set into an automatic reboot process. The pilot looks on with horror as all of the navigation displays turn blue with a white line through them. At a most crucial moment, when the pilot needs information to stabilize the aircraft, the computers are performing memory checks and restarting the display software.

Fortunately, the pilot has enough height and time to fly the plane by "feel" until the displays are functioning correctly. Had this error occurred when the plane was landing, the consequences could have been catastrophic.

What is Safety?

MIL-STD 882B (1984) defines safety as follows:

Freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property.

The terms safety, reliability, security, and correctness should not be confused. Leveson [Leveson 86] defines the differences among these

terms as follows:

In general, reliability requirements are concerned with making a system failure free, whereas safety requirements are concerned with making it mishap free . . .

Reliability is concerned with every possible software error, whereas safety is only concerned with those that result in actual system hazards . . .

Even if all failures cannot be prevented, it may be possible to ensure that the failures that do occur are minor consequences, or that even if a potentially serious failure does occur, the system will, "fail safe" . . .

Unfortunately, in many complex systems, safety and reliability may imply conflicting requirements, and thus a system cannot be built to maximize both . . .

Software is a set of instructions and data that makes a general-purpose computer into an application specific one. Software itself is neither safe nor unsafe. However, if software controls the functionality of a safety critical system, then it becomes safety critical software.

A study of system safety is described in "Safeware - System Safety and Computers" by Nancy Leveson [Leveson 95]. An extensive discussion of safety critical systems, which are usually also real-time control systems, is found in the reference [Pyle 91].

Criticality Levels

Most standards assign a criticality level to a system based on the severity of a potential catastrophe and the probability of its occurrence. These are then mapped onto categories for the system criticality levels. If software controls these safety critical systems, then the software too is assigned a criticality level. The software criticality levels correspond to the failure conditions that would result if the software were to fail.

The Federal Aviation Administration (FAA) recognizes five categories of failure conditions and five software-level definitions.

Categories of Failure

Failure Condition	Software Level
Catastrophic	Level A
Hazardous/Severe - Major	Level B
Major	Level C
Minor	Level D
No Effect	Level E

In practice, the differences between levels A and B are small:

- Certain objectives of the software design process must be independently verified.
- Source code accuracy, consistency, and compliance with the software architecture must be independently verified.
- Robustness of object code with low-level requirements must be independently verified.
- Test coverage of software structure (modified condition/decision) must be satisfied independently for level A, and is optional for level B and lower.

The Motor Industry Software Reliability Association (MISRA) classifications match integrity levels using five categories of controllability. [MISRA]

The new IEC 61508 standard uses four software integrity levels, and IEC 880 also uses four levels. [IEC 61508]

There is a strong correlation between the way software can contribute to a potential hazard and the severity level. The level assigned to some software has a great influence upon the rigor with which the software is developed and verified and the evidence that must be collected to confirm this.

Standards

Avionics - RTCA / EUROCAE

The avionics industry has taken the lead in the development of safety certification standards for computer programs.

The Radio Technical Commission for Aeronautics (RTCA), a nonprofit organization formed in 1935, has been instrumental in shaping the future of aviation through the application of electronics and telecommunications. The RTCA operates as a federal advisory committee that is composed of industry and government representatives. One of the key activities undertaken by the RTCA is the development and publication of guidance documents and minimum operational performance standards for avionics technology.

Although the acronym remains the same, RTCA now represents "Requirements and Technical Concepts for Aviation." The RTCA board of directors and international associates work in association with members of the international aviation community to propose the formation of new committees and provide input to ongoing special committees.

One such committee, SC-167, was responsible for the preparation and revision of standards for the certification of avionics software. Major areas of concern include:

- Documentation
- Integration and production systems issues
- Software development
- Software verification
- Configuration management
- Quality assurance

SC-167 was responsible for the review and revision of the document Software Considerations in Airborne Systems and Equipment Certification (DO-178A). This review resulted in a new document, DO-178B, which was issued in December, 1992. The Federal Aviation Administration (FAA) of the U.S. Department of Transportation issued an Advisory Circular on January 11, 1993, stating that DO-178B may be used as a basis for submission of material required to obtain FAA approval of digital computer software. The document is known as ED-12B in Europe and is used by the Joint Aviation Administration (JAA) as a basis for certification.

As DO-178B was used by industry, it became apparent that parts of the document were unclear or ambiguous. A special committee was formed and tasked to produce guidance materials based on experiences and expertise in the industry. SC-190/WG52 is working on the production of these guidelines initially in four principal areas:

Previously Developed Software -

Examples of topics covered are: use of software developed to different standards; use of software developed as Commercial Off The Shelf (COTS) and re-used in a safety critical application; use of software developed to a lower criticality level and subsequently used on a higher-level application.

Verification -

Topics include: the intent of structural coverage; use of high-level requirements to verify low-level requirements; definition of structural coverage terms, etc.

CNS/ATM -

DO-178B is an Airborne Safety Guideline. Ground-based systems (communication, navigation, surveillance, air traffic management) were not subject to the same safety guidelines in the past. As ground/air systems became more integrated, it was recognized that there would be additional benefit if the safety objectives and techniques for achieving them were harmonized. The ground-based community will use DO-178B for future projects, and additional guidance is under development.

Special Considerations -

A device used successfully on one aircraft type can be adapted to a different aircraft type. The history of successful use could raise the confidence of the device to a level where it could be trusted. The evidence required, the reverification guidelines, and the level of trust are the subject of additional guidance being developed to interpret the special considerations requirement.

The committee continues with the development of the guidance documents, and the intent is that, once approved, the recommendations will become an official interpretation of DO-178B.

ISO 9000

The ISO 9000 guidelines do not address the production of safety critical software, but rather focus on the issue of quality. All certification bodies insist on a quality system that instills confidence in the production of safety critical software. ISO 9000 standards are not sufficient in themselves to satisfy safety standards. However, any company undertaking the ISO 9000 certification process clearly makes a positive statement of intent regarding quality objectives.

Def-Stan 0055

The Procurement of Safety Related Software in Defence Equipment [DS 00-55] was published in August, 1997. It is composed of two parts. Part one covers the requirements and part two provides guidance. Like most others, this standard requires a thorough development and verification process. There is, however, a heavy emphasis on formal approaches to specification, design, verification, etc.

The Def-Stan 0055 standard recognizes that, although desirable, formal methods require special skills and tools, which may not be available to a project in a timely fashion. The representative program manager has a great deal of discretion on the use of formal methods, and how much they should be supplemented with alternative verification techniques.

Development Guidelines for Vehicle-Based Software

The Motor Industry Software Reliability Association (MISRA) is a consortium of automotive and component manufacturers, together with motor industry associations and a university. The MISRA members undertook to research specific issues relating to automotive software. This research resulted in nine detailed reports being published.

These development guidelines cover the software life cycle and offer a different perspective for the needs of the automobile industry. Factors in safety analysis must consider possible hazards associated with human interaction with a system. Driver experience, human reaction times, and attentiveness are some of the factors listed. Also included is the risk compensation factor, where improved safety can lead to more risky behavior.

Several of the reports discuss the use of languages and compilers:

Some safety-critical software pundits deprecate the use of `C' due to its incomplete ISO definition resulting in many aspects of the language being undefined, unspecified or implementation specific. In these aspects, it is viewed as being weaker than assembler. The advent of C++ is regarded by some with horror as the language specification is even weaker than that for 'C' and elements of the compiler are becoming very complex.

Languages recommended for high-integrity applications are ISO Pascal subset, Ada subsets and Modula 2 subsets. -- *Report 1*

... it is normally recommended that strongly typed and structured languages (e.g., Ada or Pascal) should be used, which provide compiler and run-time assistance in the finding of faults, rather than the `flexible' languages (e.g. C or Assembler) which require additional checks to be made by other means. -- *Report 2*

Since the publication of the *Development Guidelines for Vehicle Based Software*, another document entitled *Guidelines for the use of the C language in Vehicle Based Software* has been published [MISRA-C]. In the introductory paragraphs the following statements are made:

Nonetheless, it should be recognised that there are other languages available which are in general better suited to safety critical systems

Examples of languages generally recognised to be more suitable than C are Ada and Modula 2. If such languages could be available for a proposed system then their use should be seriously considered in preference to C.

The MISRA C guidelines catalog language rules. There are 92 required rules and 35 advisory rules. Of the 92 required rules, 70 of them do not apply to Ada. Of the 35 advisory rules, 27 of them do not apply

to Ada. An example of a rule that applies to both C and Ada is "no-recursion." An example of a rule that does not apply to Ada is "do not use leading zeros in numbers to denote octal-based numbers."

Many of the problems addressed by the C guidelines, which require special care and attention by the C programmer, are solved by the Ada language definition and implementation.

IEC 880

In 1986, the International Electrotechnical Commission (IEC) published a standard on Software for Computers in the Safety Systems of Nuclear Power Stations. This IEC 880 standard is applicable to the highly reliable software required for computers used in the safety systems of nuclear power plants for safety functions.

Although no specific language is recommended, guidance is given for the selection of a suitable language based on some common basic rules for safety-system programming languages. For example:

- Languages with a thoroughly tested translator
- The language must be completely and unambiguously defined
- Problem-oriented languages are strongly preferred
- The language should not prohibit:

  - Error-limiting constructs
  - Translation-time type checking
  - Run-time type and array-bound check, and parameter checking

- **Safety Critical Java**
- **Safety Critical Ada**