ISO/IEC JTC 1/SC 22/OWGV N 0116

Safety considerations for programming systems

Date	16 December 2007
Contributed by	Steve Michell
Original file name	safety-discussion-michell.doc
Notes	Closes action item #05-12

Safety considerations for programming systems

submitter : stephen michell, december 2007

1. INTRODUCTION

This paper considers the impact of safety-related standards on the development of TR24772.

2. REFERENCES

a. 178B Software Considerations in Airborne Systems and Equipment Certification
b. IEC 61508-3:1995 Functional safety of electrical / electronic / programmable electronic safety-related systems – Part 3: Software requirements
c. Draft ISO/IEC TR24772
d. A number of other (older) safety standards, such as UK DEF STD 00-55/56, MIL-STD 882-D, IEC 60880.

These have largely been replaced by IEC 61508 or start with 61508 as a basis and specialize the approaches called out in IEC61508.

3. BACKGROUND

To date, the analysis for TR24772 has been dominated by the considerations of the security issues, which rely on fault data and analysis to identify, categorize and rate insecurities.

Safety-related documents, on the other hand, take an analysis, process and verification based approach to development of software in a safety-related development. Much of this comes from the realization that the cost of the fault is so large that it cannot be analysed after the fact but must be predicted and all possible steps taken to avoid the fault.

Many of the issues discussed in safety documents are above the software development level, such as development processes, safety reviews, configuration management. These items are outside the scope of our considerations, but do impact on the choices that are made in the areas of our concern. Consequently, the items identified and discussed below are ones that are related to the issues that are related to the considerations of this document.

4. DISCUSSION

4.1 GENERAL

All of the safety-related standards use the notion that safety of software exists inside a larger safety system, with

- a definition of what safety is
- a definition safety levels, or integrity levels,
- a firm requirement for a safety plan,
- a requirement for a fault analysis of the system being developed / safety functions that include
  - detection of faults
  - recovery from faults to
    - resume normal operations,
    - change to reduced operations or
    - safe termination

4.2 APPLICATION OF SAFETY STANDARDS TO OVERALL SYSTEM

In the software-specific safety documents, the safety standards:

- Express a notion of software development existing within a strong engineering discipline that includes safety at all phases of software development, including project planning, requirements, design, development, validation, rework, change control, installation
- Develop a notion of validation corresponding to each phase of development, and state requirements that each stage of validation must be used for the corresponding development phase.
- Specify a requirement for traceability from the software safety requirements back to the system safety requirements, and between each stage of development
- Express a need to be free from ambiguity at all stages, i.e., a requirement that all requirements, designs, code, validation is clearly expressed in ways that reviewers (with domain-specific and task-specific knowledge but possibly with limited technical knowledge of the processes and mechanisms used) can understand, critique and eventually certify that stage.
- Define a notion of “safety function” that
  - maintain safe internal state(s)
  - detect internal (software) faults
  - specify requirements when safety-related systems must interface with non-safety functions
  - capacity and response time performance
  - interfaces to PES (programmable electronic system)
  - interfaces to hardware
- Connect integrity level with sets of techniques that must be applied to each level.

4.2 APPLICATION OF SAFETY STANDARDS TO OVERALL SYSTEM

- Require that software that is to implement both safety and non-safety functions, then all of the software shall be treated as safety-related, unless adequate independence between the functions can be demonstrated in the design. (similar statement for levels)
- Requirements for language
  - certificate of validation, require international std, assessed for fitness for purpose
  - unambiguously defined, or show use of unambiguous subset
    - This could be read to apply to a single instance, but then other requirements needed to handle revision and interfaces so that there is a validation process in place when a

revision of the tool set occurs.
    - features to facilitate detection of programming mistakes
- Specify a need for a coding standard to avoid unsafe language features, and to simplify the language and constructs used to aid review.
- Specify a clear need for review at every stage
- Specify a need for testing
- Significant discussion of Verification at all levels. Includes testing, human review, possibly static analysis and formal analysis. Specific considerations for safety include:
    - Code verification stipulates static verification
    - Data verification includes completeness, self-consistency, but includes **protection against alteration and corruption**
        - also completeness and **correctness**
    - correctness, completeness of initial values, no corruption
    - Failure detections

DO178-B puts a much on reviews and traceability, such as traceability between source code and object code, and ensuring that no disabled code is present. Also addresses partitioning of software.

4.3 APPLICABILITY TO SOFTWARE INTEGRITY LEVELS

Every safety standard has a notion of safety level. IEC 61508 has 4 SILs, with 4 being the highest. All safety standards apply increasingly stringent approaches as the safety integrity level increases. At the highest levels, they usually recommend complete static verification and formal verification techniques, and languages and tools that support those techniques.

5. APPLICABILITY to TR24772

Many of the error categories that we are discussing have direct applicability, and the discussion as to severity and avoidance has direct applicability. How the applicability is applied is impacted by SILs. At the highest SILs, many of these flaws must be shown to absent from the system in question. This will lead to formal statements that the software under development will contain no implementation-defined behaviours, no unspecified behaviours, and so on; in other words the range of permissibility and interpretation will be reduced and and all rules will be applied in their absolute.

For our document, it is beyond our scope to get into a discussion of the appropriateness of languages for various SILs, except to say that it is highly improbable that any standardized language or general purpose language will be suitable as-is, since they all contain unspecified behaviours and implementation dependencies. We should clearly note, however, that some languages have stronger support for static analysis and are more rigourously specified, and therefore have larger subsets which can likely be used in the intended way.

It is likely that that any security-aware or safety-aware development will attempt to avoid the insecurities identified by the document. The difference for safety-related development is that more formalism will be applied to the certification process, and this will force formalisms in language selection and use, coding standard development and application, review processes, and other processes such as management, configuration control, etc.

We may be able to handle the safety considerations by adding an additional subsection that addresses how the rigidity of the safety view applies to each feature.