White Paper

An Introduction to Safety Critical Systems

This paper provides an introduction to the development of software for safety critical systems. It is aimed to serve as a tutorial for developers who are new to the development of software for safety critical systems, discussing the issues involved, introducing some of the techniques available to developers, and providing an overview of how QA Systems' tool, Cantata, can be used to assist with the development of software for safety critical systems.

Contents

Copyright Notice

1 What is a Safety Critical System?

A safety critical system is a system where human safety is dependent upon the correct operation of the system. The emphasis of this paper is on the software element of safety critical systems, which for convenience, is often referred to as safety critical software. However, safety must always be considered with respect to the whole system, including software, computer hardware, other electronic and electrical hardware, mechanical hardware, and operators or users, not just the software element.

Safety critical software has been traditionally associated with embedded control systems. As awareness regarding how systems can impact safety has developed, the scope of safety critical software has expanded into many other types of systems. These range from aircraft fly-by-wire control systems and railway signalling systems, to software in medical devices and traffic lights!

Many everyday systems can have an impact on safety and, as a result, responsible engineers should be asking three key questions about every system that they develop:

> Can it present hazards to safety?
> What can we do to reduce hazards to an acceptable level?
> How can we verify that the developed system is safe?

Collectively, the documented process of answering these questions and applying our answers to the system development is referred to as the **safety case**. A well-conceived and executed safety case is a key element in bringing a safety critical system into use. In areas which have been traditionally concerned with safety critical systems, such as the aviation industry and the nuclear industry, a certification body will have to be convinced that a system is safe before a system can enter use. In some other areas, users have their own safety monitoring groups. Nevertheless, the vast majority of software safety is entirely in the hands and conscience of the software developers and suppliers.

2 Integrity Levels and Standards for Safety Critical Systems

The first step in developing a system is performing a preliminary hazard analysis, to determine whether the system could present a hazard to safety. If the answer is no, then hazard analysis need go no further, other than to periodically review the validity of this decision. If the answer is yes, we must conduct a more detailed hazard analysis:

> How likely is it that an error in the system will result in a particular hazard?
> How likely is the hazard to actually cause an accident?
> What is the likely magnitude of the accident in terms of injuries or deaths?

The collective results of these questions must be weighed against an ethical judgement; what is an acceptable level of safety? A reasonable starting position is that a software based system should be at least as safe as any system it replaces.

While it is good practice to make all software as reliable as possible, not all errors will lead to safety hazards. To illustrate this point, consider a medical system providing diagnostic information. If the software failed by giving a misleading output, a patient could be given the wrong medication. However, if it failed by crashing the computer, users would avoid making such a mistake. One error presents a direct hazard to safety, whilst the other error affects system availability, but not safety.

The concept of 'safety critical' is not absolute; failure of some systems will not impact safety, failure of other systems could occasionally result in minor injuries, and failure of some systems could lead to disasters. The level of safety integrity required varies from none through to a very high level of integrity.

Standards for safety critical software (see below) have now standardised on a scale of five discrete levels of safety integrity, with an integrity level of 4 being 'very high', down to a level of 0 for a system which is not safety related. The term 'safety related' is used to collectively refer to integrity levels 1 to 4. Further analysis will assign an integrity level to each component of a system, including the software.

Standard	Description
Quality Systems - Model for Quality Assurance in Design/Development, Production, Installation and Servicing. ISO9001/EN29001/BS5750 part 1	This is the recommended minimum standard of quality system for software with a safety integrity level of 0, and an essential prerequisite for higher integrity levels.
Functional Safety : Safety Related Systems IEC1508	A general standard, which sets the scene for most other safety related software standards.
Railway Applications: Software for Railway Control & Protection Systems. EN50128	A standard for the railway industry.
Software for Computers in the Safety Systems of	A standard for the nuclear industry.

Nuclear Powers Stations. IEC880	
Software Considerations in Airborne Systems and Equipment Certification. RTCA/DO178B	A standard for avionics and airborne systems.
MISRA Development Guidelines for Vehicle Based Software	Issued by the Motor Industry Software Reliability Association for automotive software.
Safety Management Considerations for Defence Systems Containing Programmable Electronics. Defence Standard 00-56	A standard for the defence industry.
The Procurement of Safety Critical Software in Defence Equipment. Defence Standard 00-55	Detailed software standard for safety critical defence equipment.

Table 1 – Relevant Standards

Differing constraints are placed on the methods and techniques used through each stage of the development lifecycle, depending on the required level of safety integrity. For example, formal mathematical methods are ‘highly recommended’ by most standards at integrity level 4, but are not required at integrity level 0 or 1. The required integrity level can consequently have a major impact on development costs, making it important not to assign an unnecessarily high integrity level to a system or any component of a system. This is not just limited to deliverable software. The integrity of software development tools, test software and other support software may also have an impact on safety.

3 Specification and Design Safety Critical Systems

The general principle when designing any safety critical system is to keep it as simple as possible, taking no unnecessary risks. There is less that can go wrong with a simple system. From a software point of view, this usually involves minimising the use of interrupts and minimising concurrency within the software. Use of concurrency also necessitates the use of an executive or run time environment to manage the concurrency, which is unlikely to have been produced to safety critical standards. Ideally, a safety critical system requiring a high integrity level would have no interrupts and only one task. However, this is not always achievable in practice.

There are two distinct philosophies for the specification and design of safety critical systems:

> To specify and design a 'perfect' system, which cannot go wrong because there are no faults in it, and to prove that there are no faults in it.

> To aim for the first philosophy, but to accept that mistakes may have been made, and to include error detection and recovery capabilities to prevent errors from actually causing a hazard to safety.

The first of these approaches can work well for small systems, which are sufficiently compact for formal mathematical methods to be used in the specification and design, and for formal mathematical proof of design correctness to be established. However, formal mathematical specification of larger and more complex systems is difficult, with human error in the specification or proof becoming a significant problem. Tools to assist with formal mathematical methods are now becoming available, which will enable the effective application of formal mathematical methods to larger systems in the future.

The second philosophy, of accepting that no matter how careful we are in developing a system, that it could still contain errors, is the approach more generally adopted. This philosophy can be applied at a number of levels:

> Within a routine, to check that inputs are valid, to trap errors within the routine, and to ensure that outputs are safe.

> Within the software, to check that system inputs are valid, to trap errors within the software, and to ensure that system outputs are safe.

> Within the system, as independent verification that the rest of the system is behaving correctly, and to prevent it from causing the system to become unsafe. The safety enforcing part is usually referred to as an interlock or protection subsystem.

Software at higher integrity levels is more expensive to develop. A cost conscious designer may consequently try to isolate functionality requiring a high integrity level from other less important functions. Such an approach is essential if is intended to use off-the-shelf software, such as a database manager, because the vast majority of off-the-shelf software products can only be attributed an integrity level of 0.

You can only have different integrity levels for different parts of a system or its software, if it can be proven that lower integrity parts cannot violate the integrity of higher integrity parts or the overall system. Consequently, split levels of integrity are usually only viable when used between processors in a system, and not within the software executing on a single processor.

Figure 1 shows a system in which control and protection has been implemented on separate processors, enabling control software to be implemented to a lower integrity level.

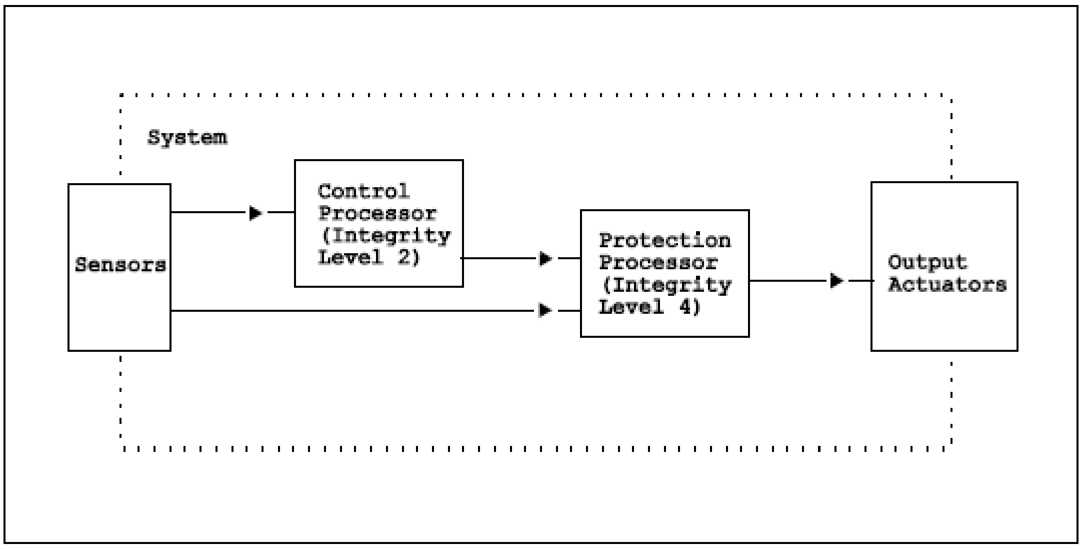
System
Sensors
Control Processor (Integrity Level 2)
Protection Processor (Integrity Level 4)
Output Actuators

Figure 1 – Split Integrity Levels between a Control and Protection System

Whatever philosophy for the specification and design of a safety related software you adopt, basic quality management principles apply, as required by the ISO9001 standard. These include defined procedures for each stage of the lifecycle, records and documentation, configuration management and quality assurance.

4 Verification of Safety Critical Systems

Verification is the most important and most expensive group of activities in the development of safety critical systems, with verification activities being associated with each stage of the development lifecycle. The actual activities are basically the same as a conscientious developer would apply to any other software development, but with a more formal and rigorous approach.

An added complication is that independent verification is usually required. The means by which this is achieved depends upon the integrity level and the user or certification body. Independent verification can vary from independent witnessing of tests, participation at reviews and audit of the developer's verification, to fully independent execution of all verification activities. The degree of independence can vary from a separate team within the overall project structure, to a verification team supplied by a completely independent company, who may never even meet the development team.

Irrespective of how much **independent verification** there is, developers should not forget their own verification work. Independent verification is an addition to verification conducted by developers, not a substitute for it. A developer's objective should be to come as close as possible to passing independent verification first time, with only minor corrections to make.

Tools used in the development of safety critical software can influence the safety integrity of the software under development. This can be a problem, because most tools are only developed to integrity level 0. Care must therefore be taken during verification to ensure that tools such as compilers and CASE tools have not introduced errors into the software under development.

Care must also be taken to ensure that faults in verification tools do not give an incorrect ‘pass’ result to any verification activity, when the results should have been a ‘fail’. Some verification tools have been developed with this requirement in mind, using safety critical standards during the development of the tool.

Taking an ISO9001 accredited quality management system as a baseline, let's look at how the individual verification activities may vary. **Reviews** become more formal, including techniques such as detailed walkthroughs of even the lowest level of design. The scope of reviews is extended to include safety criteria. If formal mathematical methods have been used during the specification and design, formal mathematical proof is a verification activity.

Static analysis is the analysis of program source code before it is executed. If static analysis is conducted at all for non-safety critical software, it is usually limited to checking coding standards and gathering metrics. For safety critical software, more complex static analysis techniques such as control flow analysis, data flow analysis, and checking the compliancy of source code with a formal mathematical specification can be applied.

Dynamic testing is the mainstay of verification, extending from the testing of individual units of code in isolation from the rest of the software, through various levels of integration, to system testing. For safety critical software, dynamic test results are only valid in the target environment. You can, however, develop dynamic tests in the convenience of a host environment, then repeat fully developed tests in the target environment.

> Safety critical software standards require techniques such as equivalence partitioning, boundary value analysis, and structural testing to specified levels of dynamic coverage. **Dynamic coverage** levels required when testing to safety critical standards are much more rigorous than simple statement coverage or decision coverage. For example RTCA/DO178B requires modified decision coverage, where a developer must show that every element of a Boolean expression can independently affect the outcome of the expression.

Most developers provide traceability between system requirements and system test cases, to ensure that all requirements have been implemented and tested at the system level. For safety critical software, the level of detailed traceability required is much higher. Traceability of requirements extends through all levels of design, into individual units of code, and into all levels of testing, through to individual test cases.

5 Using Cantata for Safety Critical Systems Development

QA Systems' tool, Cantata, dynamically proves code with intelligent unit and integration testing, in the most cost effective manner. It provides a complete test development environment, built on Eclipse, and it integrates easily with developer desktop compilers and embedded target platforms. Cantata has been successfully used by customers worldwide since the 1990s to meet the main international safety-related standards, including: ISO 26262, EN 50128, IEC 60880, DO 178B/C and IEC 62304.

Cantata provides tool support for dynamic testing and coverage analysis, to the levels of functionality required by standards for safety critical software development. The tool provides repeatability of tests and portability of test scripts between host and target environments. It can be used in both analysis mode and in non-analysis mode in both the host environment and in the target environment. A suggested strategy is:

a) Use the Cantata instrumentor (in the host environment) to perform static analysis and to prepare an instrumented version of the software under test for dynamic (coverage) analysis.

b) Execute tests in analysis mode in the host environment, using instrumented software. Ensure that coverage objectives are achieved. Correct any errors in the software under test and in the test script.

c) Verify that coverage is achieved in the target environment, by executing tests in analysis mode in the target environment.

d) Verify that the instrumentation and environment has not influenced the outcome of tests, by repeating tests in non-analysis mode in the target environment, using uninstrumented software.

The portability of Cantata test scripts between the host and target environments has been specifically designed to enable tests to be fully developed in the host environment, and then simply repeated in the target environment.

5.1 Cantata Certification

Cantata was recently certified as 'usable in the development of safety related software', up to the highest safety integrity levels for all the main safety related standards:

> IEC 61508:2010 (general industrial)
> ISO 26262:2011 (automotive)
> EN 50128:2011 (railways)
> IEC 60880:2006 (nuclear power)
> IEC 62304:2006 (medical devices)

Certification was performed by SGS-TÜV Saar GmbH, an independent third party certification body for functional safety, accredited by Deutsche Akkreditierungsstelle GmbH (DAkkS).

Cantata users are provided with a free Tool Certification Kit, consisting of: independent certification for the latest Cantata versions, a detailed Safety Manual, comprehensive standards briefings and automated tests demonstrating the correct tool operation on embedded targets.

5.2 Free Cantata Evaluation

If you want to find out more on how Cantata can accelerate and improve your software development, why not request a free trial? This can be requested from the QA Systems website or by emailing sales@qa-systems.com directly.

6 Conclusion

Safety critical software is a complex subject. This paper has necessarily had to be brief and superficial. The main objective of the techniques discussed in this paper is to minimise the risks of implementation errors. The premise is that errors could result in system behaviour that causes a hazard to safety.

Although safety critical systems have been in use for many years, the development of safety critical software is still a relatively new and immature subject. New techniques and methodologies for safety critical software are a popular research topic with universities, and are now becoming available to industry. Tools supporting the development of safety critical software are now available, making the implementation of safety critical standards a practical prospect.

Suitably trained and experienced engineers are key to the success of any software development. In the development of safety critical software, the risk of an inexperienced or untrained engineer making an error which could then lead to an accident must be avoided.

Even if you are not developing safety critical software, the reliability of your systems may benefit from adopting some of the techniques and methodologies discussed in this paper. Above all, the best way to minimise risk, both to safety, reliability and to the timescale of a software development project, is to keep it simple.