Risktec

RISKworld issue 15 spring 2009 pg 5 the newsletter of risktec solutions limited

An Introduction to Safety-Critical Software

Why Software is Different

Software is often used to implement the functionality of safety systems because it can be designed to handle complex functionality, is accurate and repeatable, and can be cheaper than hardware solutions. However, there are many examples of safety systems which have failed due to software related faults, a small sample of which are presented in Box 1.

Box 1 - Examples of software system failure

Therac-25 (1985 to 1987)
Therac-25 radiation therapy machines delivered radiation overdoses to a number of patients in Canada and the USA which resulted in three fatalities. Most of the Therac-25 safety interlocks were software based, which replaced hardware interlocks that had operated without any recorded patient injuries on earlier versions of these machines.

Ariane 5 (1996)
The unmanned Ariane 5 European spacecraft was destroyed less than a minute after launch on its maiden flight, due to a fault with software previously used successfully on earlier versions of the launcher.

Mars Climate Orbiter (1999)
A mismatch between Imperial and SI units on the NASA Mars Climate Orbiter resulted in loss of the spacecraft when it entered the Martian atmosphere too low and too fast.

UK Inland Revenue (2004)
Software errors in the UK Inland Revenue tax credit payment system contributed to a $3.54 billion tax credit overpayment.

How software fails

The failure of a safety system based entirely on "hardwired technology" tends to be dominated by so called random failures, which are typically age or wear related, as opposed to software based systems, which fail predominantly due to systematic errors. This distinction arises because systematic errors can often be identified and removed from a hardwired design, whereas this can be much more difficult with software due to a greater level of design complexity and its abstract nature. Moreover, software does not wear out and so does not fail randomly in the same sense as hardware (although the platform upon which the software runs will be subject to random failure mechanisms).

The factors that can lead to a software error, which if triggered can cause a system level failure, are peculiar to systematic errors, both in terms of their introduction and detection [see Box 2].

Safety assurance processes

The uniqueness and complexity of software based safety systems means that there can be a huge array of factors influencing the success or failure of such developments. Fortunately, there are some steps which are generally effective at reducing the risks associated with developing software safety systems.

These steps revolve around safety assurance, i.e. the planning, development, verification and configuration management processes that ensure the software meets its safety objectives. Key steps include:

- Explicitly identify all safety functional and integrity requirements before commencing the software design phase, as mistakes or omissions will be more difficult and expensive to rectify the later they are discovered and significant software modifications can be a major cause of systematic error.
- Identify at the outset the means of generating the evidence to show that each safety requirement has been met, to inform the design process, ensuring that the necessary evidence is produced as the software is developed (since retrospective generation is usually very expensive).
- Confirm the availability of safety assurance evidence when considering integrating previously developed software components, in order to reduce cost and project risk.
- Minimise the number of personnel developing the software system and ensure all interfaces are well defined. Increasing the number of personnel in order to shorten the development timescale will increase the number of interfaces, potentially leading to a greater number of errors.
- Consider the adequacy of generic safety assurance evidence for commercial off-the-shelf components (e.g. electrical protection relays, PLC shutdown systems) in the context of the safety system within which it will be deployed, since for example a system failure causing a valve to close could be safe in one system but may result in a disastrous over-pressurisation event in another.

Conclusion

Identifying software errors in safety systems is not easy, but the application of targeted safety assurance processes should help manage the associated risks to an acceptable level.

Box 2 – Typical factors in software failure

- Poor communication among software developers and end users
- Lack of operational or safety experience in programmers
- Use of inappropriate programming language
- Unwanted functionality supplied as part of commercial-off-the-shelf software
- Inadvertent change to safety functionality during software modification
- Unrepresentative software testing
- Inability to fully test all logic paths due to complexity and number of variables (e.g. timing of inputs)
- High frequency of software updates which may adversely affect safety function

UK Principal Office
Wilderspool Park
Greenall's Avenue
Warrington WA4 6HL
United Kingdom
Tel +44 (0)1925 611200
Fax +44 (0)1925 611232

Other UK Offices
Aberdeen
Ashford
Edinburgh
Glasgow
London

Middle East
Dubai
Muscat

North America
Calgary
Houston

For further information, including office contact details, visit:
www.risktec.co.uk
or email:
enquiries@risktec.co.uk